VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MATHEMATICAL MODELING (CO2011)

Assignment

"The SIR Model in COVID-19 prediction"

Advisors: Nguyen An Khuong
Nguyen Tien Thinh

Candidates: Phan Thien Phuc – 1852670
Nguyen Quang Phuc – 1852668
Vu Minh Quang – 1852699
Nguyen Hoang Viet – 1850059

HCMC, July 2020

Contents

1 The SIR Model - Overview

1.1 Detailed Introduction and Construction

The SIR model is a simple mathematical model describing how infectious disease, which was first introduced by *Kermack and McKendrick*. The formula consists a system of differential equations dependent on time, representing three compartments of the population, with three state of disease:

- SUSCEPTIBLE - People are able to infect the disease,
- INFECTIOUS - people who are infected and can spread the disease to community,
- RECOVERED - people get immunity or death so that they are not susceptible to the same illness anymore.

In the model, we also assume that who recovered from the disease will be immune to it in the future and the total population does not change in time. The model is as follows.

$$\frac{dS}{dt} = -\frac{\beta}{N} IS \tag{1}$$
$$\frac{dI}{dt} = \frac{\beta}{N} IS - \gamma I \tag{2}$$
$$\frac{dR}{dt} = \gamma I \tag{3}$$

where at the time $t \geq t_0 \geq 0, t_0$ is the first time when an infection of the disease is reported,

- $S(t)$ The number of people who are susceptible to the disease;
- $I(t)$ The number of infected people;
- $R(t)$ The number of recovered people;
- $\beta(t)$ The contact rate between the susceptible compartment and the infectious compartment;
- $\gamma(t)$ The recovery rate when a person is infected;
- $N(t)$ Total population and it is defined as the sum of the three compartments

$$N(t) := S(t) + I(t) + R(t) \tag{4}$$

- Equation (1) represents the decrease in time of the susceptible compartment. The decrease rate can be seen as the probability of the event that a susceptible individual is infected when he or she interacts with infected people.
- Equation (2) represents the change in time of the infectious compartment. It is obtained by subtracting the number of the new recovered people with the recovery rate γ from the number of the infected people and by adding the number of the new infected people to the number of the infected people;
- Equation (3) represents the increase in time of the recovered compartment. It is the number of the new recovered people with the recovery rate γ.

Some things to notice:

- The total population $S + I + R$ is constant because $\frac{dS}{dt} + \frac{dI}{dt} + \frac{dR}{dt} = -\frac{\beta}{N} IS + \frac{\beta}{N} IS - \gamma I + \gamma I = 0$
- If $I = 0$, i.e. there are no infectives, the right sides of all three equations are 0, so nothing changes. To make matters interesting, we must start with some infectives.

SIR model allows us to describe the number of people in each compartment with the ordinary differential equation. β is a parameter controlling how much the disease can be transmitted through exposure. It is determined by the chance of contact and the probability of disease transmission. γ is a parameter expressing how much the disease can be recovered in a specific period. Once the people are healed,

they get immunity. There is no chance for them to go back susceptible again.
We do not consider the effect of the natural death or birth rate on the population because the model assumes the period of the disease is much shorter than the lifetime of the human. It lets us know the importance of knowing two parameters, β and γ. When we can estimate the two values, there are several insights derived from it. If the D is the average days to recover from infectious, it is derived from γ.

$$D = \frac{1}{\gamma} \tag{5}$$

Also, we can estimate the nature of the disease in terms of the power of infection.

$$R_0 = \frac{\beta}{\gamma} \tag{6}$$

It is called a **basic reproduction number**. R_0 is the average number of people infected from one other person. If it is high, the probability of pandemic is also higher. The number is also used to estimate the herd immune threshold (HIT). If the basic reproduction number multiplied by the percentage of non-immune people (susceptible) is equal to 1, it indicates the balanced state. The number of infectious people is constant. Assume the proportion of immune people is p, the stable state can be formulated as follows.

$$R_0(1-p) = 1 \Leftrightarrow 1 - p = \frac{1}{R_0} \Leftrightarrow p_c = 1 - \frac{1}{R_0} \tag{7}$$

Therefore, p_c is the HIT to stop the spread of the infectious disease. We can stop the outbreak by vaccinating the population to increase herd immunity.
The typical time between contacts is $T_c = \beta^{-1}$, and the typical time until removal is $T_r = \gamma^{-1}$. From here it follows that, on average, the number of contacts by an infectious individual with others before the infectious has been removed is:$\frac{T_r}{T_c}$ The role of both the **basic reproduction number** and the initial susceptibility are extremely important. In fact, upon rewriting the equation for infectious individuals as follows:

$$\frac{dI}{dt} = (R_0\frac{S}{N} - 1)\gamma I,$$

It yeilds that if:

$$R_0.S(0) > N,$$

then

$$\frac{dI}{dt}(0) > 0,$$

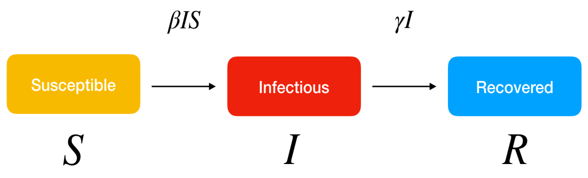
i.e., there will be a proper epidemic outbreak with an increase of the number of the infectious (which can reach a considerable fraction of the population). On the contrary, if

$$R_0.S(0) < N,$$

then

$$\frac{dI}{dt}(0) < 0,$$

i.e., independently from the initial size of the susceptible population the disease can never cause a proper epidemic outbreak. As a consequence, it is clear that both the basic reproduction number and the initial susceptibility are extremely important.



Transition rates

For the full specification of the model, the arrows should be labeled with the transition rates between compartments. Between S and I, the transition rate is assumed to be $\frac{d(\frac{S}{N})}{dt} = \frac{-\beta SI}{N^2}$, where N is the total population, β is the average number of contacts per person per time, multiplied by the probability of disease transmission in a contact between a susceptible and an infectious subject, and $\frac{SI}{N^2}$ is the fraction of those contacts between an infectious and susceptible individual which result in the susceptible person becoming infected.
Between I and R, the transition rate is assumed to be proportional to the number of infectious individuals which is γI. This is equivalent to assuming that the probability of an infectious individual recovering in any time interval dt is simply γdt. If an individual is infectious for an average time period D, then $\gamma = \frac{1}{D}$. This is also equivalent to the assumption that the length of time spent by an individual in the infectious state is a random variable with an exponential distribution. The "classical" SIR model may be modified by using more complex and realistic distributions for the I-R transition rate.
For the special case in which there is no removal from the infectious compartment ($\gamma = 0$), the SIR model reduces to a very simple SI model, which has a logistic solution, in which every individual eventually becomes infected.

2 Exercises Part

2.1 Exercise 1

2.1.1 Discrete SIR Model

Assumptions: Assume that a type of flu is spreading within a community. We also assume that

- No one enters or leaves the community, and there is no contact outside the community.
- Each person is susceptible S (able to catch this new flu); infected I (currently has the flu and can spread the flu); or removed R (already had the flu and will not get it again, which includes death).
- Initially, every person is either S or I.
- Once someone gets the disease this year, they cannot get the disease again.
- The average length of the disease is 2 weeks, over which time the person is deemed infected and can spread the disease.
- Our time period for the model will be per week.
- The community is isolated.
- The recovery time of an infected individual is exactly 2 weeks and it does not change in time;
- Who recovered from the flu will be immune to it in the future;
- A susceptible individual becomes an infected individual with constant rate ($\frac{\beta}{N}$). We also assume that the rate does not change in time.

Let's assume the following definitions for our variables:

- $S(n)$ = number in the population susceptible after **period** n
- $I(n)$ = number infected after **period** n
- $R(n)$ = number removed after **period** n

Considering R(n), our assumption for the length of time someone has the flu is 2 weeks. Thus, 1/2 or 50% of the infected people will be removed each week:

$$R(n+1) = R(n) + \gamma I = R(n) + 0.5I$$

The value $\gamma = 0.5$ is the *removal rate per week*.It represents the proportion of the infected persons who are removed from infection each week.
I(n) will have terms that both increase and decrease its amount over time. It is decreased by the number of people removed each week: $0.5 * I(n)$. It is increased by the number of susceptible people who come into contact with infected people and catch the disease: $aS(n)I(n)$. We define a as the rate at which the disease is spread, or the transmission coefficient. We realize this is a probabilistic coefficient. We will assume, initially, that this rate is a constant value that can be found from the initial conditions.
Let's illustrate as follows: Assume we have a population of 1000 students residing in the dorms. Our nurse found 5 students reporting to the infirmary initially: I(0) = 5 and S(0) = 995. After one week, the total number infected with the flu is 11. We compute a as follows:

$$\begin{aligned}
I(0) &= 5, I(1) = I(0) - 0.5I(0) + aI(0)S(0) \\
I(1) &= 11 = 5 - 2.5 + a * 5 * 995 \\
a &= 0.001709
\end{aligned}$$

Considering $S(n)$. This number is decreased only by the number that becomes infected.

$$S(n+1) = S(n) - aS(n)I(n)$$

The SIR model is

$$\begin{aligned}
R(n+1) &= R(n) + 0.5I(n) \\
I(n+1) &= I(n) - 0.5I(n) + 0.001709I(n)S(n) \\
S(n+1) &= S(n) - 0.001709S(n)I(n) \\
S(0) &= 995, I(0) = 5, R(0) = 0
\end{aligned}$$

2.1.2 Continuous SIR Model

Assumptions: Assume that a type of flu is spreading within a community. We also assume that

- No one enters or leaves the community, and there is no contact outside the community.
- Each person is susceptible S (able to catch this new flu); infected I (currently has the flu and can spread the flu); or removed R (already had the flu and will not get it again, which includes death).
- Initially, every person is either S or I.
- Once someone gets the disease this year, they cannot get the disease again.
- The average length of the disease is 2 weeks, over which time the person is deemed infected and can spread the disease.
- Our time period for the model will be per week.
- The community is isolated.
- The recovery time of an infected individual is exactly 2 weeks and it does not change in time;
- Who recovered from the flu will be immune to it in the future;
- A susceptible individual becomes an infected individual with constant rate ($\frac{\beta}{N}$). We also assume that the rate does not change in time.

Let's assume the following definitions for our variables:

- $S(t)$ = number in the population susceptible after **time** t
- $I(t)$ = number infected after **time** t
- $R(t)$ = number removed after **time** t

Considering R(t), our assumption for the length of time someone has the flu is 2 weeks. Thus, 1/2 or 50% of the infected people will be removed each week:

$$\frac{dR}{dt} = \gamma I(t) = 0.5I(t)$$

The value 0.5 is called the removal rate per week. The removal rate represents the proportion of the infected persons who are removed from infection each week. If real data are available, then we could do "data analysis" to obtain the removal rate. I(t) will have terms that both increase and decrease its amount over time. I(t) is decreased by the number removed each week, $0.5I(t)$. I(t) is increased by the number of susceptible who come into contact with an infected person and catch the disease, $aS(t)I(t)$. We define the rate a as the rate at which the disease is spread, or the transmission coefficient. We realize this is a probabilistic coefficient. We will assume, initially, that this rate is a constant value that can be found from initial conditions. We use an estimate of 0.001709 for the rate a.
Considering S(t). This number is decreased only by the number who become infected. We may use the same rate a to obtain the model.

$$\frac{dS}{dt} = -\frac{\beta}{N}I(t)S(t) = -0.001709I(t)S(t)$$

Our SIR model is shown in the following systems of differential equations:

$$\begin{aligned}
\frac{dR}{dt} &= 0.5I(t)\\
\frac{dI}{dt} &= -0.5I(t) + 0.001709I(t)S(t)\\
\frac{dS}{dt} &= 0.001709S(t)I(t)\\
S(0) &= 995, I(0) = 5, R(0) = 0
\end{aligned}$$

2.2 Exercise 2 - The RK4 method in solving the SIR system

2.2.1 Preliminary

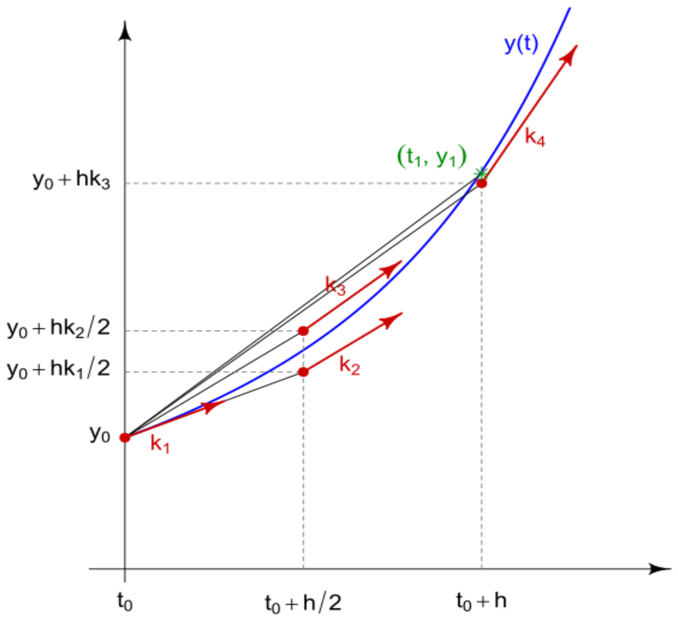
The most widely known member of the Runge–Kutta family is generally referred to as "RK4", the "classic Runge–Kutta method" or simply as "the Runge–Kutta method".RK4 is one of the classic methods for numerical integration of ODE models.
Consider the following initial value problem of ODE

$$\begin{aligned}
\frac{dy}{dt} &= f(t, y)\\
y(t_0) &= y_0
\end{aligned} \tag{8}$$

where y(t) is the unknown function (scalar or vector) which I would like to approximate.
The iterative formula of RK4 method for solving ODE (8) is as follows

$$\begin{aligned}
y_{n+1} &= y_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k4)\\
k_1 &= f(t_n, y_n)\\
k_2 &= f(t_n + \frac{\Delta t}{2}, y_n + \frac{k_1\Delta t}{2})\\
k_3 &= f(t_n + \frac{\Delta t}{2}, y_n + \frac{k_2\Delta t}{2})\\
k_4 &= f(t_n + \Delta t, y_n + k_3\Delta t)\\
t_{n+1} &= t_n + \Delta t\\
n &= 0, 1, 2, 3, ...
\end{aligned} \tag{9}$$

The SIR model is defined as (1), (2), (3). where S(t) is the number of susceptible people in the population at time t, I(t) is the number of infectious people at time t, R(t) is the number of recovered people at time



t, β is the transmission rate, γ represents the recovery rate, and N=S(t)+I(t)+R(t) is the fixed population. According to the general iterative formula (9), the iterative formulas for S(t), I(t) and R(t) of SIR model can be written out.

$$\begin{aligned}
S_{n+1} &= S_n + \frac{\Delta t}{6}(k_1^S + 2k_2^S + 2k_3^S + k4^S) \\
k_1^S &= f(t_n, S_n, I_n) = -\frac{\beta S_n I_n}{N} \\
k_2^S &= f(t_n + \frac{\Delta t}{2}, S_n + \frac{k_1^S \Delta t}{2}, I_n + \frac{k_1^I \Delta t}{2}) = -\frac{\beta}{N}(S_n + \frac{k_1^S \Delta t}{2})(I_n + \frac{k_1^I \Delta t}{2}) \\
k_3^S &= f(t_n + \frac{\Delta t}{2}, S_n + \frac{k_2^S \Delta t}{2}, I_n + \frac{k_2^I \Delta t}{2}) = -\frac{\beta}{N}(S_n + \frac{k_2^S \Delta t}{2})(I_n + \frac{k_2^I \Delta t}{2}) \\
k_4^S &= f(t_n + \Delta t, S_n + k_3^S \Delta t, I_n + k_3^I \Delta t) = -\frac{\beta}{N}(S_n + k_3^S \Delta t)(I_n + k_3^I \Delta t)
\end{aligned} \tag{10}$$

$$\begin{aligned}
I_{n+1} &= I_n + \frac{\Delta t}{6}(k_1^I + 2k_2^I + 2k_3^I + k4^I) \\
k_1^I &= f(t_n, S_n, I_n) = \frac{\beta S_n I_n}{N} - \gamma I_n \\
k_2^I &= f(t_n + \frac{\Delta t}{2}, S_n + \frac{k_1^S \Delta t}{2}, I_n + \frac{k_1^I \Delta t}{2}) = \frac{\beta}{N}(S_n + \frac{k_1^S \Delta t}{2})(I_n + \frac{k_1^I \Delta t}{2}) - \gamma(\frac{I_n + k_1^I \Delta t}{2}) \\
k_3^I &= f(t_n + \frac{\Delta t}{2}, S_n + \frac{k_2^S \Delta t}{2}, I_n + \frac{k_2^I \Delta t}{2}) = \frac{\beta}{N}(S_n + \frac{k_2^S \Delta t}{2})(I_n + \frac{k_2^I \Delta t}{2}) - \gamma(\frac{I_n + k_2^I \Delta t}{2}) \\
k_4^I &= f(t_n + \Delta t, S_n + k_3^S \Delta t, I_n + k_3^I \Delta t) = \frac{\beta}{N}(S_n + k_3^S \Delta t)(I_n + k_3^I \Delta t) - \gamma(I_n + k_3^I \Delta t)
\end{aligned} \tag{11}$$

$$\begin{aligned}
R_{n+1} &= R_n + \frac{\Delta t}{6}(k_1^R + 2k_2^R + 2k_3^R + k_4^R) \\
k_1^R &= f(t_n, I_n) = \gamma I_n \\
k_2^R &= f(t_n + \frac{\Delta t}{2}, I_n + \frac{k_1^I \Delta t}{2}) = \gamma(\frac{I_n + k_1^I \Delta t}{2}) \\
k_3^R &= f(t_n + \frac{\Delta t}{2}, I_n + \frac{k_2^I \Delta t}{2}) = \gamma(\frac{I_n + k_2^I \Delta t}{2}) \\
k_4^R &= f(t_n + \Delta t, I_n + k_3^I \Delta t) = \gamma(I_n + k_3^I \Delta t)
\end{aligned} \tag{12}$$

Note that since the population N = S(t) + I(t) + R(t) is constant, there will have $\frac{dS}{dt} + \frac{dI}{dt} + \frac{dr}{dt} = 0$. Therefore, only two of the three ODEs are independent and sufficient to solve the ODEs. Here, only iterative formulas for S(t) and I(t) are used and R(t) is calculated by S(t)=N - I(t) - R(t).

2.2.2 Implemetation

RK4 SIR Function

Firstly, I need to define some classes in order to make the code easier to imagine:

- **SIRValue** is used to store all the SIR input values including number of suspected, infected and recovered.

```python

# SIR Class
class SIRValue:
    def __init__(self, suspected_num, infected_num, recovered_num):
        self.suspected = (suspected_num, 0)[suspected_num < 0]
        self.infected = (infected_num, 0)[infected_num < 0]
        self.recovered = (recovered_num, 0)[recovered_num < 0]

    def showValue(self):
        suspected_str = str(self.suspected)
        infected_str = str(self.infected)
        recovered_str = str(self.recovered)

        print("Number of Suspected People: " + suspected_str)
        print("Number of Infected People: " + infected_str)
        print("Number of Recovered People: " + recovered_str)

    def getTotalPopulation(self):
        return self.suspected + self.infected + self.recovered

    def getValueByList(self):
        return [self.suspected, self.infected, self.recovered]
```

- **SIRRatio** is used to store all the input ratios including: β - beta(contact rate) and γ - gamma(recovery rate)

```python
class SIRRatio:
    def __init__(self, infected_rate, recovered_rate):
        self.infected = infected_rate
        self.recovered = recovered_rate
```

- **Slop** is used to store the SIRValue for futher calculation in other slops of RK4 (RK4 model needs to calculate 4 slops - $K1, K2, K3, K4$

```python
class Slop:
    def __init__(self, slop_value):
        self.value = slop_value
```

- **SIRRungeKuttaSlop** is used to store all the **Slop**s to calculate the next RK4 value

```python
class SIRRungeKuttaSlop:
    def __init__(self, slop_values):
        self.values = slop_values
```

Secondly, as the given SIR values are initial infected number I_0, initial recovered number R_0 and total population N. I need to implement:

- **calculateSuspected** function to calculate the suspected number:

```python
# SIR Helper Functions
def calculateSuspected(N, sir_value):
    infected_num = sir_value.infected
    recovered_num = sir_value.recovered
    return N - infected_num - recovered_num
```

- Moreover, as working on the helper functions of SIR, I also implement the **printSIRValues** function to print out the result in readable format:

```python
def printSIRValues(sir_values, dt):
    i = 0
    for value in sir_values:
        print("\nWeek " + str(i))
        value.showValue()
        i += dt
```

Next, as RK4 model requires to evaluate the derivative of the value in the period of time Δt:

- Calculate the derivative of Suspected in the period of time by the function: $\frac{dS}{dt} = -\frac{\beta}{N}IS$

```python
def dSdt(t, sir_value, sir_ratio):
    infected_num = sir_value.infected
    suspected_num = sir_value.suspected
    total_population = sir_value.getTotalPopulation()

    #  Infected ratio per person = beta / N
    infected_rate = sir_ratio.infected
    infected_ratio_per_person = infected_rate / total_population

    diffSuspected = -(infected_ratio_per_person *
                      (suspected_num * infected_num))
    return diffSuspected
```

- Calculate the derivative of Infected in the period of time by the function: $\frac{dI}{dt} = \frac{\beta}{N}IS - \gamma I$

```python
def dIdt(t, sir_value, sir_ratio):
    diffSuspected = dSdt(t, sir_value, sir_ratio)
    diffRecovered = dRdt(t, sir_value, sir_ratio)

    diffInfected = -diffSuspected - diffRecovered
    return diffInfected
```

- Calculate the derivative of Recovered in the period of time by the function: $\frac{dS}{dt} = \gamma I$

```python
def dRdt(t, sir_value, sir_ratio):
    infected_num = sir_value.infected
    recovered_rate = sir_ratio.recovered

    diffRecovered = (recovered_rate * infected_num)
    return diffRecovered
```

In the next step, we implement the Runge Kutta model's helper functions in order to calculate the slop more easyly:

- **calculateSlopValues** function is used to evaluate the differential values of the slop

```python
def calculateSlopValues(t, sir_value, sir_ratio):
    dS = dSdt(t, sir_value, sir_ratio)
    dI = dIdt(t, sir_value, sir_ratio)
    dR = dRdt(t, sir_value, sir_ratio)

    return [dS, dI, dR]
```

- **calculateInitialSlop** function is used to evaluate the first slop of RK4 by the given RK4 function: $k_1 = f(t_n, y_n)$

```python
def calculateInitialSlop(time, sir_value, sir_ratio):
    [KS, KI, KR] = calculateSlopValues(time, sir_value, sir_ratio)

    suspected_slop = Slop(KS)
    infected_slop = Slop(KI)
    recovered_slop = Slop(KR)

    K1Slops = [suspected_slop, infected_slop, recovered_slop]

    return SIRRungeKuttaSlop(K1Slops)
```

- **calculateMiddleSlop** function is used to evaluate the second and third slop of RK4 by the given RK4 function: $k_2 = f(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2}k_1)$, $k_3 = f(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2}k_2)$

```python
def calculateMiddleSlop(time, dt, sir_value, sir_ratio, prev_slop):
    #  Get the current sir values
    [suspected_num, infected_num, recovered_num] = sir_value.getValueByList()

    #  Retrieve value from prev slop
    [prev_sus, prev_inf, prev_rec] = [slop.value for slop in prev_slop]

    #  Evaluate next SIR-value for counting the difference
    #  S_next = S_current + (dt * S_prev_slop/2)
    k_suspected_num = suspected_num + (dt*prev_sus / 2)
    k_infected_num = infected_num + (dt*prev_inf / 2)
    k_recovered_num = recovered_num

    k_sir_value = SIRValue(k_suspected_num, k_infected_num, k_recovered_num)

    #  Calculate the slop values
    [KS, KI, KR] = calculateSlopValues(time + dt/2, k_sir_value, sir_ratio)

    suspected_slop = Slop(KS)
    infected_slop = Slop(KI)
    recovered_slop = Slop(KR)

    KSlop_value = [suspected_slop, infected_slop, recovered_slop]

    return SIRRungeKuttaSlop(KSlop_value)
```

- **calculateLastSlop** function is used to evaluate the last slop of RK4 by the given RK4 function: $k_4 = f(t_n + \Delta t, y_n + \Delta t k_1)$

```
def calculateLastSlop(time, dt, sir_value, sir_ratio, prev_slop):
    #  Get the current sir values
    [suspected_num, infected_num, recovered_num] = sir_value.getValueByList()

    #  Retrieve value from prev slop
    [prev_sus, prev_inf, prev_rec] = [slop.value for slop in prev_slop]

    # TODO Refactor this code
    #  Evaluate next SIR-value for counting the difference
    #  S_next = S_current + (dt * S_prev_slop)
    k_suspected_num = suspected_num + (dt*prev_sus)
    k_infected_num = infected_num + (dt*prev_inf)
    k_recovered_num = recovered_num

    k_sir_value = SIRValue(k_suspected_num, k_infected_num, k_recovered_num)

    #  Calculate the slop values
    [KS, KI, KR] = calculateSlopValues(time + dt, k_sir_value, sir_ratio)

    suspected_slop = Slop(KS)
    infected_slop = Slop(KI)
    recovered_slop = Slop(KR)

    KSlop_value = [suspected_slop, infected_slop, recovered_slop]

    return SIRRungeKuttaSlop(KSlop_value)
```

In the last and final step, we need to implement the **Runge Kutta - SIR** model functions to take in the inputs and calculate the logic of the Runge Kutta method:

- **approximateRK4SIR** function is used to evaluate all slops of the Runge Kutta method and all the SIR values in the given period of time Δt.

```
# RungeKutta SIR Model Function
def approximateRK4SIR(sir_value, sir_ratio, t, dt=1):
    suspected_num = sir_value.suspected
    infected_num = sir_value.infected
    recovered_num = sir_value.recovered

    sir_values = []
    curr_sir_value = SIRValue(suspected_num, infected_num, recovered_num)
    sir_values.append(curr_sir_value)

    prev_slop = [None]*3
    KS = [None]*4
    KI = [None]*4
    KR = [None]*4

    # TODO Refactor the code here too
    for i in range(0, t, dt):
        # K1
        slop = calculateInitialSlop(i, sir_values[i], sir_ratio)
        prev_slop = slop.values
        KS[0] = prev_slop[0]
        KI[0] = prev_slop[1]
        KR[0] = prev_slop[2]

        # K2
        slop = calculateMiddleSlop(i, dt, sir_values[i], sir_ratio, prev_slop)
        prev_slop = slop.values
        KS[1] = prev_slop[0]
        KI[1] = prev_slop[1]
        KR[1] = prev_slop[2]

        # K3
        slop = calculateMiddleSlop(i, dt, sir_values[i], sir_ratio, prev_slop)
        prev_slop = slop.values
        KS[2] = prev_slop[0]
        KI[2] = prev_slop[1]
        KR[2] = prev_slop[2]

        # K4
        slop = calculateLastSlop(i, dt, sir_values[i], sir_ratio, prev_slop)
        prev_slop = slop.values
        KS[3] = prev_slop[0]
        KI[3] = prev_slop[1]
        KR[3] = prev_slop[2]

        # Next SIR Value
        suspected_num = calculateNextValue(suspected_num, dt, KS)
        infected_num = calculateNextValue(infected_num, dt, KI)
        recovered_num = calculateNextValue(recovered_num, dt, KR)

        # Add to SIR Array
        curr_sir_value = SIRValue(suspected_num, infected_num, recovered_num)
        sir_values.append(curr_sir_value)

    return sir_values
```

- **RK4SIR** function is used to handle all the inputs and triggers the logic of Runge Kutta function to execute.

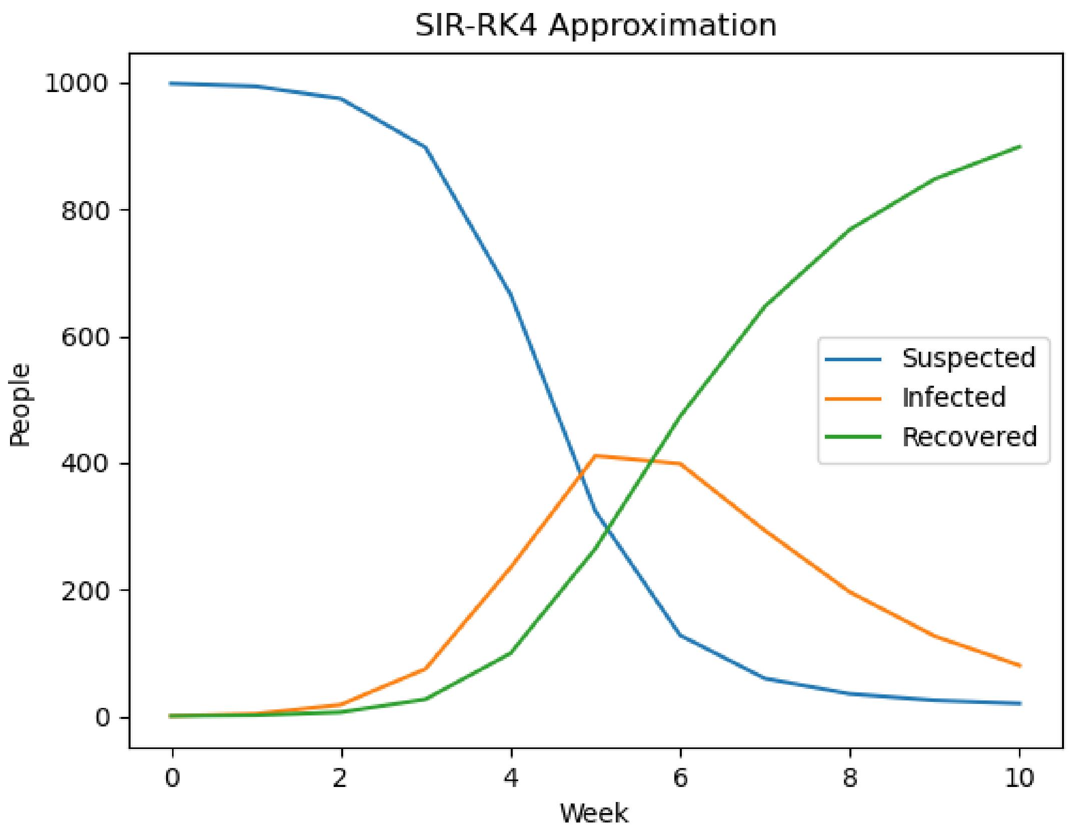
```
def RK4SIR(total_population, I0, R0, beta, gamma, time, dt):
    sir_value = SIRValue(0, I0, R0)
    sir_value.suspected = calculateSuspected(total_population, sir_value)

    sir_ratio = SIRRatio(beta, gamma)

    approximated_values = approximateRK4SIR(sir_value, sir_ratio, time, dt)

    return approximated_values
```

Then to test if the RK4SIR function is actually working we need to implement the ploting machanizm in order to visualize the SIR model using graphic.
This is how we implement the plotting

```
# Testing SIR Model
sir_values = RK4SIR(N, INIT_INFECTED, INIT_RECOVERED, BETA, GAMMA, TIME, 1)
printSIRValues(sir_values, 1)

# Plot Graph
x = []
for i in range(0, TIME + 1):
    x.append(i)

Sus = []
Inf = []
Rec = []
for value in sir_values:
    Sus.append(value.suspected)
    Inf.append(value.infected)
    Rec.append(value.recovered)

```

```python
plt.plot(x, Sus, label="Suspected")
plt.plot(x, Inf, label="Infected")
plt.plot(x, Rec, label="Recovered")

plt.xlabel('Week')
plt.ylabel('People')

plt.title('SIR-RK4 Approximation')

plt.legend()
plt.show()
```

And this is the result of the plot with $\beta = 2, \gamma = 0.5, N = 1000, I_0 = 1, R_0 = 1$



2.3 Exercise 3

2.3.1 Background

The Metropolis–Hastings algorithm can be used to take random samples from any probability distribution. Provied that we know a function f(x) that is atleast propotional to the original probability density function.
In this problem, we will consider taking sample of the parameter β and γ in the SIR model the prior from distribution $\pi(\beta, \gamma)$. Assuming β and γ are indepentdent, we have:

$$\pi(\beta, \gamma) = \pi(\beta).\pi(\gamma)$$

Where $\pi(\beta)$ is the prior distribution of β and $\pi(\gamma)$ is the prior distribution γ
Sample draws from this distribution will be used later on in excersice 4 along with real data regarding covid-19. Therefore, we need to give a reasonable guess about the prior distribution of β and γ
For β - the tranmission rate under SIR model. In the case of covid-19 disease, this rate is not easy to estimate and the result varied a lot from region to region as well as it's also depending alot on social distancing policies. Therfore, we'll assume that it has a prior uniform distribution $U(10^{-3}, 6)$, meaning

that β can have an equal chance of having any value between 10^{-3} and 6.
For γ - the recovering rate under SIR model. It's reasonable to let it be the inverse of the disease recovering time. For example, with the average recovering time of covid-19 being 2 weeks, γ should have an expected value of 1/14. Or in other words, one over fourteen of the infected group will recover per day. With this in mind, we can let γ follow the normal distribution $N(1/14, 0.1/14)$. Most of the value of γ in this distribution is in the range of 1/28 to 3/28, corresponding to an estimated recovering time from 1.33 weeks to 4 weeks, which fit nicely with real recovering time of covid-19.
In this assignment, we will construct our own Metropolis-Hastings algorithm that will perform the following steps:
1. Initialize β_0 and γ_0.
2. Set $\beta := \beta_0$ and $\gamma := \gamma_0$.
3. Select the next $\beta*$ and $\gamma*$ using current β, γ randomly from $Q(\beta, \gamma)$. Namely, we choose this proposal distribution to be comprised of two probability distribution $N(\beta, \sigma_1)$ and $N(\gamma, \sigma_2)$. Where σ_1 and σ_2 are constansts that can be adjusted to better fit our prior distribution.
4. Calculated the ratio that'll tell us how likely the new $\beta*$ and $\gamma*$ fit the original distribution compared to the current ones

$$r = \frac{\pi(\beta*, \gamma*)Q(\beta, \gamma|\beta*, \gamma*)}{\pi(\beta, \gamma)Q(\beta*, \gamma*|\beta, \gamma)}$$

Note: because we choose our transistion model to follow a Normal distribution with constant variance, $Q(\beta, \gamma|\beta*, \gamma*)$ will be equal to $Q(\beta*, \gamma*|\beta, \gamma)$ because this is a symmertric distribution.
5. Generate a value α randomly from a continuous uniform distribution U(0; 1).
6. If α < r, set $\beta_{i+1} := \beta*$ and $\gamma_{i+1} := \gamma*$, and move to Step 8.
7. Otherwise, set $\beta_{i+1} := \beta_i$ and $\gamma_{i+1} := \gamma_i$, and move to Step 8.
8. Repeat Step 2 with $\beta := \beta_{i+1}$ and $\gamma := \gamma_{i+1}$ until we get m elements.

2.3.2 Implementation

Implementing Metropolis-Hasting algorithm as a helper function to support calculating the prior sampling in file: "Ex3libMHSimplified.py".

```python
import numpy as np

def metropolisHasting(m, logPdf , proposal , logProposalPDF , t0, burnIn = 0):
    #  m: number of sample to draw
    # logPdf: h(t), the pdf that's atleast proportional to our desired distribution
    # proposal(t) = q(t) : return a new t' from current t
    # proposalPDF(tPrime , t) = q(tPrime | t) : the probability density function of q(t)
    # t0: inital parameter
    # burnIn: number of ignored first iteration

    #Return:
    #traceValue: The return trace of t at all iteration, this will be our returned sample
    #accepted: The accepted values, used for plotting
    #rejected: The rejected values, used for plotting
    #samplingPlotAc: The accepted values's iteration, used for plotting
    #samplingPlotRe: The rejected values's iteration, used for plotting

    t = t0;
    traceValue = []
    accepted = []
    rejected = []
    samplingPlotAc = []
    samplingPlotRe = []

    for j in range(burnIn):
            tPrime = proposal(t)

            p = logPdf(t) +logProposalPDF(tPrime, t)
            pPrime = logPdf(tPrime) + logProposalPDF(t, tPrime)

            if pPrime > p:
                t = tPrime
            else:
                a = np.random.uniform(0, 1)
                if a < np.exp(pPrime - p):
                    t = tPrime

    for i in range(m):
        tPrime = proposal(t)

        p = logPdf(t) +logProposalPDF(tPrime, t)
        pPrime = logPdf(tPrime) + logProposalPDF(t, tPrime)

        if pPrime > p:
            t = tPrime
            traceValue.append(t)
            accepted.append(t)
            samplingPlotAc.append(i)
        else:
            a = np.random.uniform(0, 1)
            if a < np.exp(pPrime - p):
                t = tPrime
                traceValue.append(t)
                accepted.append(t)
                samplingPlotAc.append(i)
            else:
                traceValue.append(t)
                rejected.append(tPrime)
                samplingPlotRe.append(i)
    return traceValue, accepted, rejected, samplingPlotAc, samplingPlotRe
```

Calling the MH function and passing the right probability functions:

```
import math
import numpy as np
import matplotlib.pyplot as plt
import Ex3libMHSimplified as MH

m = int(input("Enter m: "))
sigma1 = float(input("Enter sigma1: ")) #1.73
sigma2 = float(input("Enter sigma2: ")) #0.1/7
burnIn = int(input("Enter burnIn: "))

normal_density = lambda sigma, mu, x :1/(sigma * np.sqrt(2 * np.pi)) * np.exp( - (x - mu
    )**2 / (2 * sigma**2))
uniform_density = lambda a, b, x: 1/(b-a) if (a <= x and x <= b) else 0
gamma_density = lambda k, theta, x : x**(k-1)*np.exp(-x/theta) / (math.gamma(theta) *
    theta**k) #note: theta is the scale = 1/rate

def piBeta(beta):
    """
    prior distribution of beta is set to uniform U(10^-10, 1) because of lacking
    knowlegde for covid-19 tranmission rate
    """
    return uniform_density(10**-3, 6, beta)

def piGamma(gamma):
    """
    prior distribution of gamma is assumed to follow N(0.5, 0.1), where the mean 0.5 is
    the inverse of averaged recovery time for
    covid-19 (2 weeks) and the standard deviation 0.1 is estimate for the reported
    recovered period from ~ 1-4 weeks
    """
    return normal_density(0.1/7, 0.5/7, gamma)
    #return uniform_density(10**-2, 1, gamma)

def logPDF(t):
    #pi(beta, gamma) = pi(beta) * pi(gamma)
    return np.log(piBeta(t[0]) * piGamma(t[1]))

def normalProposal(t):
    # tPrime ~ Normal(t, sigma^2)
    return [np.random.normal(t[0],sigma1,1)[0],np.random.normal(t[1],sigma2,1)[0]]

def logNormalProposalPDF(tPrime, t):
    return np.log(normal_density(sigma1,t[0],tPrime[0]) * normal_density(sigma2,t[1],
    tPrime[1]))

dummyPDF = lambda tPrime, t: 1 # we can use this instead of the proposalPDF if it's
    symmertric

trace, ac, re, iac, ire = MH.metropolisHasting(m, logPDF, normalProposal,
    logNormalProposalPDF, [1, 0.1], burnIn)
```

Plotting the trace and sampling process from MH return results:

```

sumg = 0
sumb = 0
betaTrace = []
gammaTrace = []

for b,g in trace:
    sumb += b
    sumg += g
    betaTrace.append(b)
    gammaTrace.append(g)

meanb = sumb/len(trace)
meang = sumg/len(trace)

betaAccepted = []
gammaAccepted= []
for b,g in ac:
    betaAccepted.append(b)
    gammaAccepted.append(g)

betaRejected = []
gammaRejected= []
for b,g in re:
    betaRejected.append(b)
    gammaRejected.append(g)

dataHist = plt.figure(figsize=(12,9))
ax = dataHist.add_subplot(2,1,1)
ax.plot(iac, betaAccepted, '+', color='blue', label="Accepted")
ax.plot(ire, betaRejected, 'x', color='red', label="Rejected")
ax.set_ylabel("Value")
ax.set_title("Figure 1: Sampling Plot of beta")
ax.legend()

ax = dataHist.add_subplot(2,1,2)
ax.plot(iac, gammaAccepted, '+', color='blue', label="Accepted")
ax.plot(ire, gammaRejected, 'x', color='red', label="Rejected")
ax.set_xlabel("Iteration")
ax.set_ylabel("Value")
ax.set_title("Figure 2: Sampling Plot of gamma")
ax.legend()


bTrace = plt.figure(figsize=(10,10))
ax = bTrace.add_subplot(2,2,1)
ax.plot(betaTrace)
ax.set_xlabel("Iteration")
ax.set_ylabel("Value")
ax.set_title("Figure 3: Trace of beta")

ax = bTrace.add_subplot(2,2,2)
ax.plot(gammaTrace)
ax.set_xlabel("Iteration")
ax.set_ylabel("Value")
ax.set_title("Figure 4: Trace of gamma")

ax = bTrace.add_subplot(2,2,3)
ax.hist(betaTrace, bins=50 ,)
ax.set_xlabel("Value")
ax.set_ylabel("Frequency")
ax.set_title("Figure 5: Histogram of beta")

```

```python
ax = bTrace.add_subplot(2,2,4)
ax.hist(gammaTrace, bins=50 ,)
ax.set_xlabel("Value")
ax.set_ylabel("Frequency")
ax.set_title("Figure 6: Histogram of gamma")


print("Acceptance Rate = ", len(ac)/(len(ac)+len(re)))

print("Expected b = ", meanb)
print("Expected g = ", meang)

print("Done", flush = True)


plt.show()

input()
```

Run this exercise with library "matplotlib","numby", required python 3 package, 2 file "Ex3libMHSimplified.py","Ex3PriorSampling.py" in the same directory.

1. Open Command Promt then enter "python Ex3-PriorSampling.py"
2. Enter m: 10000
3. Enter sigma1: 1.73
4. Enter sigma2: 0.014 ($\frac{1}{70}$)
5. Enter burnIn: 3000

Results: the first one show the sampling procress happend all along 10000 iteration, the second show the process of the first 1000 iteration

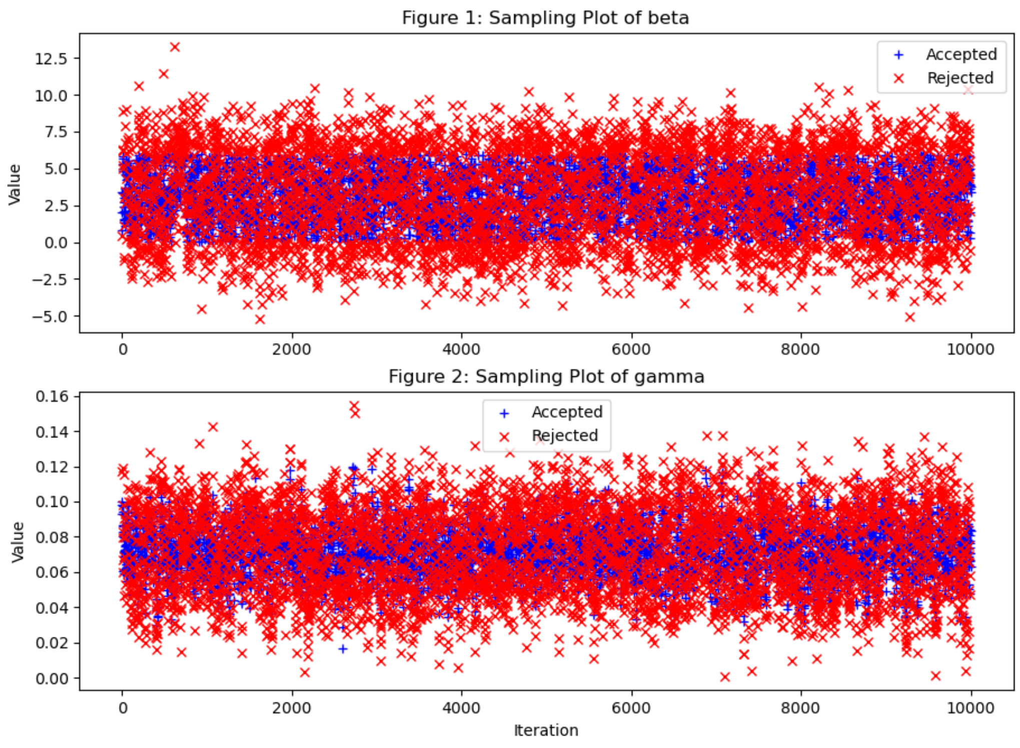


Figure 1: Sampling Plot of beta

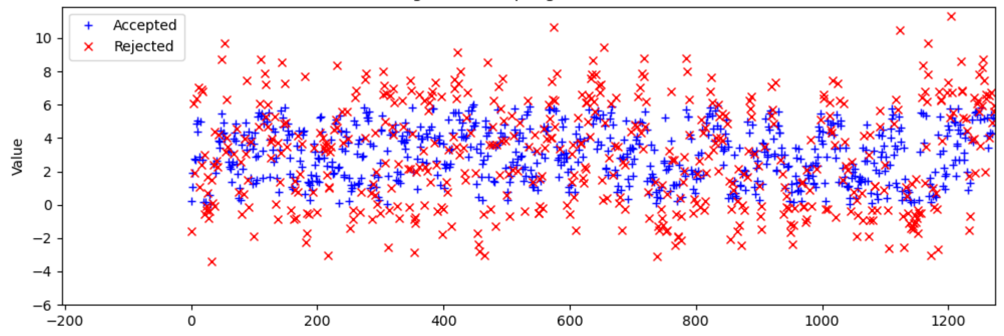


Figure 2: Sampling Plot of gamma

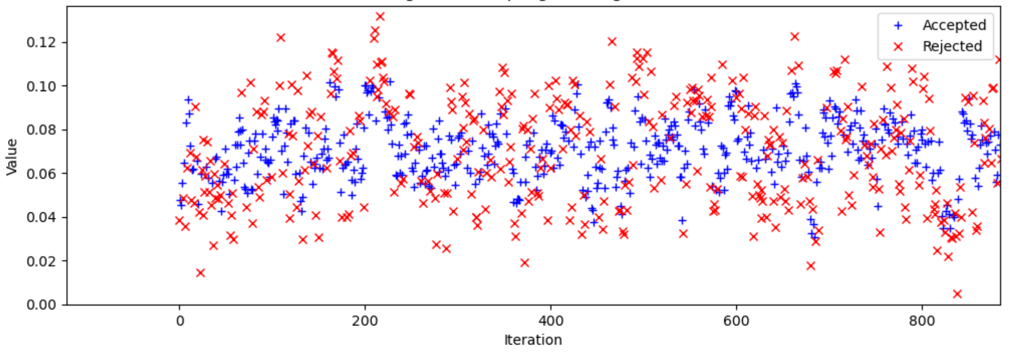


Figure 3: Trace of beta

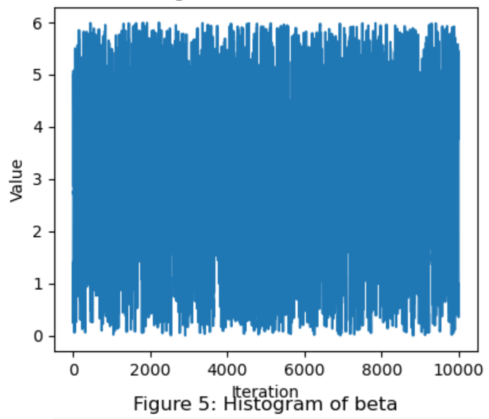


Figure 4: Trace of gamma

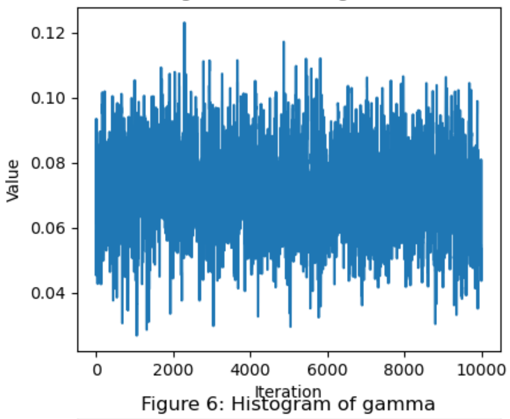


Figure 5: Histogram of beta

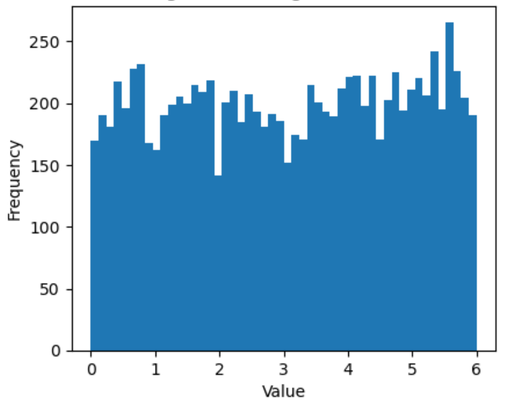
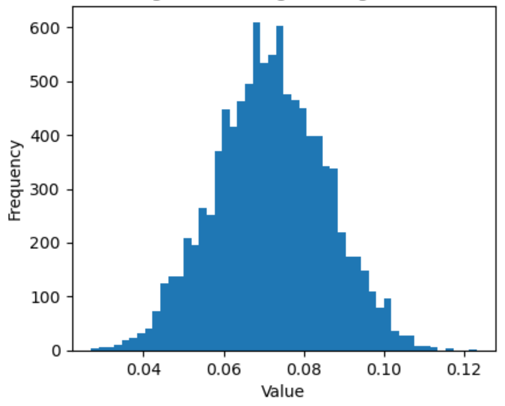


Figure 6: Histogram of gamma



2.4 Craw data for Exercise 4

[36]:
```
EU_nation = \
"""\
Austria, Belgium, Bulgaria, Croatia, Cyprus, \
Czechia, Denmark, Estonia, Finland, France, Germany,\
Greece, Hungary, Ireland, Italy, Latvia, Lithuania, Luxembourg,\
Malta, Netherlands, Poland, Portugal, Romania, \
Slovakia, Slovenia, Spain, Sweden \
""".replace(" ","").split(",")

# United Kingdom did not official exit from the EU
# so we still consider UK is a part of EU

EU_nation.append('United Kingdom')

EU_poplation = EU_poplation[EU_poplation['Name'].isin(EU_nation)].\
                reset_index(drop=True)
```

2.4.1 Get EU Case

JOHN HOPSKIN Corona-Virus Repository

We use data from COVID-19/csse_covid_19_data/csse_covid_19_time_series/ that was cloned from https://github.com/CSSEGISandData/COVID-19 in 08/07/2020

This folder contains daily time series summary tables, including confirmed, deaths and recovered. All data is read in from the daily case report. The time series tables are subject to be updated if inaccuracies are identified in our historical data. The daily reports will not be adjusted in these instances to maintain a record of raw data.

Two time series tables are for the US confirmed cases and deaths, reported at the county level. They are named time_series_covid19_confirmed_US.csv, time_series_covid19_deaths_US.csv, respectively.

Three time series tables are for the global confirmed cases, recovered cases and deaths. Australia, Canada and China are reported at the province/state level. Dependencies of the Netherlands, the UK, France and Denmark are listed under the province/state level. The US and other countries are at the country level. The tables are renamed time_series_covid19_confirmed_global.csv and time_series_covid19_deaths_global.csv, and time_series_covid19_recovered_global.csv, respectively.

General Information

We decide to use 3 data file:

- TIME_SERIES_COVID19_CONFIRMED_GLOBAL_CSV
- TIME_SERIES_COVID19_RECOVERED_GLOBAL_CSV_PATH
- TIME_SERIES_COVID19_DEATHS_GLOBAL_CSV_PATH

Those files share some simlarites:

The Data about Corona-Virus from the JOHN HOPSKIN is contain the data from more than 173 Countries/Regions recorded from 1/22/20 to 7/8/20 (MM/DD/YY) in the time, we cloned to our repos.

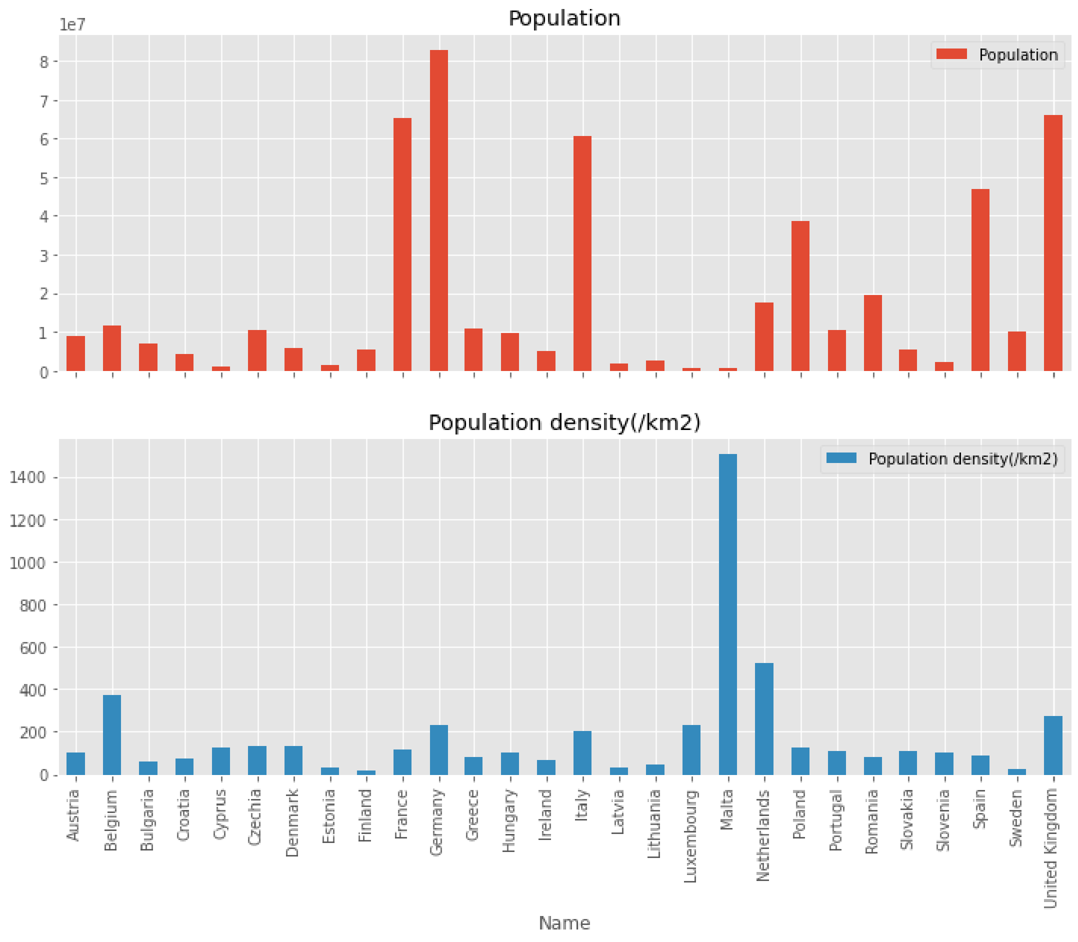
Only have Null value at the Province/State, which is reasonable.

[13]:
```
dat_confirmed = data_process.read_csv(file_path[\
                    'TIME_SERIES_COVID19_CONFIRMED_GLOBAL_CSV_PATH'])
```

[71]:
```
dat_recovered = data_process.read_csv(file_path[\
                    'TIME_SERIES_COVID19_RECOVERED_GLOBAL_CSV_PATH'])
```

[72]:
```
dat_death = data_process.read_csv(file_path[\
                    'TIME_SERIES_COVID19_DEATHS_GLOBAL_CSV_PATH'])
```

2.4.2 EU General Analysis



EU COVID-19 case on 07/08/2020

[57]:
```
EU_07_08_2020 = EU_confirmed[['Country/Region','7/8/20']]

EU_07_08_2020 = EU_07_08_2020.rename(columns={'7/8/20':'Confirmed'})

EU_07_08_2020 = pd.concat([EU_07_08_2020, EU_recover['7/8/20']], axis =1)

EU_07_08_2020 = EU_07_08_2020.rename(columns={'7/8/20':'Recover'})

EU_07_08_2020 = pd.concat([EU_07_08_2020, EU_death['7/8/20']], axis =1)

EU_07_08_2020 = EU_07_08_2020.rename(columns={'7/8/20':'Death'})

EU_07_08_2020.set_index('Country/Region',inplace = True)
```

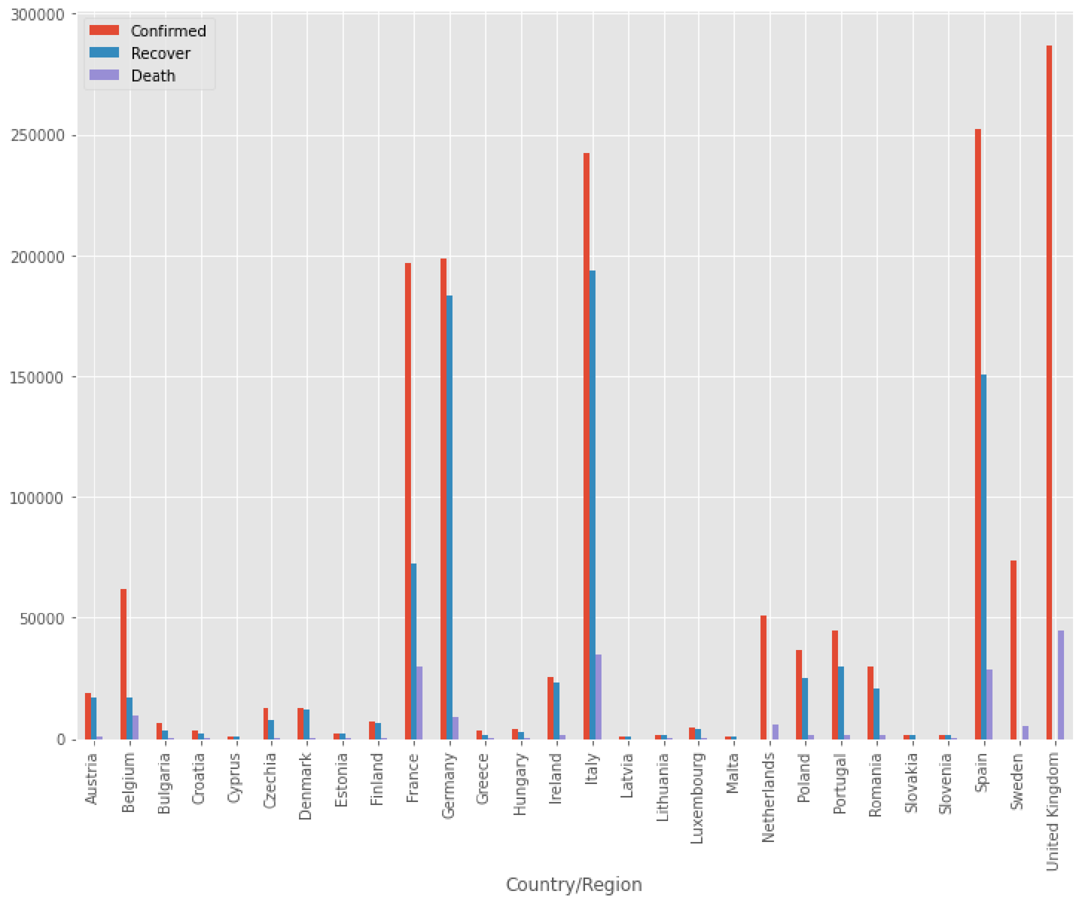
As we review the data, some countries actually do not have the actual number of recovered case, So we decide to remove :

- United Kingdom

- Sweden
- Netherlands

Due to the large different in number of cases between country, we will choose country that had more than 5000 confirmed cases on 07/08/2020

[58]:
```
EU_07_08_2020.plot.bar(figsize =[12,9])
```



[59]:
```
EU_07_08_2020.drop(['United Kingdom','Sweden','Netherlands'],
                   inplace =True)

EU_07_08_2020 = EU_07_08_2020[EU_07_08_2020['Confirmed'] > 5000]

EU_07_08_2020.plot.bar(figsize =[12,9])
```

2.4.3 Processed daily cases

Calculate new case daily

For some unknown reasons, The newly case update fall below 0, which does not make any sense.

[13]:
```
for df_name, df in [['daily_confirmed', confirmed],
                    ['daily_recovered', recovered],
                    ['daily_death', death]]:

    nations = df.columns[1:]
```

```
new_daily_case = df[['Date']][:-1]

for nation in nations:

    new_daily_case[nation] = get_new_daily_case(df, nation)

ix_below_zero = np.where(new_daily_case[nations] < 0)[0]

print (df_name)

display(new_daily_case.iloc[ix_below_zero])
```

```
          Date Austria Belgium Bulgaria Czechia Denmark Finland France
86  2020-04-17      76    1045       32      57     169     192    -17
90  2020-04-21      52     933       49      99     217     115  -2206
92  2020-04-23      69    1496      137      86     137     111   1610
97  2020-04-28      45     525       48      75     157     166  -2512
100 2020-05-01      27     485       39      18      96     125   1212
111 2020-05-12      36     202       46      48      76      51   -226
114 2020-05-15      92     345       37      49      67      58   -112
122 2020-05-23      17     282       19      65      71      11   -105
123 2020-05-24      36     250        6      47      27      20    307
124 2020-05-25      18     113       10      48      41      29   -279
131 2020-06-01      26      98       19      62      35       2   -840
148 2020-06-18      48     128       81     126      47      14    569
153 2020-06-23      41      88      128     127      54      12   -187
154 2020-06-24      28     109      166      93      21       5   -255
156 2020-06-26      58     103      112     260       0       7   -194
157 2020-06-27      74      86       66     305       0       0   -723
163 2020-07-03     115     111      180     121       0       6   -358
164 2020-07-04     115     178       63      75       0       5    -11
167 2020-07-07      92      65      240     129      12       3   -293

    Germany Ireland Italy Poland Portugal Romania   Spain
86     1945     778  3491    363      663     351     887
90     2357     631  3370    313      603     468    4211
92     1870     577  3021    381      444     321  -10034
97     1627     376  2086    422      183     362    2144
100     890     343  1900    270     -161     165    1366
111     927     159   888    283      219     224     661
114     519      92   875    241      227     267     515
122     342      57   531    395      152     213     482
123     272      59   300    305      165     213    -372
124     600      37   397    443      219     146     859
131     285       4   318    230      195     119     294
148     482      13  -148    301      375     320     307
153     391       5   577    294      367     321     334
154     500       9   296    298      311     460     400
156     422      23   175    319      323     325     564
157     235       2   174    193      457     291     301
163     418      11   235    314      413     416       0
164     325      18   192    231      328     391       0
167     356       4   193    277      443     555     383

daily_recovered

          Date Austria Belgium Bulgaria Czechia Denmark Finland France
32  2020-02-23       0       0        0       0       0       0      0
```

```
54  2020-03-16      -5       0        0       0       0       0      0
56  2020-03-18       0       0        0       0       0       0      0
57  2020-03-19       0     -30        0       1       0       0      0
114 2020-05-15      53     159       28      41     148       0     -1
118 2020-05-19     204     160       38     104     120    -200    467
130 2020-05-31       3      32       16      84      50       0     -3
144 2020-06-14       7      21       54      70      22       0   -101
156 2020-06-26      23      23       18      14       0       0    -95
163 2020-07-03      49      18        6       4       0       0    -89
167 2020-07-07      35      16      129     100      18     100   -230

    Germany Ireland Italy Poland Portugal Romania Spain
32        0       0    -1      0        0       0     0
54        0       5   192      0        0       7   498
56        8       0   415    -12        0       6    26
57       67       0     0      0        2       0   481
114    1003       0  2605    257      494     204  1663
118    1285    1590  2881    280       21     190     0
130     280       0   848    178      143     170     0
144     603       0   640    157      183      98     0
156     369       0   969    754      231     349     0
163     700       0   477    476      348     309     0
167     492       0   825    640      269     265     0


daily_death

          Date Austria Belgium Bulgaria Czechia Denmark Finland France
74  2020-04-05      16     185        2      11       8      -1    833
79  2020-04-10      18     327        3      10      13       1    635
110 2020-05-11       3      54        2       1      -6       4    347
114 2020-05-15       1      46        3       1       6       4     -2
116 2020-05-17       0      28        2      -1       1       2    131
117 2020-05-18       3      28        2       5       3       1   -217
123 2020-05-24       1      32        0       2       1       1     90
123 2020-05-24       1      32        0       2       1       1     90
130 2020-05-31       0      19        0       1       2      -2     28
130 2020-05-31       0      19        0       1       2      -2     28
140 2020-06-10       1       7        1      -2       0       1     27
142 2020-06-12       2       4        0      -1       3       0     24
153 2020-06-23       0       9        1       4       0       0      9
156 2020-06-26       2       1        1       0       0       0     -1
157 2020-06-27       2       0        3      -1       0       0      0
163 2020-07-03       0       6        2      -2       0       0      0
164 2020-07-04       1       0        5      -3       0       0      1
165 2020-07-05       0       3        4       2       1       0     23
167 2020-07-07       0       2        5       0       0       0     -1
167 2020-07-07       0       2        5       0       0       0     -1

    Germany Ireland Italy Poland Portugal Romania  Spain
74      226      16   636     13       16      25    700
79      -31      33   619     27       35      21    525
110      77      21   172     28       19      20    176
114      41      15   153      8       13      24    104
116      41       4    99     11       13      13    146
117      78      14   162     12       16      17     69
123      26      -2    92     11       14      20  -1918
123      26      -2    92     11       14      20  -1918
130      15      -2    60     10       14      10      0
```

```
130        15       -2    60     10      14      10      0
140        20        8    53      9       7       9      0
142        10        0    78     15       7      14      0
153        14        6   -31     21       3      16      2
156         3        4     8      6       6      10      3
157         0        1    22      3       3      23      2
163        10        1    21      5       7      23      0
164         3        0     7      5       9      19      0
165        -1        0     8      4       6      18      3
167        14       -4    15     14       2      18      4
167        14       -4    15     14       2      18      4
```

Fix and Save daily_case + Cumulative case

[22]:
```
for df_name, df in [['daily_confirmed', confirmed],
                    ['daily_recovered', recovered],
                    ['daily_death', death]]:

    nations = df.columns[1:]

    new_daily_case = df[['Date']][:-1]

    for nation in nations: # column based series

        # Calculate new daily_case
        new_daily_case[nation] = get_new_daily_case(df, nation)

        # Engineer below_zero and sudden_drop_zero Error by
        # smoothing average of 3
        # drop limit is : 50

        remove_below_zero(new_daily_case[nation],
                          number_of_average = 3)

        remove_sudden_drop_zero(new_daily_case[nation],
                                drop_limit = 50,
                                number_of_average = 3)

    # Save Daily_Case:
    file.save_pickle(dir_path['PROCESSED_DIR'] +'/'+
                     df_name + '.pkl', new_daily_case)

    new_daily_case.to_csv(dir_path['PROCESSED_DIR'] +'/'+
                     df_name + '.csv', index = False)

    # Calculate Cumulative Case

    cumumlative_case = new_daily_case.iloc[:,1:].cumsum()

    cumumlative_case.reset_index(inplace = True)

    cumumlative_case['index'] = new_daily_case['Date']

    # Save Cumulative Case
    file.save_pickle(dir_path['PROCESSED_DIR'] +'/cumumlative' +
                     df_name + '.pkl', cumumlative_case)
    cumumlative_case.to_csv(dir_path['PROCESSED_DIR'] +'/cumumlative'+
                     df_name + '.csv', index = False)
```

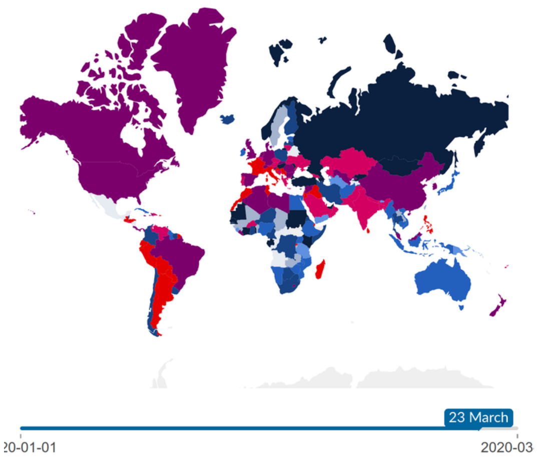
2.4.4 Policy stringency

The government policy data is cloned from git@github.com:owid/covid-19-data.git

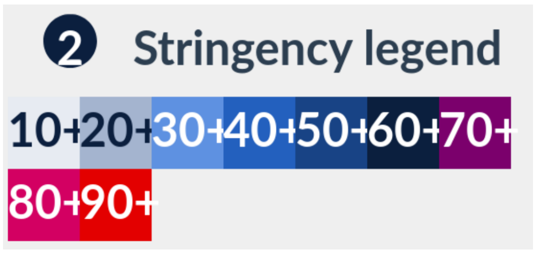
The policy_stringency data is a measured of how Goverment Policy Stringency in continuous number in range from 0 to 100.

This data is come from Oxford University.

With: + 0 is lowest level, no policy or prepare for the COVID-19 Out Break + 100 is is highest levean absolute alert, Complete Shutdown. . .



Click on bar or drag slider to selected day



```
[79]: # fix different name of Czech Republic
      dat['location'].replace({'Czech Republic': 'Czechia'}, inplace = True)
      dat = dat.pivot(index='date',columns='location',
                      values = 'stringency_index')
      dat.columns.name = None
      dat.reset_index(inplace= True)
      dat = dat[nations][1:190] # Get rows from 01/01/2020 - 08/07/2020
      # Fill NaN by mean
      mean_values = dat.mean(axis=1)
      for i, _ in enumerate(dat):
          dat.iloc[:, i] = dat.iloc[:, i].fillna(mean_values)
      # save data
      file.save_pickle(dir_path['PROCESSED_DIR'] + '/policy_stringency.pkl',
                       dat)
      dat.to_csv(dir_path['PROCESSED_DIR'] + '/policy_stringency.csv',
                 index = False)
```

2.5 Exercise 4

2.5.1 Background

At the moment, gorverment arount the world are implement many policies to stop the spreading of COVID-19.Therefore, we are really in the head of an effective way to measure the efficiency of our policy.
However, due to the limited information about the current disease and the reliability of the data. It's hard to calculate and interpret the information.
In this part, we are using the SIR model on the actual data, knowning the limitation of the our information and the over simplication of our assumpstions.
In our case, we consider R is the number of people that could not get infected and could not spread the disease (recovered + death)
We assume:
The daily new recorded case $X_1 = \frac{dR}{dt} \sim$ Poission(γI)

$$X_1 = \frac{dR}{dt} \sim Poission(\gamma I) = \gamma I_{t-1} \tag{13}$$

The daily new case X_2

$$X_2 = \frac{dR}{dt} + \frac{dI}{dR} = \frac{\beta}{N} I_{t-1} S_{t-1} = \beta I \sim Poission(\beta I) \tag{14}$$

In our data,

$$\begin{cases} N \gg I \Rightarrow S \approx N - I - R \\ N \gg R \end{cases}$$

We standardlized the two expressions (13), (14) to Normal Distribution:

$$Y_1 = \frac{X_1 - \gamma I}{\sqrt{\gamma I}} \sim N(0,1) \tag{15}$$

$$Y_2 = \frac{X_2 - \beta I}{\sqrt{\beta I}} \sim N(0,1) \tag{16}$$

We want to know the development of he outbreak.If more and more people are affected:

$$\frac{dI}{dt} > 0 \Leftrightarrow \frac{\beta}{N} IS - \gamma I > 0 \Leftrightarrow \beta I - \gamma I > 0 \Leftrightarrow \frac{\beta}{\gamma} > 1$$

We are interested in $R_0 = \frac{\beta}{\gamma}$

$$\begin{aligned} E(R_0) &= \int \pi(\beta, \gamma | X) R_0(\beta, \gamma) d(\beta, \gamma) \\ &\propto \int \pi(X|\beta, \gamma) \pi(\beta, \gamma) R_0(\beta, \gamma) d(\beta, \gamma) \\ &\approx \sum_{i=1}^{m} \pi(X|\beta_i, \gamma_i) \frac{\beta_i}{\gamma_i} \end{aligned}$$

where (β_i, γ_i) is selected from the probability $\pi(\beta, \gamma)$ and m is the size of sample.
We consider β_i and γ_i is independent

$$p(X|\beta_i, \gamma_i) = p(Y_1|\beta_i) p(Y_2|\gamma_i) \tag{17}$$

As we've said before, $p(Y_1|\beta_i)$ and $p(Y_2|\gamma_i)$ are both follow the standard normal distribution N(0, 1), therefore:

$$log(p(Y_1|\beta_i)) = \sum_{k=1}^{n} log(\frac{1}{\sqrt{2\pi}} e^{\frac{-Y_{1k}^2}{2}})$$

$$log(p(Y_2|\gamma_i)) = \sum_{k=1}^{n} log(\frac{1}{\sqrt{2\pi}} e^{\frac{-Y_{2k}^2}{2}})$$

where n is the lenght of our real data.
It's here that we run into a computational problem:
the result of this $p(Y_1|\beta_i)$ and $p(Y_2|\gamma_i)$ is extremely small, which make sense since technically, the probability of the random variable exactly equal any data point $P(Z = Y_k) = \int_{Y_k}^{Y_k} \frac{1}{\sqrt{2\pi}} e^{\frac{-Y_{1k}^2}{2}}$ for a continous distribution is, theoratically, zero.

Even if we ignore the integral part and just calculate as $P(Z = Y_k) = \frac{1}{\sqrt{2\pi}} e^{\frac{-Y_{1k}^2}{2}}$, the result will be in the range of (0, 0.24) with maximum probability when $Y_k = 0$ (the mean of N(0, 1)). And that is just the probability of a SINGLE data point. We still have to multiply the probability of hundred of data points together. A single out of place data point with probability like 0.00000001 (which actually happen pretty frequently consider that real data is not perfectly fit the SIR model) will lower the whole result to nearly zero. It means that the more data we use, the lower our probability will become.
Using the sum of log equation above, we can actually calculate how small that is: $log(p(Y_1|\beta_i))$ and $log(p(Y_2|\gamma_i))$ are both return result ranging -1000000000 to -1000, for any country data and with any combination of β and γ.
Our goal is to calculate the sum:

$$\sum_{i=1}^{m} p(X|\beta_i, \gamma_i) \frac{\beta_i}{\gamma_i}$$

But since as explain above: $p(X|\beta_i, \gamma_i)$ = at least e^{-1000}, any calculator we known of refuse to compute this to anything other than zero. making EVERY TERM of the sum become zero.

Due to the limitation and unreliability natural of our data, as well as to get at least a usable probability. We'll instead use a method of counting the ratio of number of data points that is inside an predetermined range over the data size:

1. Calculate the likelihood of $p(Y_1|\beta_i)$ and $p(Y_2|\gamma_i)$ with $Y_1 = y_{11}, y_{12}, ..., y_{1k}$ and $Y_2 = y_{21}, y_{22}, ..., y_{2k}$
2. for j = 1 to k do:

   If $-1 < Y_{1j} < 1 \Rightarrow$ We assume $p(Y_{1j}|\beta_i) = 1$ else $p(Y_{1j}|\beta_i) = 0$

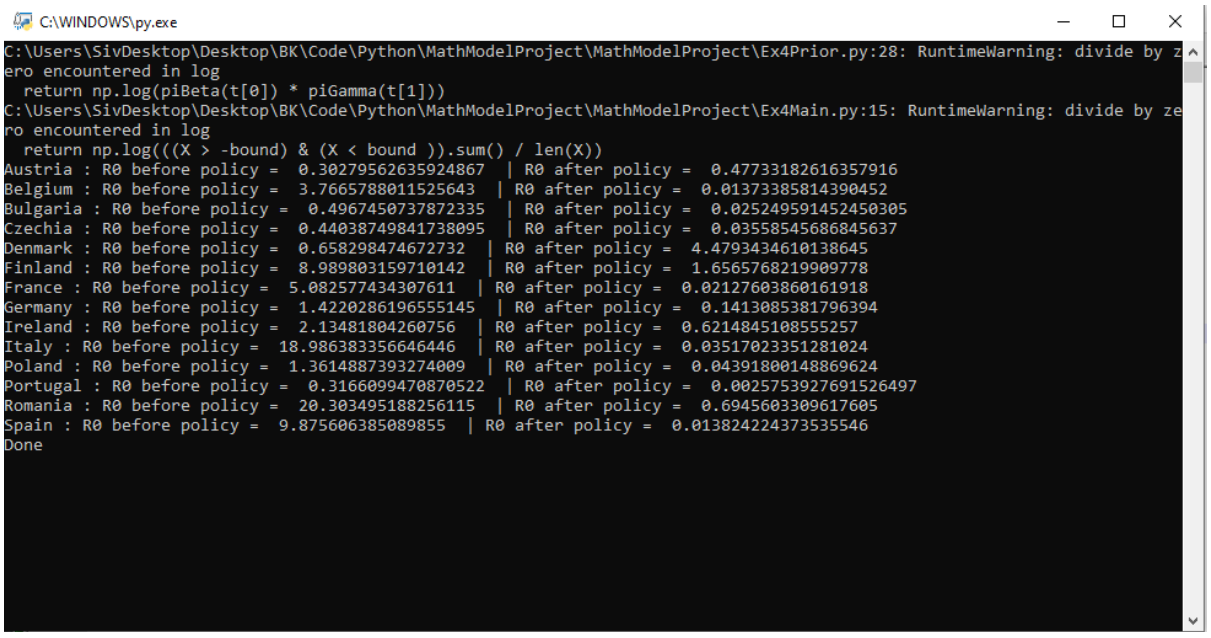
   If $-1 < Y_{2j} < 1 \Rightarrow$ We assume $p(Y_{2j}|\gamma_i) = 1$ else $p(Y_{2j}|\gamma_i) = 0$
3. $p(Y_1|\beta_i) = \frac{\sum p(Y_{1i}|\beta_i)}{k}$
4. $p(Y_2|\gamma_i) = \frac{\sum p(Y_{2i}|\gamma_i)}{k}$

Note: the acceptance range from -1 to 1 is chosen arbitrary, for N(0, 1) this range will consist of about 69% of the whole distribution density.

2.5.2 Analyzing The Basic Reproduction Number

By using the Metropolis–Hastings sampler in Exercise 3 and the COVID-19 data crawl from above, we can say that the policy and social distancing mostly effect the R_0 of each country, or say in another way that most of social distacing and policy help to control to the outbreak of COVID-19 for like: Belgium, France, Ireland, Italy, Poland, Romania, Spain, based on the result, the R_0 are noticeably lower than it should be.The R_0 of Denmark and Austria are higher than other country, that might be caused by the relatively lower rate of recovery of the country.
Note: The R_0 values show below are not nesscessary equal to real worl R_0, those number should only proportional to R_0 by a constant.

Country	R_0 before policy and social distacing	R_0 after policy and social distacing
Austria	0.30279562635924867	0.47733182616357916
Belgium	3.7665788011525643	0.01373385814390452
Bulgaria	0.4967450737872335	0.025249591452450305
Czechia	0.44038749841738095	0.03558545686845637
Denmark	0.658298474672732	4.4793434610138645
Finland	8.989803159710142	1.6565768219909778
France	5.082577434307611	0.02127603860161918
Germany	1.4220286196555145	0.1413085381796394
Ireland	2.13481804260756	0.6214845108555257
Italy	18.986383356646446	0.03517023351281024
Poland	1.3614887393274009	0.04391800148869624
Portugal	0.3166099470870522	0.0025753927691526497
Romania	20.303495188256115	0.6945603309617605
Spain	9.875606385089855	0.013824224373535546



```
C:\Users\SivDesktop\Desktop\BK\Code\Python\MathModelProject\MathModelProject\Ex4Prior.py:28: RuntimeWarning: divide by z
ero encountered in log
  return np.log(piBeta(t[0]) * piGamma(t[1]))
C:\Users\SivDesktop\Desktop\BK\Code\Python\MathModelProject\MathModelProject\Ex4Main.py:15: RuntimeWarning: divide by ze
ro encountered in log
  return np.log(((X > -bound) & (X < bound )).sum() / len(X))
Austria : R0 before policy =  0.30279562635924867  | R0 after policy =  0.47733182616357916
Belgium : R0 before policy =  3.7665788011525643  | R0 after policy =  0.01373385814390452
Bulgaria : R0 before policy =  0.4967450737872335  | R0 after policy =  0.025249591452450305
Czechia : R0 before policy =  0.44038749841738095  | R0 after policy =  0.03558545686845637
Denmark : R0 before policy =  0.658298474672732  | R0 after policy =  4.4793434610138645
Finland : R0 before policy =  8.989803159710142  | R0 after policy =  1.6565768219909778
France : R0 before policy =  5.082577434307611  | R0 after policy =  0.02127603860161918
Germany : R0 before policy =  1.4220286196555145  | R0 after policy =  0.1413085381796394
Ireland : R0 before policy =  2.13481804260756  | R0 after policy =  0.6214845108555257
Italy : R0 before policy =  18.986383356646446  | R0 after policy =  0.03517023351281024
Poland : R0 before policy =  1.3614887393274009  | R0 after policy =  0.04391800148869624
Portugal : R0 before policy =  0.3166099470870522  | R0 after policy =  0.0025753927691526497
Romania : R0 before policy =  20.303495188256115  | R0 after policy =  0.6945603309617605
Spain : R0 before policy =  9.875606385089855  | R0 after policy =  0.013824224373535546
Done
```

2.5.3 Implementation

Implementing exercise 4
Calling MH sample for beta and gamma in Ex4Prior.py, which is the same as Ex3_PriorSampling call but without the plotting part:

```python
import Ex4Prior as pr
import Ex3libMHSimplified as MH
import csv
import numpy as np
import math
import scipy.special as sp


def loglikelihood_standard_normal(X): #for a continuous distribution, this gave a very
    very low number
    n = len(X)
    return -1/2*np.sum(np.power(X,2))-n*np.log(np.sqrt(2*np.pi))

bound = 1 #confidence range, should be positive
def loglikelihood_standard_normal_accept_ratio(X):
    return np.log(((X > -bound) & (X < bound )).sum() / len(X))

def loglikelihoodXinGammaDist(beta, gamma, X):#pdf of gammaDist, we don't use this, it's
     very imprecise
    n = len(X)
    return n*beta*np.log(gamma) - n*np.log(sp.gamma(beta)) + (beta-1)*np.sum(np.log(X))
    - gamma*np.sum(X)


#calling ex3 sampling function for prior of beta and gamma, the sample is stored in the
    list 'trace'
m = 10000#int(input("Enter m: "))
burnIn = 3000#int(input("Enter burnIn: "))
trace, ac, re, iac, ire = MH.metropolisHasting(m, pr.logPDF, pr.normalProposal, pr.
    dummyPDF, [1, 0.1], burnIn)
```

Loading real data crawed from github into the program:

```
#extracting and reprocess csv
with open('cumumlativedaily_confirmed.csv','rt')as f:
  fileContent = csv.reader(f)
  data1 = []
  for row in fileContent:
      data1.append(row)
with open('cumumlativedaily_death.csv','rt')as f:
  fileContent = csv.reader(f)
  data2 = []
  for row in fileContent:
      data2.append(row)
with open('cumumlativedaily_recovered.csv','rt')as f:
  fileContent = csv.reader(f)
  data3 = []
  for row in fileContent:
      data3.append(row)

numberOfDay = len(data1) - 1
numberOfWeek = int(numberOfDay / 7)
numberOfCountry = len(data1[0]) - 1 #will crash if data is empty

countryList = []
for i in range (0, numberOfCountry):
    countryList.append(data1[0][i+1])

offset = 0

IRList = [[0 for x in range(numberOfDay-offset)] for y in range(numberOfCountry)]
for i in range (0, numberOfCountry):
    for j in range (0, numberOfDay-offset):
        IRList[i][j] = int(data1[j+1+offset][i+1])
IRList = np.array(IRList)

RList = [[0 for x in range(numberOfDay-offset)] for y in range(numberOfCountry)]
for i in range (0, numberOfCountry):
    for j in range (0, numberOfDay-offset):
        RList[i][j] = int(data2[j+1+offset][i+1]) + int(data3[j+1+offset][i+1])
RList = np.array(RList)

offset += 1
IList = [[0 for x in range(numberOfDay-offset)] for y in range(numberOfCountry)]
for i in range (0, numberOfCountry):
    for j in range (0, numberOfDay-offset):
        IList[i][j] = IRList[i][j] - RList[i][j]
IList = np.array(IList)

dIdRList = [[0 for x in range(numberOfDay-offset)] for y in range(numberOfCountry)]
for i in range (0, numberOfCountry):
    for j in range (0, numberOfDay-offset):
        dIdRList[i][j] = IRList[i][j+1] - IRList[i][j]
        if dIdRList[i][j] <= 0 : dIdRList[i][j] = 0.001
dIdRList = np.array(dIdRList)

dRList = [[0 for x in range(numberOfDay-offset)] for y in range(numberOfCountry)]
for i in range (0, numberOfCountry):
    for j in range (0, numberOfDay-offset):
        dRList[i][j] = RList[i][j+1] - RList[i][j]
        if dRList[i][j] <= 0 : dRList[i][j] = 0.001
dRList = np.array(dRList)

```

```
#csv part done
```

Setting up the period before and after social distancing, then do the sum

$$\sum_{i=1}^{m} \pi(X|\beta_i, \gamma_i)\frac{\beta_i}{\gamma_i}$$

```
#ContryOffset = [42,42,51,45,45,46,35,35,47,30,48,45,48,36]
ContryOffset = [42,45,51,45,45,46,40,40,49,37,50,45,56,45]
#gammaDelay =   [0,0, 0,0,0,0,0,0, 0,0,0,0,0,0]
#betaDelay =    [0,0, 0,0,0,0,0,0, 0,0,0,0,0,0]
PolicyStart =  [64,69,64,62,64,63,68,69,66,45,66,70,63,65] #lag time included
PolicyEnd = [102,137,120,88,167,127,131,117,134,102,128,133,130,125]
R0List = []
R0ListPolicy = []
for i in range (0, numberOfCountry, 1):
    dRL = []
    dIdRL = []
    IListGamma = []
    IListBeta = []
    #maxDelay = gammaDelay[i] if gammaDelay[i] > betaDelay[i] else betaDelay[i]

    #start = PolicyStart[i] if(PolicyStart[i] != -1) else len(dRList[i])-maxDelay
    start = PolicyStart[i]
    for j in range(ContryOffset[i], start+1):
        #dRL.append(dRList[i][j+gammaDelay[i]])
        #dIdRL.append(dIdRList[i][j+betaDelay[i]])
        dRL.append(dRList[i][j])
        dIdRL.append(dIdRList[i][j])
        IListGamma.append(IList[i][j])
        IListBeta.append(IList[i][j])

    dRL = np.array(dRL)
    dIdRL = np.array(dIdRL)
    IListGamma = np.array(IListGamma)
    IListBeta = np.array(IListBeta)

    dRLPolicy = []
    dIdRLPolicy = []
    IListPolicy = []
    #for k in range(start, len(dRList[i])-maxDelay):
        #dRLPolicy.append(dRList[i][k+gammaDelay[i]])
        #dIdRLPolicy.append(dIdRList[i][k+betaDelay[i]])
    for k in range(start, PolicyEnd[i]):
        dRLPolicy.append(dRList[i][k])
        dIdRLPolicy.append(dIdRList[i][k])
        IListPolicy.append(IList[i][k])

    dRLPolicy = np.array(dRLPolicy)
    dIdRLPolicy = np.array(dIdRLPolicy)
    IListPolicy = np.array(IListPolicy)

    sum = 0
    sumPolicy = 0
    for beta, gamma in trace:
        lamdaGamma = gamma*IListGamma
        lamdaBeta = beta*IListBeta
        likelihoodGamma = loglikelihood_standard_normal_accept_ratio((dRL-lamdaGamma) /
    np.sqrt(lamdaGamma))
        likelihoodBeta = loglikelihood_standard_normal_accept_ratio((dIdRL-lamdaBeta) /
    np.sqrt(lamdaBeta))
        likelihood = likelihoodGamma + likelihoodBeta
        sum += np.exp(likelihood) * beta/gamma

        lamdaGamma = gamma*IListPolicy
        lamdaBeta = beta*IListPolicy
        likPolicyGamma = loglikelihood_standard_normal_accept_ratio((dRLPolicy-
    lamdaGamma) / np.sqrt(lamdaGamma))
        likPolicyBeta = loglikelihood_standard_normal_accept_ratio((dIdRLPolicy-
    lamdaBeta) / np.sqrt(lamdaBeta))
        sumPolicy += np.exp(likPolicyBeta+ likPolicyGamma) * beta/gamma

    R0List.append(sum)
    R0ListPolicy.append(sumPolicy)

for i in range (0, numberOfCountry, 1):
    print(countryList[i], ": R0 before policy = ", R0List[i], " | R0 after policy = ",
    R0ListPolicy[i])

print("Done", flush = True)

input()
```

2.6 Exercise 5

2.6.1 Background

Mean Squared Errors

In machine learning, our main goal is to minimize the error which is defined by the Loss Function. And every type of Algorithm has different ways of measuring the error.

In statistics, the mean squared error (MSE) or mean squared deviation (MSD) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value.

The MSE assesses the quality of a predictor (i.e., a function mapping arbitrary inputs to a sample of values of some random variable), or an estimator (i.e., a mathematical function mapping a sample of data to an estimate of a parameter of the population from which the data is sampled). The definition of an MSE differs according to whether one is describing a predictor or an estimator.

If a vector of n predictions is generated from a sample of n data points on all variables, and Y is the vector of observed values of the variable being predicted, with $\hat{Y}$ being the predicted values (e.g. as from a least-squares fit), then the within-sample MSE of the predictor is computed as.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Minimize Loss(training process) The most popular minimizer use some variation of gradient descent which required the gradient of the loss function to be known. We want to obtain the optimal (β, γ) by performing first-order iterative optimization algorithm.

However, this is very hard to achieve because the derivatives: $\frac{\delta I}{\delta \beta}$ and $\frac{\delta R}{\delta \gamma}$ is almost impossible to calculate.

Therefore, we will not choose new β and γ based on any directions and instead random a new β and γ based on the current β and γ (the ones produce the smallest loss). To prevent falling indefinitly into local minima of the loss function, we will expanded the randomness(ie. standard deviation) for choosing new beta and gamma after a period amount of time has passed without finding smaller loss.

2.6.2 Implementation

Our training loss function takes in:

- **lossFunction**: The function to calculate the loss value of the given data and the predicted data (produce using RK4's Algorithm with the initial beta and gamma).
- **proposal**: The function to produce new beta and gamma from the current beta and gamma followed normal distribution $N0$.
- **data**: represent the real data given by the .csv file.

- **t0**: an array of initial beta and gamma respectively.
- **randomness**: the initial randomness to be passed to the proposal which is used to find the closest *localminima* of the loss.
- **randomnessScaleFactor**: scaling to increase used to bump up randomness at searching for a long time.
- **searchTimeBeforeScaling**: number of loop without found the new local minima off loss. This number will grow as the search area grows.
- **m**: number of accepted new { β, γ } that give the new local minimum loss.

Output: The ouput of our training function is the current loss and the most suitable { β, γ } founded.

```python
def traniningLoss(lossFuntion, proposal, data, t0, randomness, randomnessScaleFactor,
    searchTimeBeforeScaling, m):
    # lossFuntion: L(t, data): loss value of predicted data base on t and real data, it'
    s up to you to decide the prediction outside
    # proposal(t, randomness) = q(t, randomess) : return a new t' from current t, should
     be sysmetric
    # data: real world data, used to find loss
    # t0: inital parameter
    # randomness: the inital randomness to be passed to the proposal, the length of this
     list should be the same as t0
    # randomnessScaleFactor: scaling to increase randomness used to bump up randomness
    at searching for a long time
    # searchTimeBeforeScaling: Number of loop without new change in loss value before
    scaling up the randoomness. This number will grow as the search area grows, by scale
    to the power of dimension
    # m: number of accepted new t before stop

    #Return:
    #t: final [beta, gamma]

    #converting all list to array to prevent further complication of matrix operation
    later on, and also to boost performance
    randomness = np.array(randomness)
    t0 = np.array(t0)
    data = np.array(data)
    #end convert

    #E.g: if one dimension of t scale by a factor of 2, and the dimension is 2, the area
     will be quadruple, therefore, we'll spend 4 times more time looking inside it
    dimension = len(t0)
    timeScaleFactor = int(np.power(randomnessScaleFactor, dimension))

    t = t0.copy()
    rand = randomness.copy()
    searchTime = searchTimeBeforeScaling
    loss = lossFuntion(t, data)

    ac = 0
    scaleCounter = 0
    while ac < m:
        tNew = proposal(t, rand)
        newLoss = lossFuntion(tNew, data)

        if(newLoss < loss):
            t = tNew
            loss = newLoss

            print(ac, int(loss), t, rand, searchTime, flush=True)
            rand = randomness.copy() # reset randomness
            searchTime = searchTimeBeforeScaling # reset searchTime
            scaleCounter = 0 # reset counter
            ac += 1
        else:
            scaleCounter += 1
            if(scaleCounter > searchTime):
                rand = randomnessScaleFactor*rand
                searchTime = timeScaleFactor*searchTime

                print("BumpUp randomness: ", rand, searchTime, flush=True)
                scaleCounter = 0 # reset counter


        """
        r = 1 if loss > newLoss else 0.01
        a = np.random.uniform(0, 1)
        if a <= r:
            print(insideTol, int(loss), t)
            t = tNew
            loss = newLoss
        """

    return t, loss
```

SIRlossFunction: used to calculate the approximate loss of the predicted model and the real data.

```
def SIRlossFunction(t, data):
    predictI, predictR = rk4.RK4SIR(1000000000, I0, R0, t[0], t[1], time, 1)
    #dataI = np.array(data[0])
    #dataS = np.array(data[1])
    return (meanSquareError(data[0], predictI) + meanSquareError(data[1], predictR))/2
```

meanSquareError: calculate the approximate squared loss of the predicted data and the real data by the following formula:
$\text{MSE} = \frac{\sum(data-prediction)^2}{numberofdata}$

```
def meanSquareError(data, prediction):
    return np.sum(np.power(data - prediction, 2)) / len(data)
```

normalProposal: use to find new β and γ given in the random range

```
def normalProposal(t, randomness): #only propose non-negative for beta and gamma
    # tPrime ~ Normal(t, sigma)
    r = []
    i = 0
    for para in t:
        newT = np.random.normal(para, randomness[i], 1)[0]
        while newT <= 0: #ensure non-negative
            newT = np.random.normal(para, randomness[i], 1)[0]
        r.append(newT)
        i += 1
    return np.array(r)
```

Retrieve data from .csv files

```
#extracting and reprocess csv
with open('cumulativedaily_confirmed.csv','rt')as f:
  fileContent = csv.reader(f)
  data1 = []
  for row in fileContent:
      data1.append(row)
with open('cumulativedaily_death.csv','rt')as f:
  fileContent = csv.reader(f)
  data2 = []
  for row in fileContent:
      data2.append(row)
with open('cumulativedaily_recovered.csv','rt')as f:
  fileContent = csv.reader(f)
  data3 = []
  for row in fileContent:
      data3.append(row)

numberOfDay = len(data1) - 1
numberOfWeek = int(numberOfDay / 7)
numberOfCountry = len(data1[0]) - 1 #will crash if data is empty

countryList = []
for i in range (0, numberOfCountry):
    countryList.append(data1[0][i+1])

offset = 0
```

```

IRList = [[0 for x in range(numberOfDay-offset)] for y in range(numberOfCountry)]
for i in range (0, numberOfCountry):
    for j in range (0, numberOfDay-offset):
        IRList[i][j] = int(data1[j+1+offset][i+1])
    #IRList[i] = np.array(IRList[i])
IRList = np.array(IRList)

RList = [[0 for x in range(numberOfDay-offset)] for y in range(numberOfCountry)]
for i in range (0, numberOfCountry):
    for j in range (0, numberOfDay-offset):
        RList[i][j] = int(data2[j+1+offset][i+1]) + int(data3[j+1+offset][i+1])
    #RList[i] = np.array(RList[i])
RList = np.array(RList)

IList = [[0 for x in range(numberOfDay-offset)] for y in range(numberOfCountry)]
for i in range (0, numberOfCountry):
    for j in range (0, numberOfDay-offset):
        IList[i][j] = IRList[i][j] - RList[i][j]
    #IList[i] = np.array(IList[i])
IList = np.array(IList)
```

Call user's inputs and log out the data in the console.

```
print("Country List:")
count = 0
for s in countryList:
    print(count,". ",s)
    count += 1

countryID = int(input("Choose a country: "))
start = int(input("Choose start day index (0-168): "))
end = int(input("Choose end day index (startDay -> 168): "))

data = []
data.append(np.array(IList[countryID])[start:end])
data.append(np.array(RList[countryID])[start:end])

I0 = data[0][0]
R0 = data[1][0]
time = end - start - 1

#betaGamma, loss = traniningLoss(1.1, SIRlossFunction, normalProposal, data, [0, 0],
    200)
generation = int(input("Number of generation: "))
betaGamma, loss = traniningLoss(SIRlossFunction, normalProposal, data, [0.1, 0.1],
    [0.0001, 0.0001], 2, 5, generation)

print("beta = ", betaGamma[0])
print("gamma = ", betaGamma[1])
print("Loss = ", loss)
```

Plot the graph for visualization

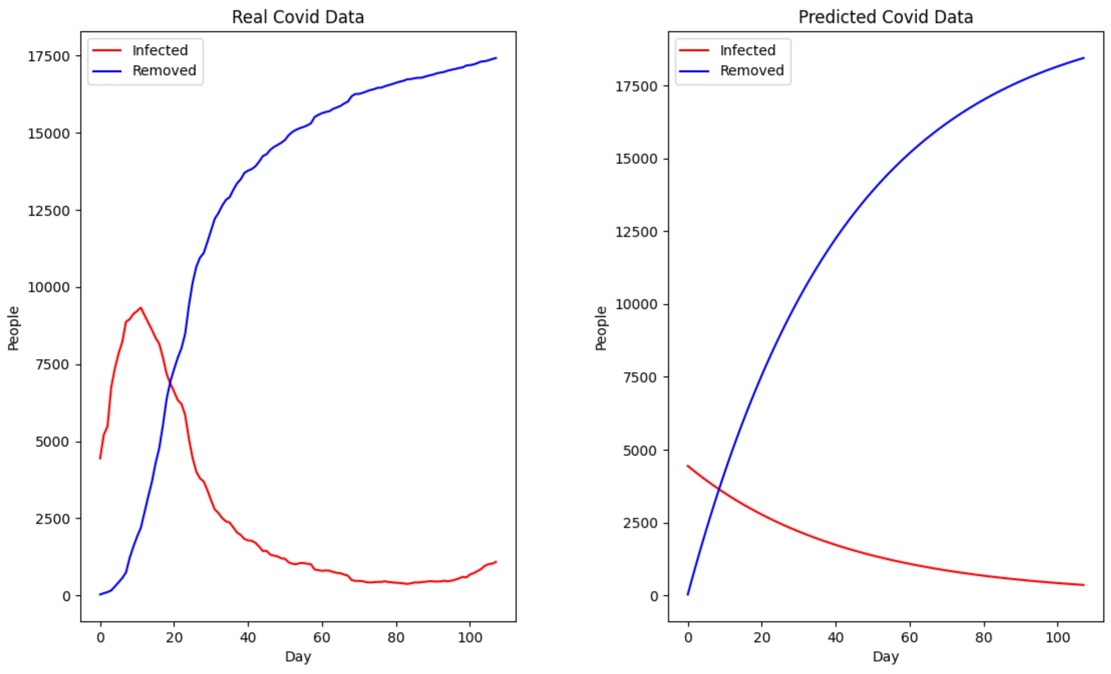
```
#! Ploting Graph
I_predict, R_predict = rk4.RK4SIR(1000000000, I0, R0, betaGamma[0], betaGamma[1], time,
    1)
x = []
for i in range(0, time + 1):
    x.append(i)

plt.figure(figsize=(12,7))
# Real Data
plt.subplot(121)
plt.plot(x, data[0], label="Infected", color='red')
plt.plot(x, data[1], label="Removed", color='blue')
plt.xlabel('Week')
plt.ylabel('People')

plt.yscale('linear')
plt.title('Real Covid Data')
plt.legend()

```

```
# Predicted Data
plt.subplot(122)
plt.plot(x, I_predict, label="Infected", color='red')
plt.plot(x, R_predict, label="Removed", color='blue')
plt.xlabel('Week')
plt.ylabel('People')

plt.yscale('linear')
plt.title('Predicted Covid Data')
plt.legend()

plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.25, wspace
    =0.35)
plt.show()
```

2.6.3 Exercise 5 - Plotting Results

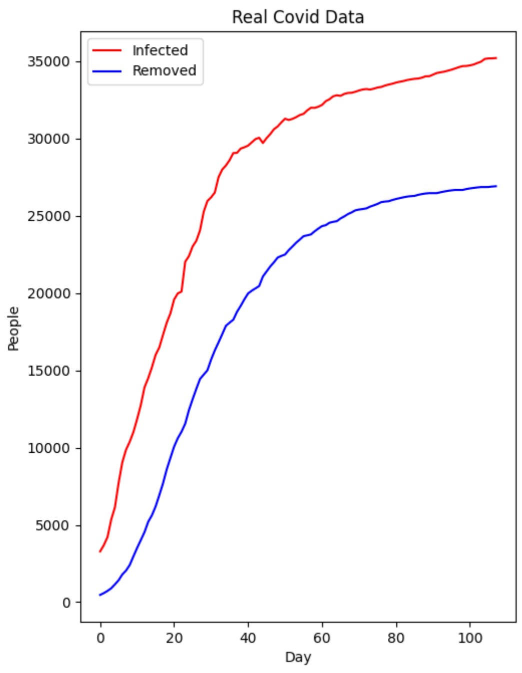
Austria:

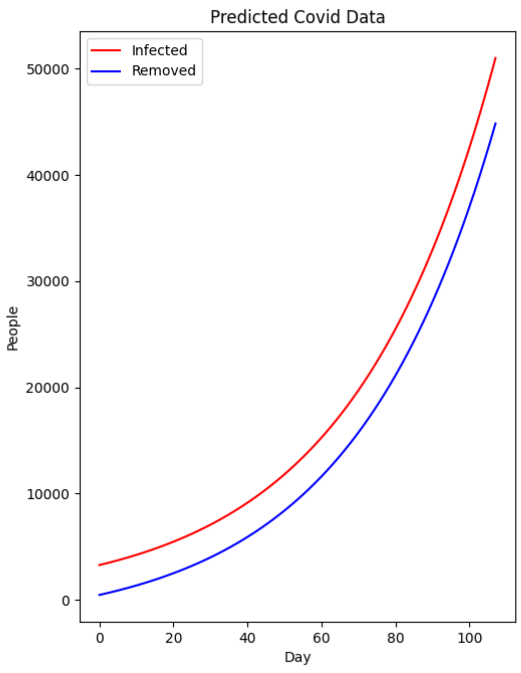
- Start date: 3/21/2020
- End date: 7/7/2020
- Number of Generation: 10000
- Final β: 0.08261414389469697
- Final γ: 0.10616970863299684
- Loss: 2900914.241658367



Belgium:

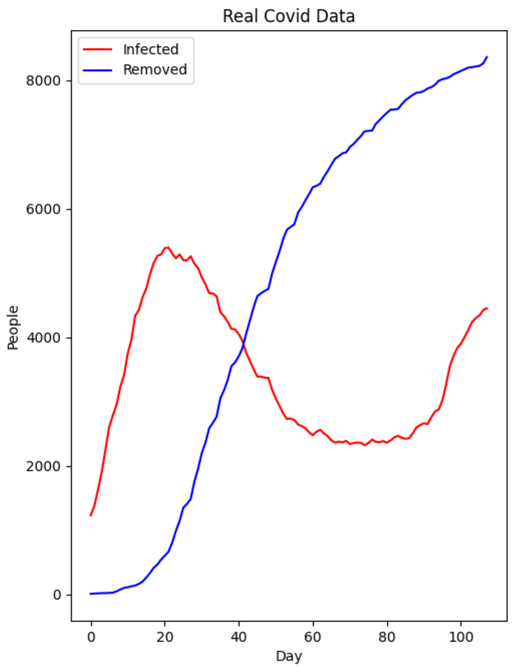
- Start date: 3/21/2020
- End date: 7/7/2020
- Number of Generation: 10000
- Final β: 0.04950314791362771
- Final γ: 0.023850517722867056
- Loss: 145353181.81186876

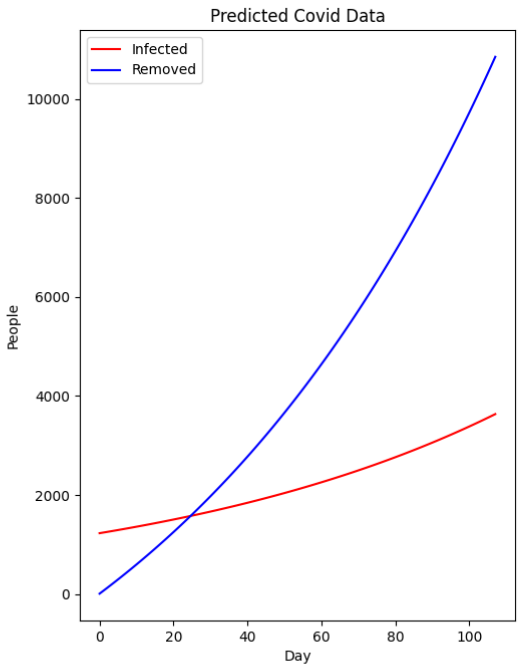




Czechia:

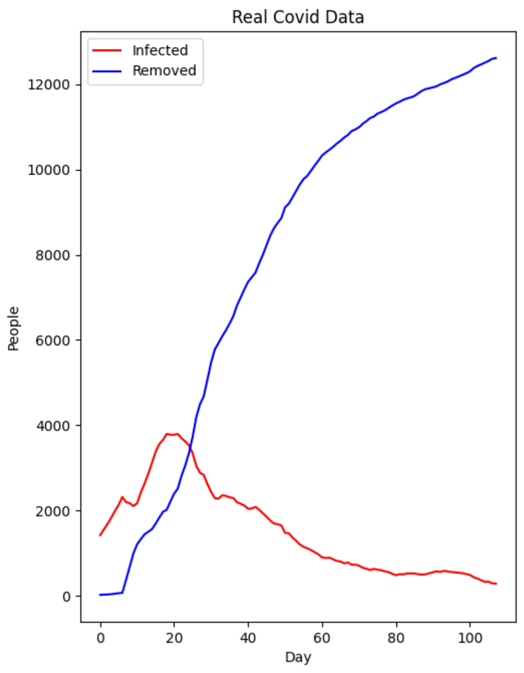
- Start date: 3/21/2020
- End date: 7/7/2020
- Number of Generation: 10000
- Final β: 0.055822538072428936
- Final γ: 0.04568700712825702
- Loss: 2339607.771273511

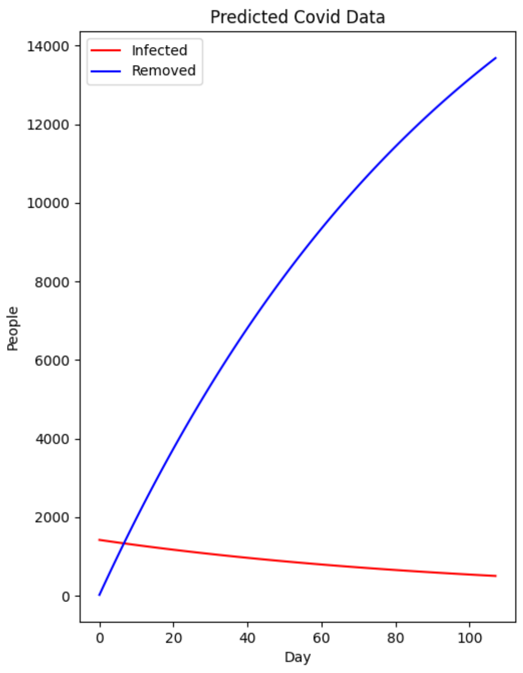




Denmark:

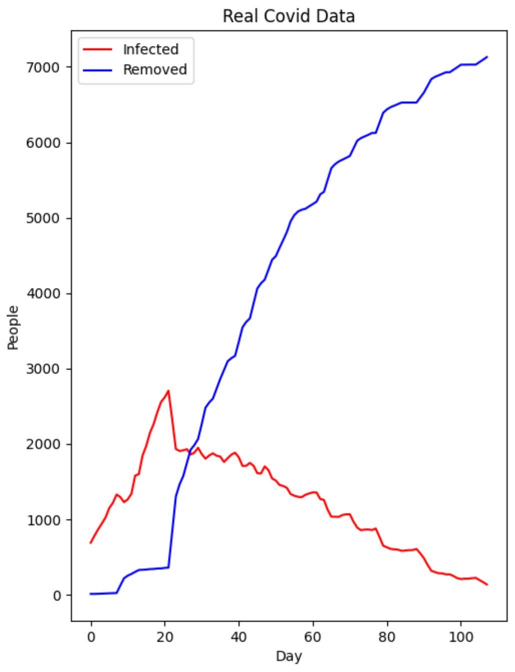
- Start date: 3/21/2020
- End date: 7/7/2020
- Number of Generation: 10000
- Final β: 0.1339769450044817
- Final γ: 0.14361893176363139
- Loss: 850400.5881574824

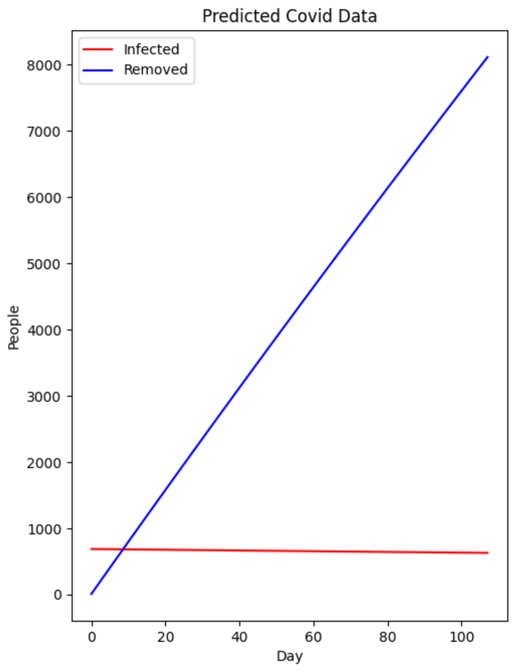




Finland:

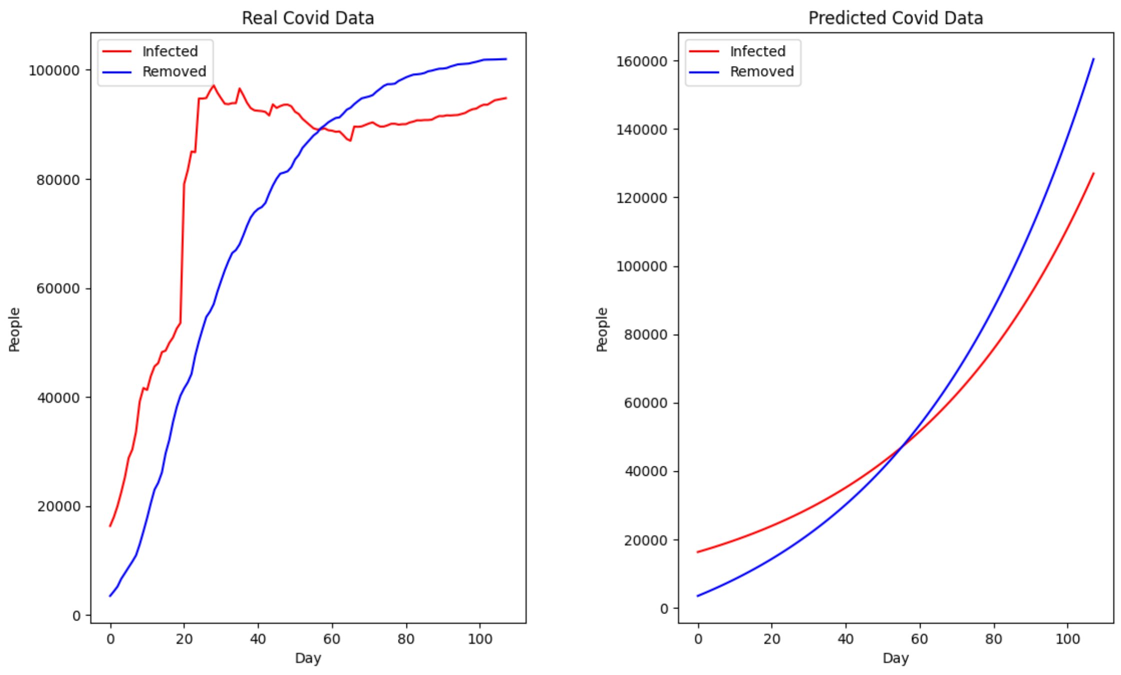
- Start date: 3/21/2020
- End date: 7/7/2020
- Number of Generation: 10000
- Final β: 0.114039185029659
- Final γ: 0.11487348612529852
- Loss: 493898.796715808





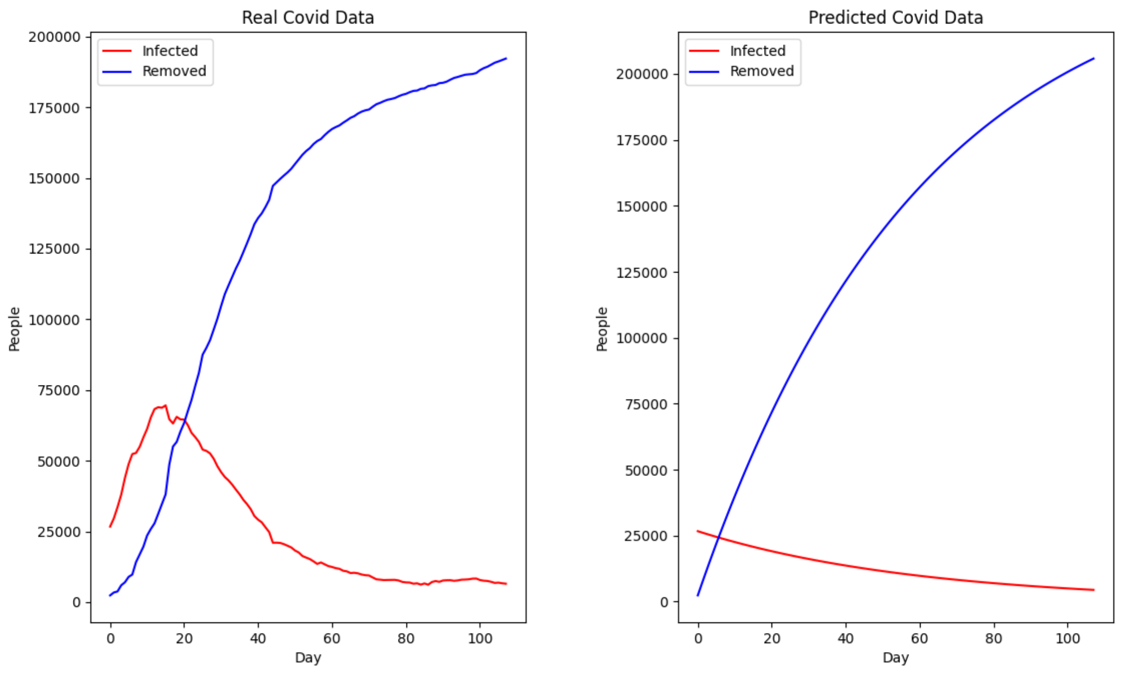
France:

- Start date: 3/21/2020
- End date: 7/7/2020
- Number of Generation: 10000
- Final β: 0.04633654638644967
- Final γ: 0.027170784222120638
- Loss: 1239411674.187388



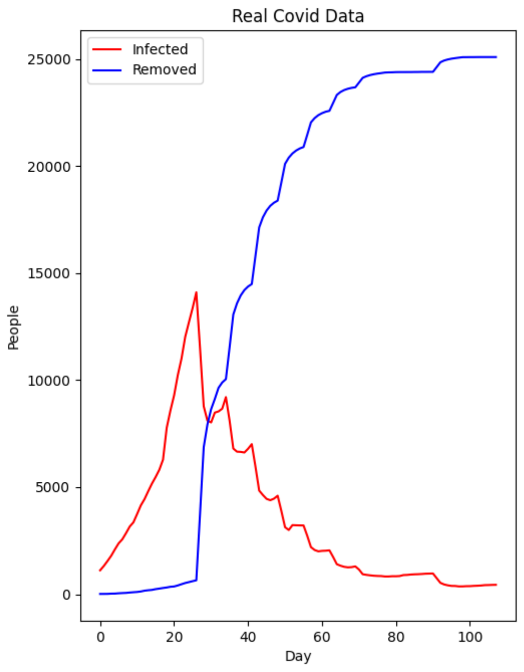
Germany:

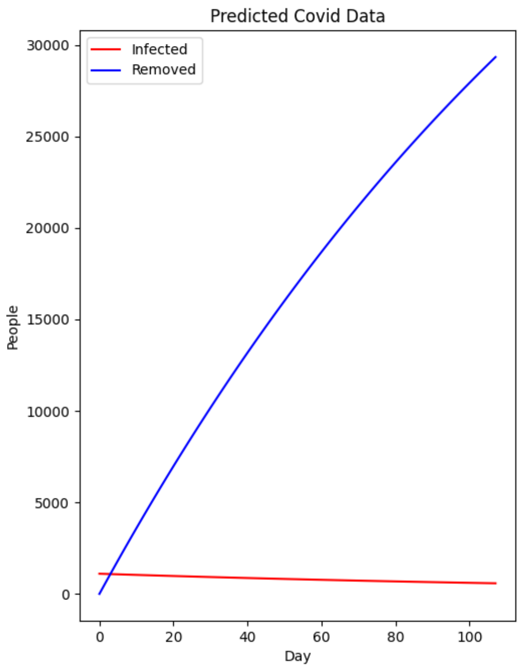
- Start date: 3/21/2020
- End date: 7/7/2020
- Number of Generation: 10000
- Final β: 0.1362806160946653
- Final γ: 0.1529868333277667
- Loss: 277737758.93634045



Ireland:

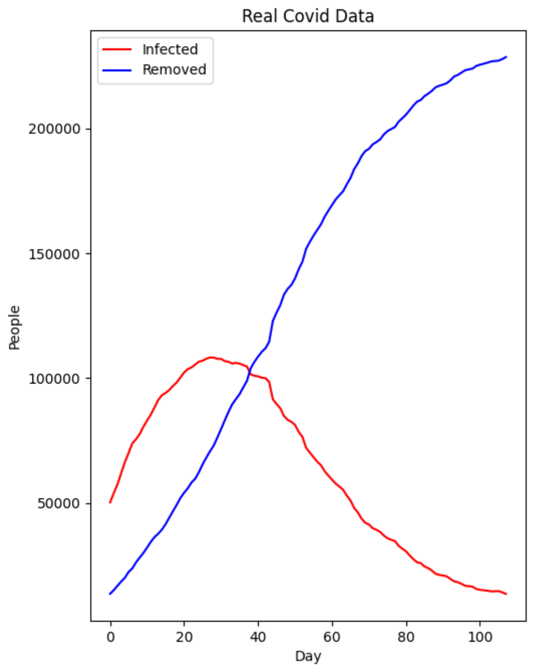
- Start date: 3/21/2020
- End date: 7/7/2020
- Number of Generation: 10000
- Final β: 0.3259669200892496
- Final γ: 0.3318723646336088
- Loss: 15315810.569687268

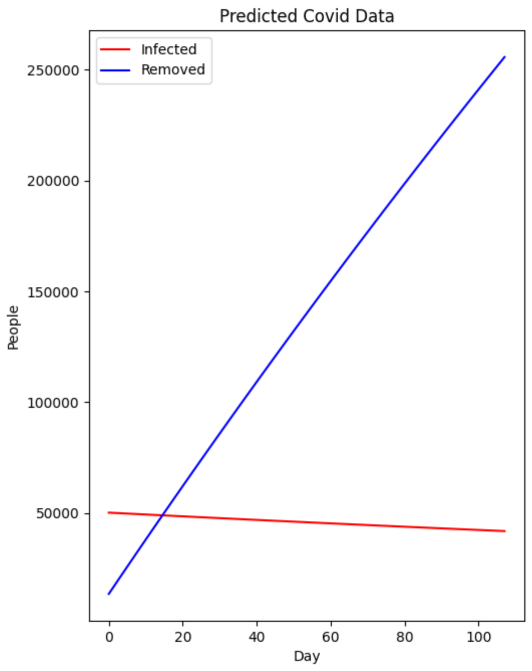




Italy:

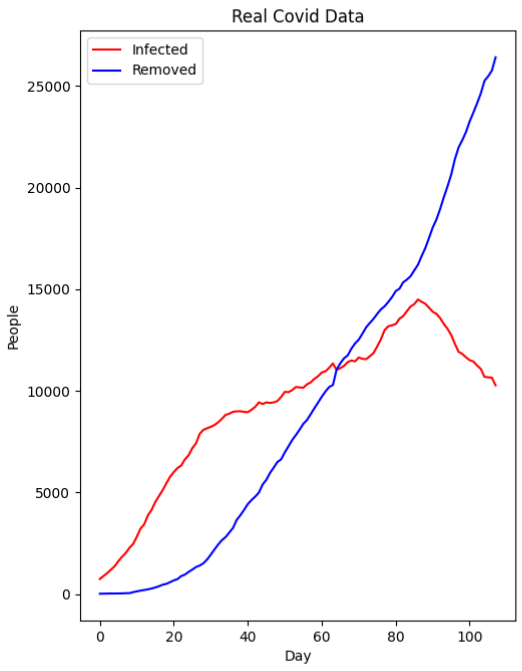
- Start date: 3/21/2020
- End date: 7/7/2020
- Number of Generation: 10000
- Final β: 0.04747182556627497
- Final γ: 0.0491518304164244
- Loss: 676299376.2818551

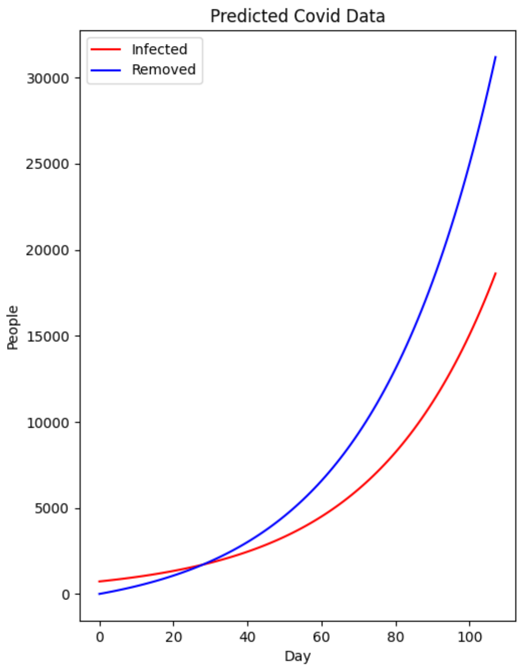




Poland:

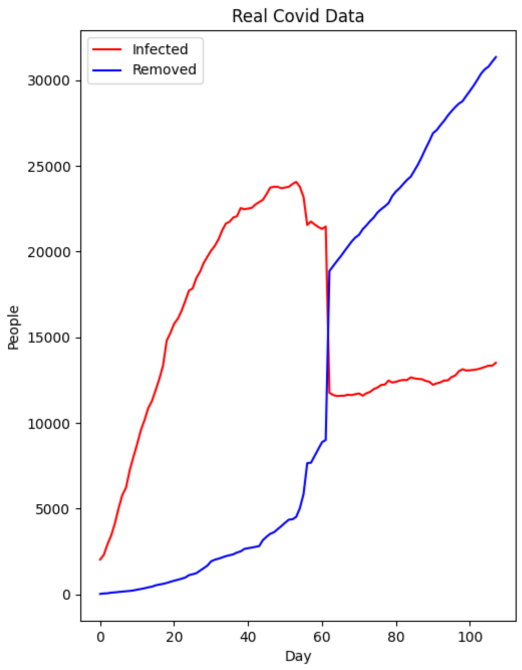
- Start date: 3/21/2020
- End date: 7/7/2020
- Number of Generation: 10000
- Final β: 0.08280057631634499
- Final γ: 0.05261566866900815
- Loss: 15200404.856771646

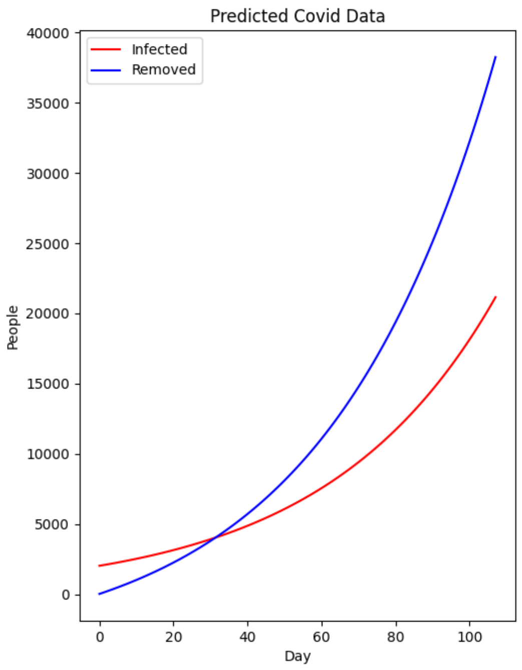




Portugal:

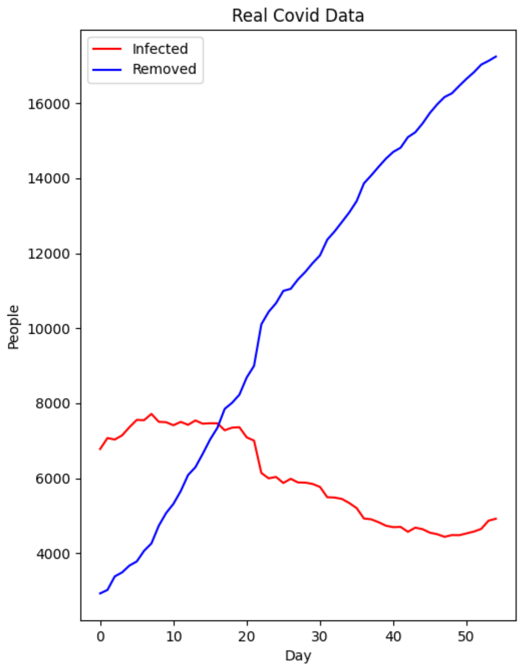
- Start date: 3/21/2020
- End date: 7/7/2020
- Number of Generation: 10000
- Final β: 0.06566296591854241
- Final γ: 0.043766732501216765
- Loss: 64874679.73756245

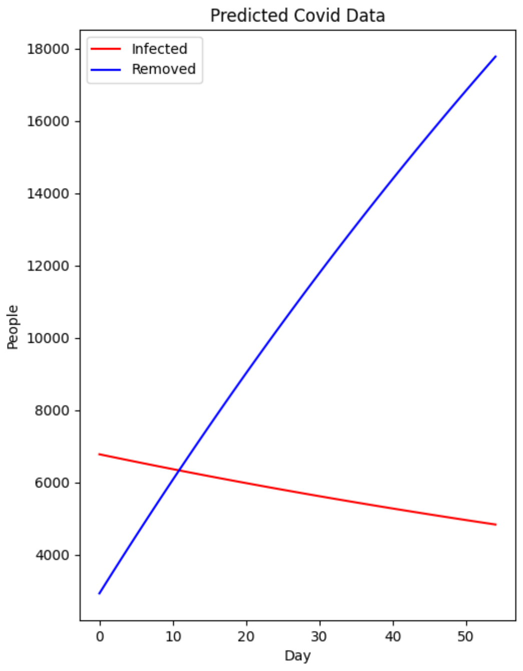




Romania:

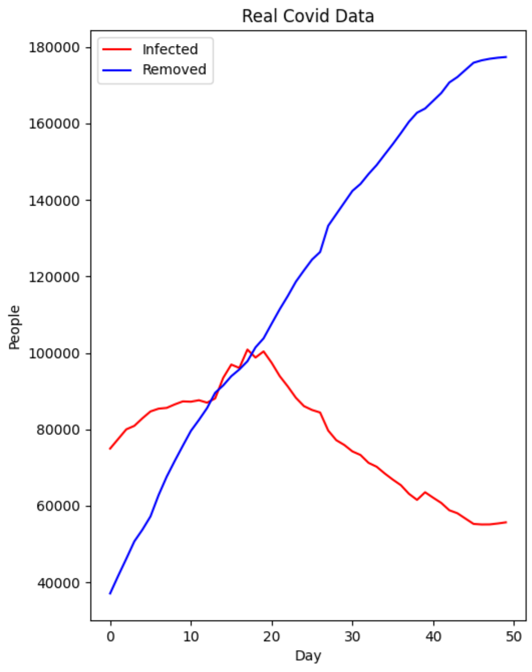
- Start date: 4/20/2020
- End date: 6/14/2020
- Number of Generation: 10000
- Final β: 0.04155789340282452
- Final γ: 0.04781414893777089
- Loss: 363462.5800399513

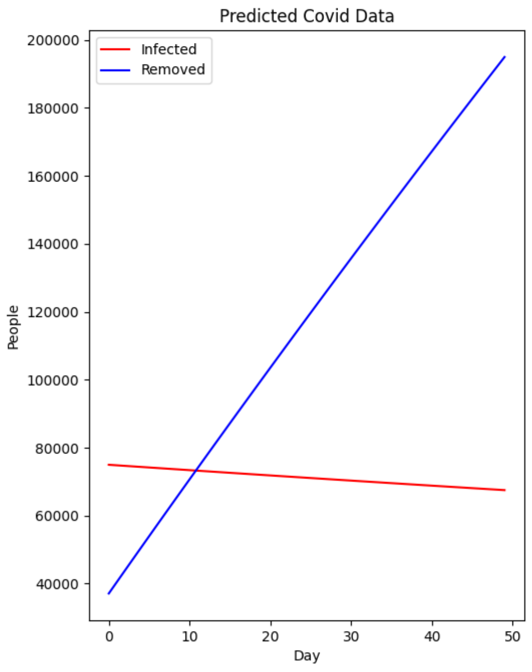




Spain:

- Start date: 3/31/2020
- End date: 5/20/2020
- Number of Generation: 10000
- Final β: 0.043141937893975184
- Final γ: 0.045273619630837356
- Loss: 116182856.16206549





3 Conclusion

As governments continue to respond to COVID-19, it is imperative to study what measures are effective and which are not. While the information presented here do, of course, can not describe and predict the whole situation, they can provide useful insights that help governments adopted an evidence-based approach to the measures they deploy.

It is our hope that scholars, medical professionals, policymakers, and concerned citizens will make use of our information to enhance all countries' response.

References

[1] S. T. Ho Lam and A. Suchard Marc. Simple MCMC under SIR. 2005. URL: https://cran.r-project.org/web/packages/MultiBD/vignettes/SIR-MCMC.pdf

[2] T. Wu Joseph, Leung Kathy, and Leung Gabriel. "Nowcasting and forecasting the potential domestic and international spread of the 2019-nCoV outbreak originating in Wuhan, China: a modelling study". In: 395 (2020).

[3] Frank Giordano, William P Fox, and Steven Horton. A first course in mathematical modeling. Nelson Education, 2013.

[4] W. K. Hastings. "Monte Carlo Sampling Methods Using Markov Chains and Their Applications". In: Biometrika 57 (1) (1970), pp. 97–109.

[5] Luca Magri and Nguyen Anh Khoa Doan. "First-principles Machine Learning for COVID-19 Modeling". In: arXiv preprint arXiv:2004.09478 (2020)