COVID-19

Mathematical model played a critical role during the COVID-19 pandemic. One of the simplest models for an infectious disease is the susceptible-infectious-removed (SIR) model, that divides the population in three groups. Susceptible individuals can become infected when they contact an infectious individual, and infectious individuals are removed either by recovering or by death.

Question 1

Assuming that the population is *closed*, we can consider only the susceptible and infectious compartments of the model. Let x be the fraction of the population in the susceptible compartment, and y the fraction in the infectious compartment. Note that the *closed* population assumption means that $1-x-y$ is the fraction in the removed compartment. The SIR model ordinary differential equations are

$$\begin{aligned} x' &= -\beta x y, \\ y' &= \beta x y - \gamma y, \end{aligned}$$

where β models the chances of infection when a susceptible meets an infectious individual, and γ is the recovery rate (how long it takes to recover or to die from the disease).

(a) Implement a function `euler(beta,gamma,x0,y0,T,h)` that receives values for the model parameters, the initial values of the variables x, y and the final simulation time T (we are assuming the initial time is $t = 0$), and the step h, and applies the Euler method to solve the SIR system. Your function should return a dataframe with three columns, t, x and y. Test it with $x_0 = 0.9$, $y_0 = 0.1$, $\beta = 0.6$, $\gamma = 0.2$, $h = 10^{-1}$ and $T = 40$, and display the first few rows of the dataframe with `head`.

```r
euler <- function(beta, gamma, x0, y0, T, h) {
  # Initialize time sequence from 0 to T with step h
  times <- seq(0, T, by = h)
  n <- length(times)
```

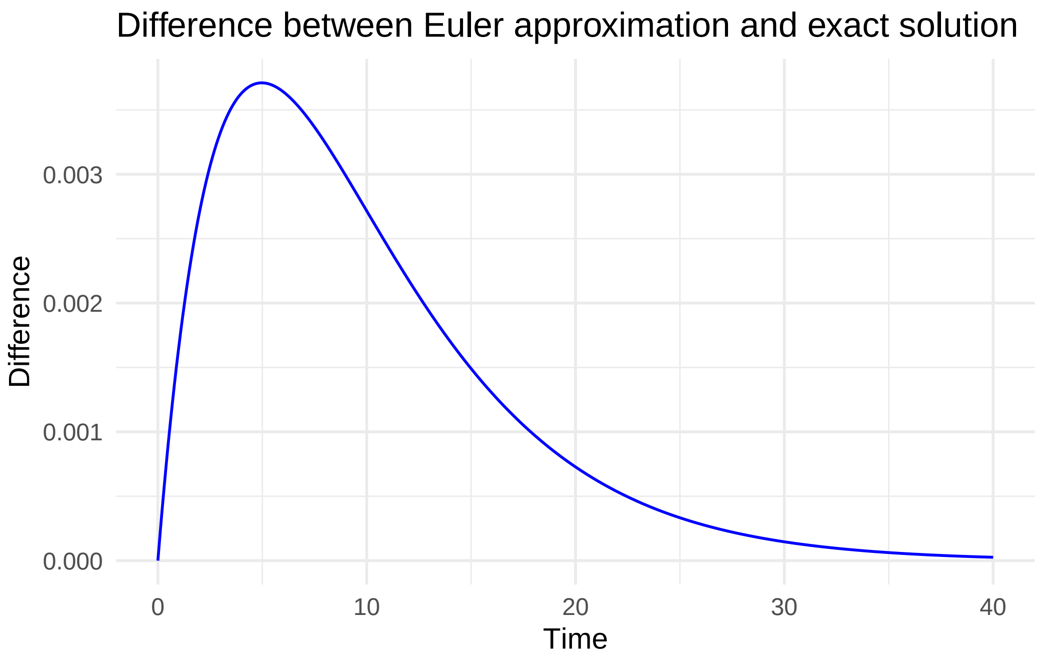
```
    # Initialize vectors
    x <- numeric(n)
    y <- numeric(n)

    # Initialise values
    x[1] <- x0
    y[1] <- y0

    # Update x & y
    for (i in 1:(n-1)) {
      x[i+1] <- x[i] - beta * x[i] * y[i] * h
      y[i+1] <- y[i] + (beta * x[i] * y[i] - gamma * y[i]) * h
    }

    # dataframe stores time, susceptible and infected fractions
    result <- data.frame(t = times, S = x, I = y)
    return(result)
  }
  result <- euler(beta = 0.6, gamma = 0.2, x0 = 0.9, y0 = 0.1, T = 40, h = 0.1)
  head(result)
```

```
    t         S         I
1 0.0 0.9000000 0.1000000
2 0.1 0.8946000 0.1034000
3 0.2 0.8890499 0.1068821
4 0.3 0.8833485 0.1104459
5 0.4 0.8774948 0.1140907
6 0.5 0.8714879 0.1178157
```

(b) If the fraction of susceptibles is 0, the equation for the infectious can be solved analytically (it is an exponential function). Solve the SIR model with $x_0 = 0$, $y_0 = 1$, $\beta = 0.6$, $\gamma = 0.2$, $h = 10^{-1}$ and $T = 40$, and add a column `exact` to the dataframe produced by the function `euler` with the exact solution at the corresponding time t. Plot the difference between the exact solution and the Euler method as a function of time, using `geom_line`.

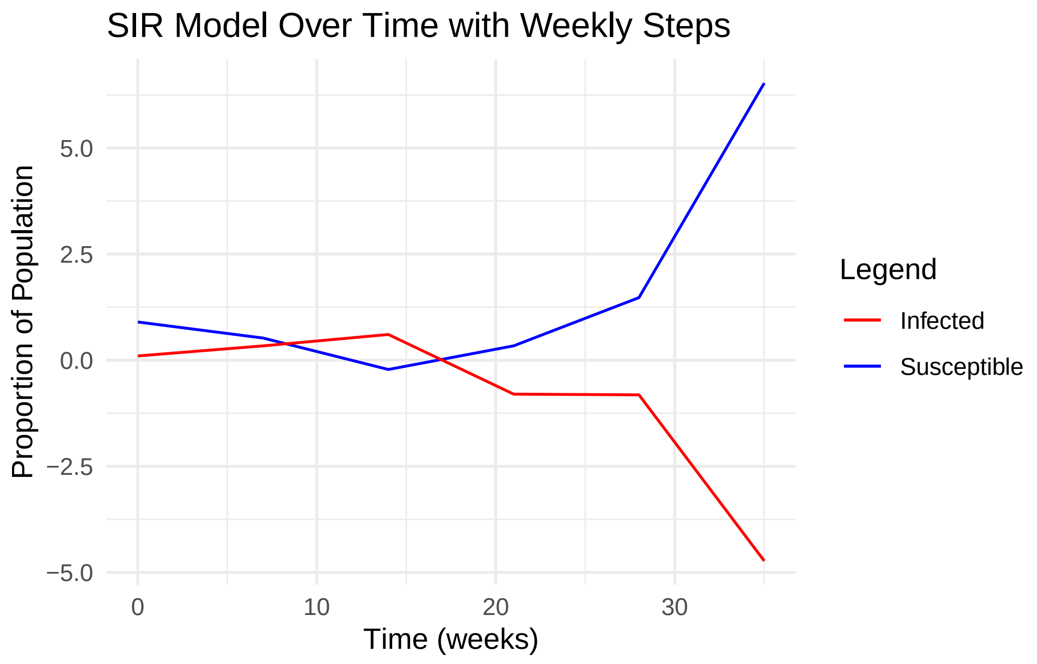
```
library(ggplot2)
library(purrr)
library(dplyr)

# Calculate the result of euler function
result <- euler(beta = 0.6, gamma = 0.2, x0 = 0, y0 = 1, T = 40, h = 0.1)
```

```r
gamma <- 0.2

# Calculate exact column
result$exact <- exp(-gamma*result$t)

# Calculate difference of exact and euler method
result$difference <- abs(result$I - result$exact)

# Plot the difference between the Euler method and the exact solution
ggplot(result, aes(x = t, y = difference)) +
  geom_line(color = "blue") +
  labs(title = "Difference between Euler approximation and exact solution",
       x = "Time",
       y = "Difference") +
  theme_minimal()
```



(c) If we are interested on the weekly cases, it might seem reasonable to use a time step h of a week. Use the `euler` function to solve the SIR model with the same parameters and initial values as in Question 1a, but use $h = 7$ days. Plot x and y as a function of time, in the same plot with two different colours. Write one or two sentences to give an explanation of your findings.

```r
# Execute the SIR model
result_weekly <- euler(beta = 0.6, gamma = 0.2, x0 = 0.9, y0 = 0.1, T = 40, h = 7)

# Generate a plot that displays the time evolution of susceptible and infectious groups.
ggplot(result_weekly, aes(x = t)) +
  geom_line(aes(y = S, color = "Susceptible")) +
  geom_line(aes(y = I, color = "Infected")) +
  scale_color_manual(values = c("Susceptible" = "blue", "Infected" = "red")) +
  labs(title = "SIR Model Over Time with Weekly Steps",
       x = "Time (weeks)",
       y = "Proportion of Population",
       color = "Legend") +
  theme_minimal()
```



```r
# The nagative values are likely a result of using a large step size in Euler method,
# as the next value of a variable is estimated based on the step size.
# If the step size is too large, this estimation can overshoot.
```

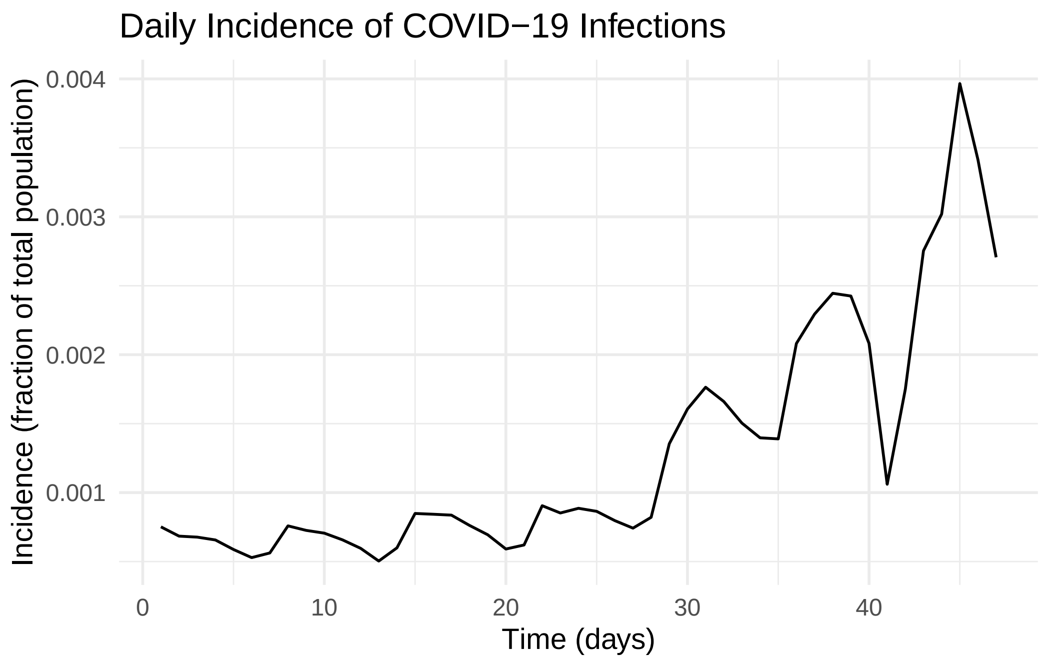
Question 2

The data in `omicron.csv` contains the number of *new* COVID-19 infected individuals during the wave caused by the Omicron variant of the virus. It useful to note that the Office for National Statistics population estimate for England from the 2021 census 2021 is 56,490,048.

In the SIR model $x(t)$, $y(t)$ and $z(t)$ are the total proportion of individuals in each compartment at any given time. In this model infected individuals remain in the $y(t)$ compartment for several days. After being infected for several days individuals recover and move into the recovered compartment $z(t)$. It is important to realise that $y(t)$ is **not** the proportion of new infected individuals per time increment (h). If $h = 1$ day then the proportion of new infected individuals (that we will call *incidence*) per day is $y(kh) - y((k-1)h)$ where $k = 1, 2, 3, \ldots$.

(a) Add columns to the dataframe to represent time in days (the first row being $t = 1$) in a column `t`, and the number of new cases expressed as a fraction of the total population in a column called `incidence`. Plot `incidence` vs `t`.

```r
library(tidyverse)
library(lubridate)

# Load the data
omicron_data <- read.csv("omicron.csv", stringsAsFactors = FALSE)

# Process data
omicron_data$cases <- as.numeric(gsub(" ", "", omicron_data$cases))
omicron_data$t <- seq(1, nrow(omicron_data), by=1)

# Calculate the incidence as a fraction of the total population
total_population <- 56490048
omicron_data$incidence <- (omicron_data$cases / total_population)

# Plot incidence vs date
ggplot(omicron_data, aes(x = t, y = incidence)) +
  geom_line() +
  labs(title = "Daily Incidence of COVID-19 Infections",
       x = "Time (days)",
       y = "Incidence (fraction of total population)") +
  theme_minimal()
```



(b) Implement a function `incidence(beta,gamma,x0,y0,T,h)` that receives the same parameters as `euler` but returns a dataframe with to columns, time t and the corresponding incidence at each time. Test it with $x_0 = 0.9$, $y_0 = 0.1$, $\beta = 0.6$, $\gamma = 0.2$, $h = 10^{-1}$ and $T = 40$, and display the first few rows of the dataframe with `head`.

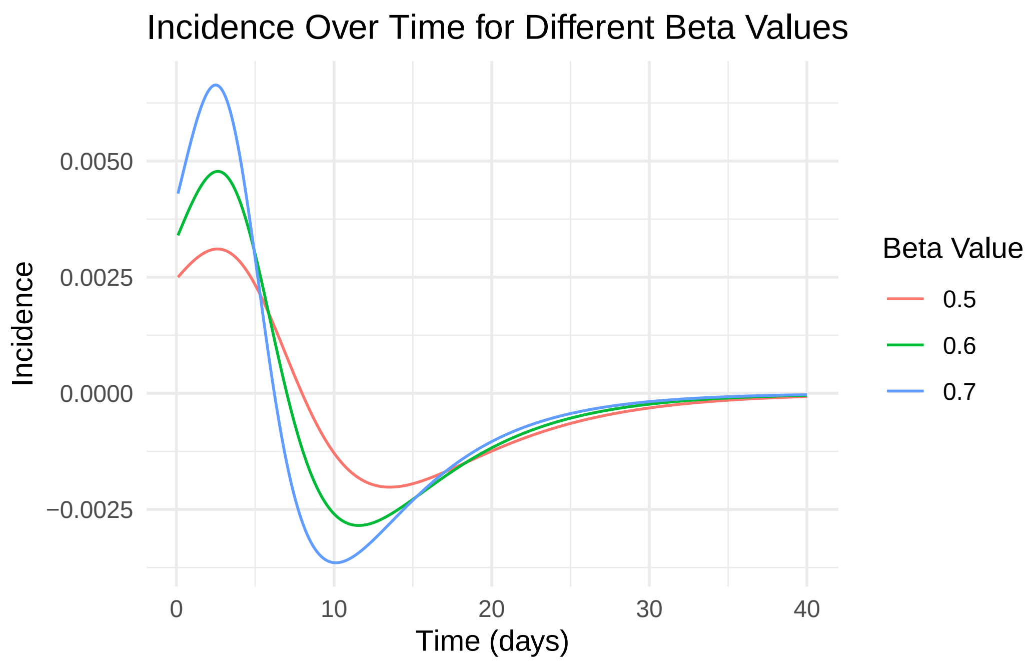
```r
incidence <- function(beta, gamma, x0, y0, T, h) {
  times <- seq(0, T, by = h)
  incidence_values <- numeric(length(times) - 1)

  # Initialize current state
  x <- x0
  y <- y0

  # Calculate the incidence at each time step
  for (k in 2:length(times)) {
    # Apply the SIR model
    x_new <- x - h * beta * x * y
    y_new <- y + h * (beta * x * y - gamma * y)

    # Incidence is the change in y
    incidence_values[k - 1] <- y_new - y
    # Update state
    x <- x_new
    y <- y_new
  }

  # Dataframe with time and incidence
  incidence_data <- data.frame(time = times[-1], incidence = incidence_values)
  return(incidence_data)
}

result_incidence <- incidence(beta = 0.6, gamma = 0.2, x0 = 0.9, y0 = 0.1, T = 40,
                              h = 0.1)
head(result_incidence)
```

```
  time   incidence
1  0.1 0.003400000
2  0.2 0.003482098
3  0.3 0.003563769
4  0.4 0.003644814
5  0.5 0.003725025
6  0.6 0.003804184
```

(c) Write a function factory to fix the values of `gamma,x0,y0,T,h`. Use it to plot the incidence for $\beta = 0.5, 0.6, 0.7$ as a function of time, for $\gamma = 0.2, h = 10^{-1}, x_0 = 0.9, y_0 = 0.1$.

```r
# Define function factory to fix the values of gamma,x0,y0,T,h
function_factory <- function(gamma, x0, y0, T, h) {
  function(beta) {
    incidence(beta, gamma, x0, y0, T, h)
  }
}
specific_incidence <- function_factory(0.2, 0.9, 0.1, 40, 0.1)
# Initialize an empty data frame
plot_data <- data.frame()

# Generate data for each beta value
for (beta in c(0.5, 0.6, 0.7)) {
  beta_data <- specific_incidence(beta)
  beta_data$beta <- as.factor(beta)
  plot_data <- rbind(plot_data, beta_data)
}
```

```r
# Plot the incidence for different beta values
ggplot(plot_data, aes(x = time, y = incidence, color = beta)) +
  geom_line() +
  labs(title = "Incidence Over Time for Different Beta Values",
       x = "Time (days)",
       y = "Incidence",
       color = "Beta Value") +
  theme_minimal()
```



Question 3

(a) Create a function `error(beta,gamma,x0,y0,data)` that receives the same parameters as the function `euler` and a dataframe `data` that contains a column `t` and a column `incidence` (the incidence in the real data), and returns the squared difference $\sum(m_t - d_t)^2$, where m_t is the incidence from the model solution and d_t is the incidence in the data, and the sum is over all the *data* points t. For the Euler method, use the step h at which the data is defined (i.e. 1 day), and similarly use the maximum time T from the data. Test your function with $\beta = 0.6, \gamma = 0.1, x_0 = 0.9, x_1 = 0.1$

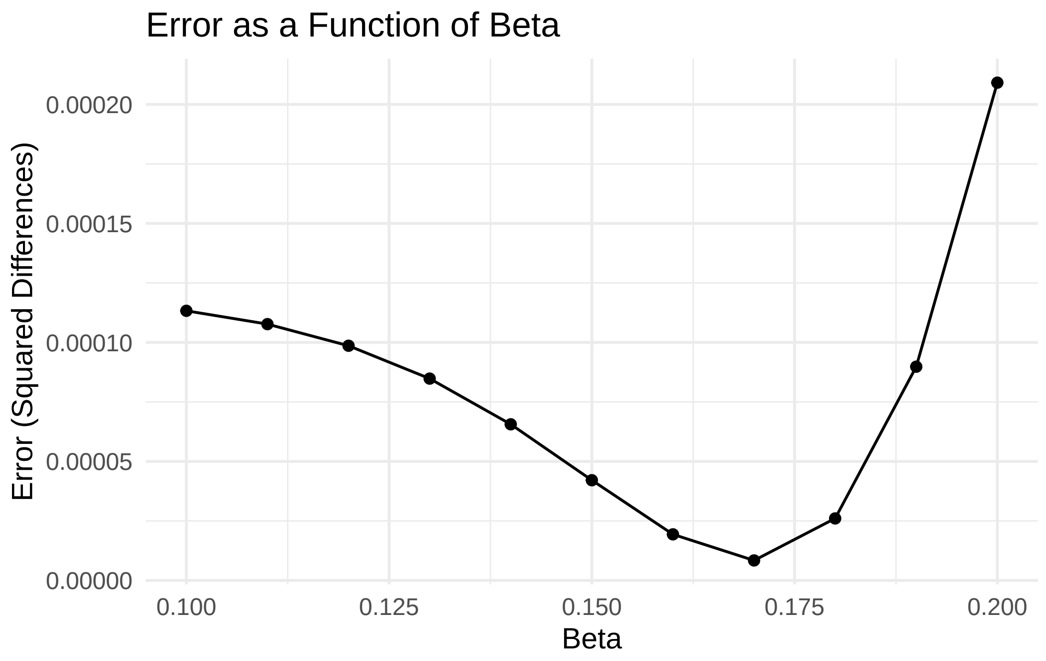
```r
# Reassign the actual incidence data
actual_data <- omicron_data

# Define the error function
error <- function(beta, gamma, x0, y0, data) {
  T <- max(data$t)
  h <- data$t[2] - data$t[1]

  # Generate model data
  model_data <- incidence(beta, gamma, x0, y0, T, h)

  # Compute the sum of squared differences between model and actual incidence
  squared_diffs <- sum((model_data$incidence - data$incidence)^2)

  return(squared_diffs)
}

# Calculate the error for the given parameters
calculated_error <- error(beta = 0.6, gamma = 0.1, x0 = 0.9, y0 = 0.1,
                          data = actual_data)
print(calculated_error)
```

```
[1] 0.049242
```

(b) Implement a function factory `factory_error` to fix the values of `gamma,x0,y0,data`. Plot the error as a function of β for $\gamma = 0.1, x_0 = 1 - 0.004, y_0 = 0.004$, and $\beta = 0.1, 0.11, 0.12, \dots, 0.2$.

```r
# Define a function factory
factory_error <- function(fixed_gamma, fixed_x0, fixed_y0, fixed_data) {
  function(beta) {
    return(error(beta, fixed_gamma, fixed_x0, fixed_y0, fixed_data))
  }
}

# Create a specific error function with fixed parameters
specific_error_func <- factory_error(0.1, 1 - 0.004, 0.004, actual_data)

# Define a range of beta values
beta_values <- seq(0.1, 0.2, by = 0.01)

# Compute errors for each beta value
```

```r
errors <- sapply(beta_values, function(beta) specific_error_func(beta))

# Combine beta values and errors into a dataframe for plotting
error_data <- data.frame(beta = beta_values, error = errors)
ggplot(error_data, aes(x = beta, y = error)) +
  geom_line() +
  geom_point() +
  labs(title = "Error as a Function of Beta",
       x = "Beta",
       y = "Error (Squared Differences)") +
  theme_minimal()
```



(c) Write a function factory `findmin_factory(gamma,x0,y0,data)` that returns `findmin(beta_min,beta_max,N)` to evaluate the error at N equidistant values of β between `beta_min` and `beta_max`. Use it to find the value of β that minimises the error for $\beta \in [0.1, 0.2]$, with $N = 100$, for $\gamma = 0.1, x_0 = 1 - 0.004, y_0 = 0.004$ and the omicron data.

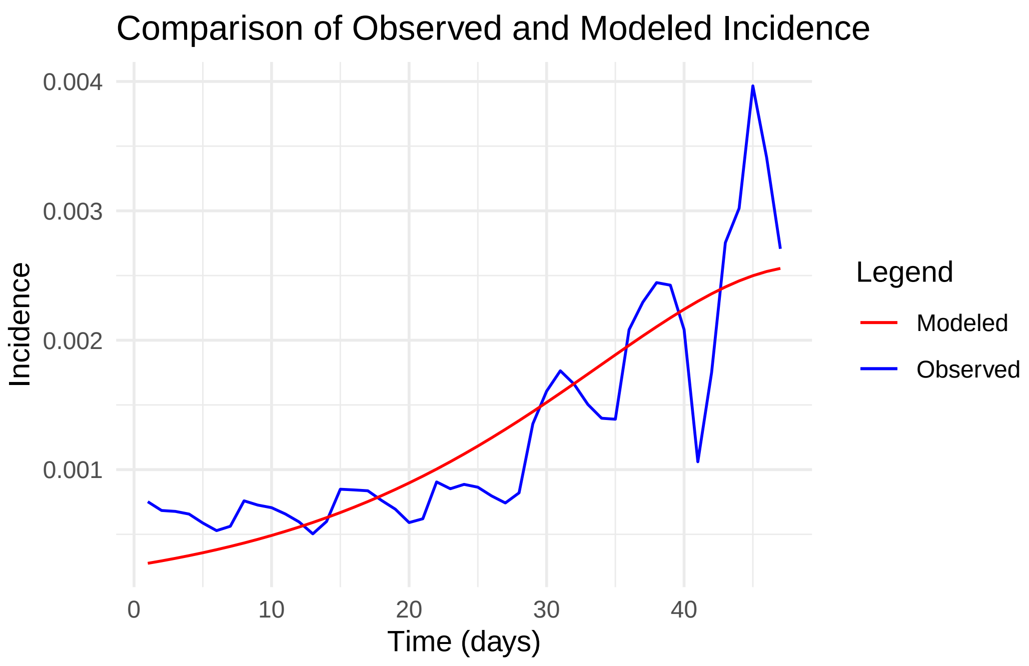
```r
# Define a function factory for finding the minimum error beta
findmin_factory <- function(gamma, x0, y0, data) {
  # Factory function to find the optimal beta within a range
  function(beta_min, beta_max, N) {
    # Generate a sequence of beta values
    betas <- seq(beta_min, beta_max, length.out = N)
    # Calculate errors for each beta value using the error function
    errors <- sapply(betas, function(beta) {
      error(beta, gamma, x0, y0, data)
    })
    min_index <- which.min(errors)
    # Return the beta value with minimum error along with the error value
    return(list(beta = betas[min_index], error = errors[min_index]))
  }
}

# Instantiate the function using the factory
findmin_error_function <- findmin_factory(gamma = 0.1, x0 = 1-0.004, y0 = 0.004,
                                          data = actual_data)
# Execute the function to find the optimal beta value that minimizes the error
optimization_result <- findmin_error_function(beta_min = 0.1, beta_max = 0.2, N = 100)
print(optimization_result)
```

```
$beta
[1] 0.169697

$error
[1] 8.396004e-06
```

(d) Plot the observed incidence and the model incidence with the optimal β, in one plot, using different colours for model and data.

```r
# Define the parameter with optimal beta
beta <- optimization_result$beta
gamma <- 0.1
x0 <- 1-0.004
y0 <- 0.004
t <- max(actual_data$t)
h <- actual_data$t[2] - actual_data$t[1]

# Get the data into a dataframe for plotting
comparison_data <- data.frame(
  t = omicron_data$t,
  incidence = omicron_data$incidence,
  model_incidence = incidence(beta, gamma, x0, y0, t, h)$incidence
)
```

```r
# Plotting the comparison
ggplot(comparison_data, aes(x = t)) +
  geom_line(aes(y = incidence, colour = "Observed")) +
  geom_line(aes(y = model_incidence, colour = "Modeled")) +
  scale_color_manual(values = c("Observed" = "blue", "Modeled" = "red")) +
  labs(title = "Comparison of Observed and Modeled Incidence",
       x = "Time (days)",
       y = "Incidence",
       color = "Legend") +
  theme_minimal()
```



Question 4

The SIR model can be extended to include a vaccinated compartment.

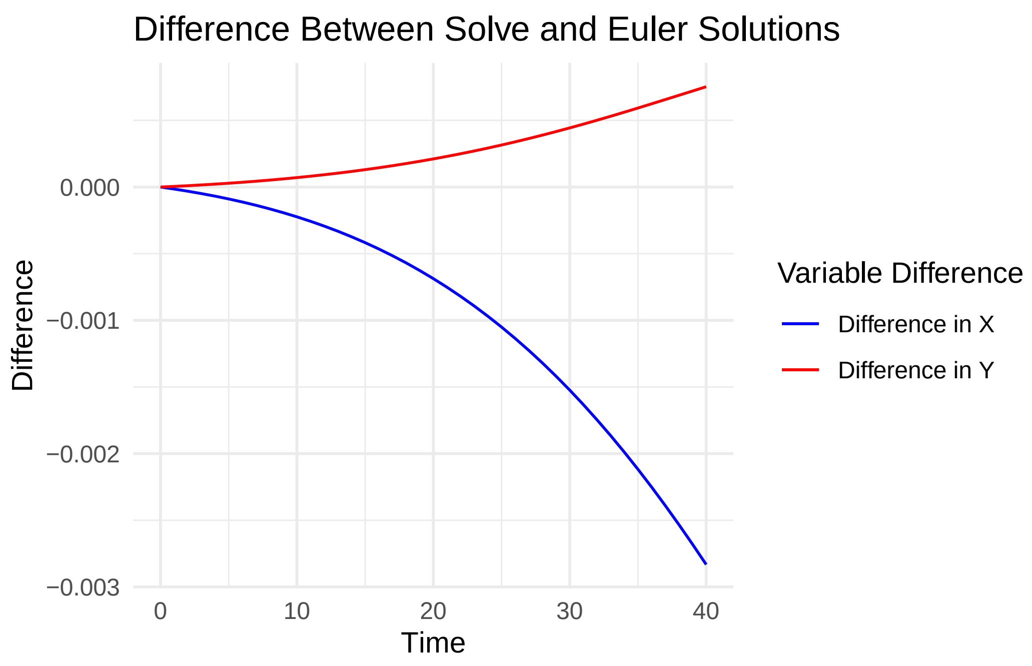
$$
\begin{aligned}
x' &= -\beta xy - \mu x, \\
y' &= \beta xy - \gamma y, \\
v' &= \mu x,
\end{aligned}
$$

where μ is a vaccination rate. We will call this model SIRV.

(a) Write a function `solve(beta,gamma,mu,x0,y0,v0,T,h)` to solve the SIRV model using `deSolve` (in R) or `scipy` (in Python). The function should return a dataframe with columns `t,x,y,v`, with the solution evaluate at times $0, h, 2h, \ldots$. Test it by plotting the difference between the solution using `solve` with `mu=0` and the solution using `euler`, for $\beta = 0.15, \gamma = 0.1, x_0 = 1 - 0.004, y_0 = 0.004, v_0 = 0, T = 40, h = 1$. Plot only the difference between solutions for the variables x and y, each variable in a different colour.

```r
library(deSolve)
# Define a function to solve the SIRV model using numerical integration
solve <- function(beta, gamma, mu, x0, y0, v0, T, h) {
  # Defines the SIRV model differential equations
  sirv_model <- function(t, state, parameters) {
    with(as.list(c(state, parameters)), {
      dx <- -beta * x * y - mu * x
      dy <- beta * x * y - gamma * y
      dv <- mu * x
      return(list(c(dx, dy, dv)))
    })
  }

  times <- seq(0, T, by = h)
  # Initial state of the model
  initial_state <- c(x = x0, y = y0, v = v0)
  # Parameters passed to the model
  parameters <- c(beta = beta, gamma = gamma, mu = mu)

  # Solving the model using the ode function from deSolve
  out <- ode(y = initial_state, times = times, func = sirv_model, parms = parameters)
  out <- as.data.frame(out)
  names(out)[2:4] <- c("x", "y", "v")
  return(out)
}

# Solve the SIRV model with mu = 0
sirv_results <- solve(beta = 0.15, gamma = 0.1, mu = 0, x0 = 1-0.004, y0 = 0.004,
                      v0 = 0, T = 40, h = 1)

# Solve the SIR model with the given Euler method
euler_results <- euler(beta = 0.15, gamma = 0.1, x0 = 1-0.004, y0 = 0.004, T = 40,
                       h = 1)

# Calculate differences SIRV and Euler results for susceptible and infected populations
```

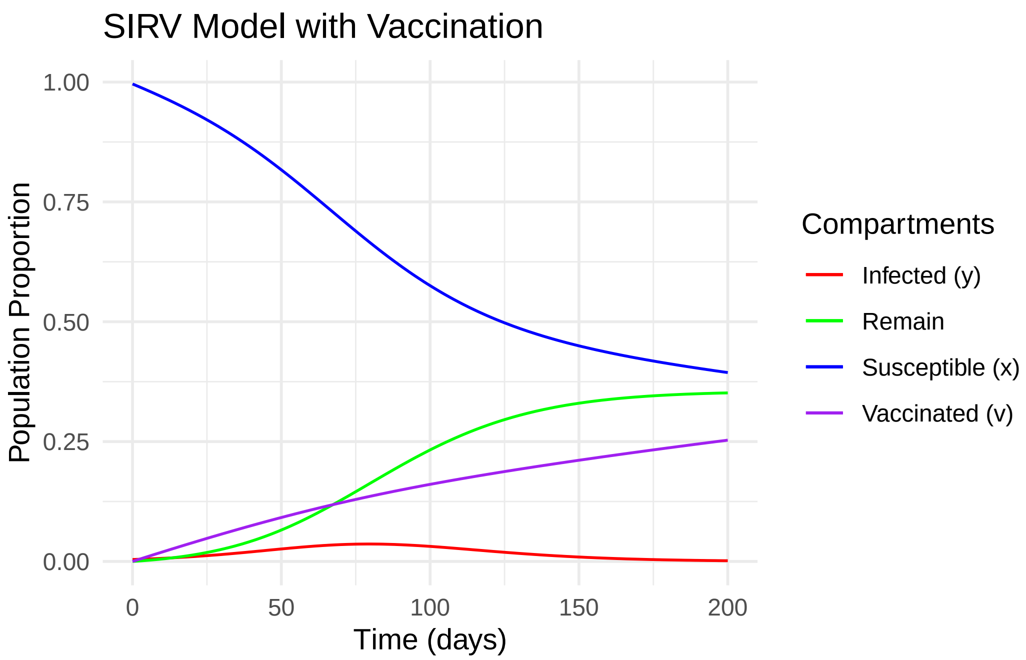
```r
differences <- data.frame(
  t = euler_results$t,
  dx = sirv_results$x - euler_results$S,
  dy = sirv_results$y - euler_results$I
)

# Plot the differences
ggplot(differences, aes(x = t)) +
  geom_line(aes(y = dx, color = "Difference in X")) +
  geom_line(aes(y = dy, color = "Difference in Y")) +
  labs(title = "Difference Between Solve and Euler Solutions",
       x = "Time", y = "Difference") +
  scale_color_manual(values = c("Difference in X" = "blue", "Difference in Y" = "red")) +
  theme_minimal() +
  guides(color = guide_legend(title = "Variable Difference"))
```



(b) Plot $x, y, 1 - x - y - v, v$, using different colours for each variable, using the same parameters as in Question 4a, with $\mu = 0.002$ and $T = 200$.

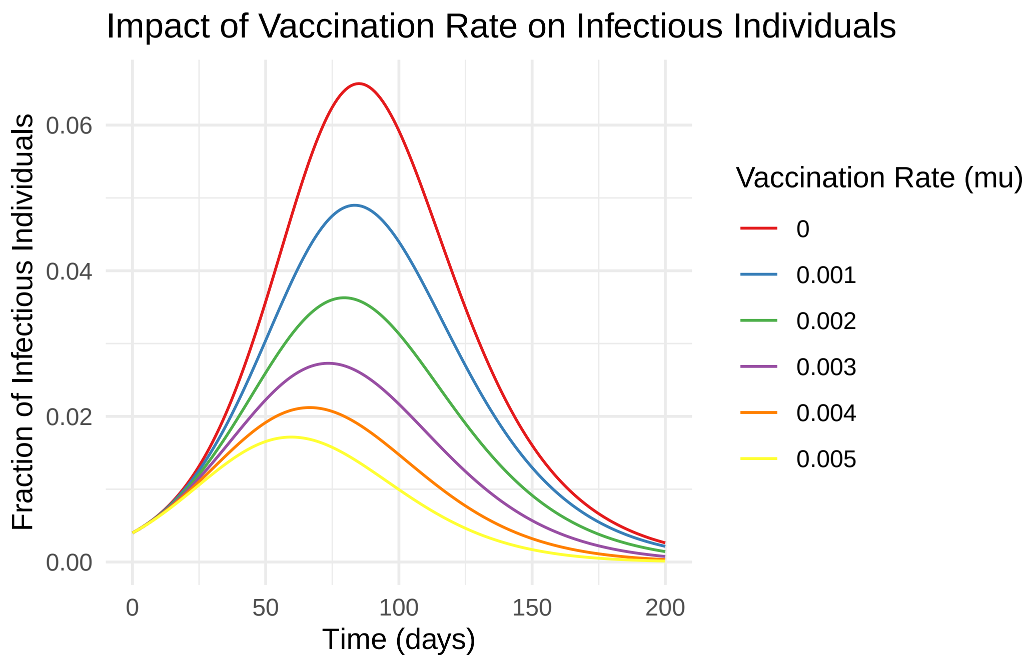
```r
sirv_results_mu <- solve(beta = 0.15, gamma = 0.1, mu = 0.002, x0 = 1-0.004,
                         y0 = 0.004, v0 = 0, T = 200, h = 1)
```

```r
# Calculate the remained population
sirv_results_mu$remain <- 1 - sirv_results_mu$x - sirv_results_mu$y - sirv_results_mu$v

# Plot the compartment within the SIRV model over time
ggplot(sirv_results_mu, aes(x = time)) +
  geom_line(aes(y = x, color = "Susceptible (x)")) +
  geom_line(aes(y = y, color = "Infected (y)")) +
  geom_line(aes(y = remain, color = "Remain")) +
  geom_line(aes(y = v, color = "Vaccinated (v)")) +
  labs(title = "SIRV Model with Vaccination",
       x = "Time (days)",
       y = "Population Proportion") +
  scale_color_manual(values = c("Susceptible (x)" = "blue",
                                "Infected (y)" = "red",
                                "Remain" = "green",
                                "Vaccinated (v)" = "purple")) +
  theme_minimal() +
  guides(color = guide_legend(title = "Compartments"))
```



(c) Plot fraction of infectious individuals using the same parameters, with $\mu = 0, 0.001, \ldots, 0.005$. Write one or two sentences discussing the impact of vaccinations in view of this results.

```r
# Define vaccination rates to test, from 0 to 0.005, increasing by 0.001
mu_values <- seq(0, 0.005, by = 0.001)
# Initialize an empty dataframe to store results from each simulation
infectious_results <- data.frame()

# Loop through each vaccination rate to simulate the model
for (mu in mu_values) {
  results <- solve(beta = 0.15, gamma = 0.1, mu = mu, x0 = 1 - 0.004, y0 = 0.004,
                   v0 = 0, T = 200, h = 1)
  results$mu <- mu
  # Combine results for all mu values into a single dataframe
  infectious_results <- rbind(infectious_results, results[, c("time", "y", "mu")])
}

# Plotting the fraction of infectious individuals for different mu values
ggplot(infectious_results, aes(x = time, y = y, color = as.factor(mu))) +
  geom_line() +
  labs(title = "Impact of Vaccination Rate on Infectious Individuals",
       x = "Time (days)",
       y = "Fraction of Infectious Individuals",
       color = "Vaccination Rate (mu)") +
  scale_color_brewer(palette = "Set1", name = "Vaccination Rate\n(mu)") +
  theme_minimal() +
  guides(color = guide_legend(title = "Vaccination Rate (mu)"))
```



```
# The graph shows that higher vaccination rates lead to fewer people getting infected and
# can delay the outbreak's peak. This suggests vaccinations are effective in controlling
# the spread of infectious diseases.
```