SISMID Exersice 1 – Solution

Fanny Bergström

2023-06-01

Exercise 1.1 (Final Outbreak Size)

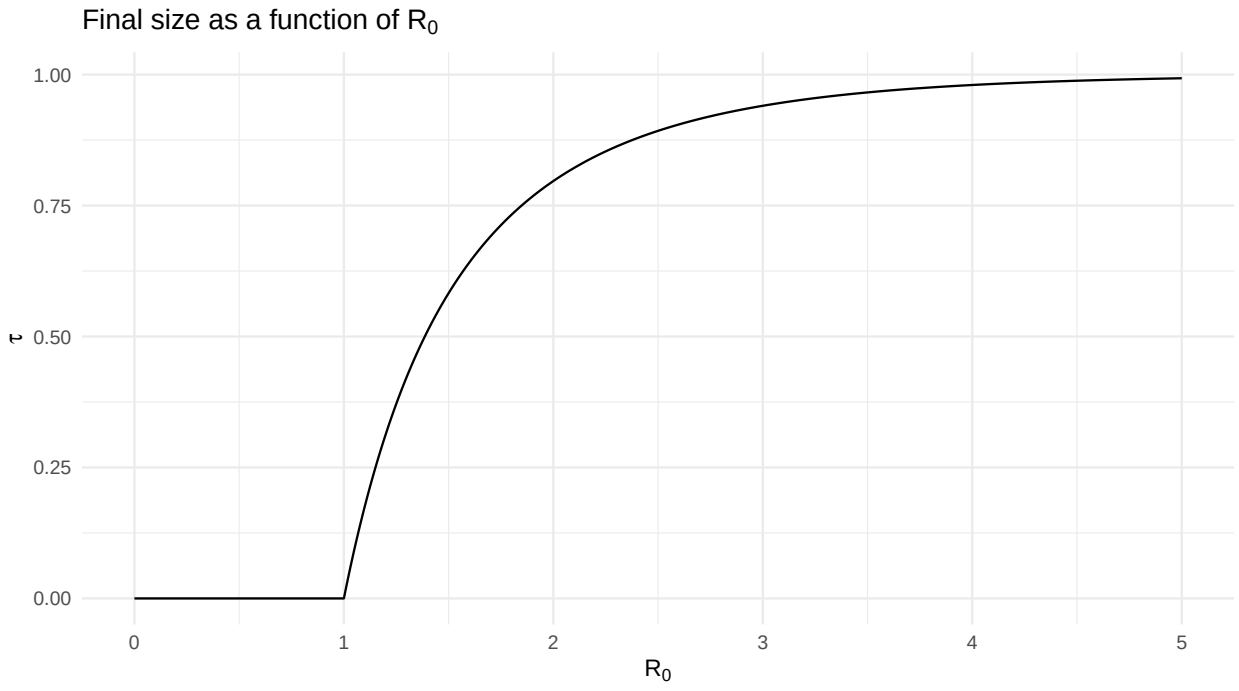
(a) Solve the final size equation. Create a function of R_0 solving for τ numerically.

```
################################################################################
# Function of R_0 solving for tau numerically
#  R_0 - reproduction number
#  tau - final size
#  Return: tau
################################################################################

tau <- function(R_0) {
  final_eq <- function(tau) {
    ##########################
    # give the expression here
    ##########################
    1 - tau - exp(-R_0 * tau)
  }
  "
  Note: This final size equation is always a solution tau = 0.
  When R_0 <=1, it gives only one solution tau = 0;
  if R_0 > 1, then there are two solutions.
  "
  result1 <- uniroot(final_eq, lower = 0, upper = 1)$root
  result2 <- uniroot(final_eq, lower = 1e-12, upper = 1, extendInt = "yes")$root

  # fill in here (minimum or maximum of result1 and 2 or others?)
  result <- max(result1, result2)

  return(result)
}
```

Compute and plot τ against $R_0 \in [0, 5]$:

```
################################################################################
#  Compute tau for R_0 in [0,5]
#  Plot tau against R_0
################################################################################

# Create a vector of 10000 R0 values between 0.00001 and 5
```

```r
R0_vec <- seq(from = 0, to = 5, length.out = 10000)

# Create a vector of the corresponding values of tau
tau_vec <- as.vector(10000)
for (i in 1:10000) {
  R0 <- R0_vec[i]
  ############################################################
  # fill in here (use the previous function tau(...) )
  tau_vec[i] <- tau(R_0 = R0)
  ############################################################
}

# Create a plot of tau against R0
ggplot(mapping = aes(x = R0_vec, y = tau_vec)) +
  geom_line() +
  xlab(expression("R"[0])) +
  ylab(expression(tau)) +
  ggtitle(expression("Final size as a function of R"[0])) +
  theme_minimal()
```



(b) Now suppose there is a fraction r of initially immune, then the final fraction infected among the initially susceptible. Create a function of R_0 and r solving for τ numerically

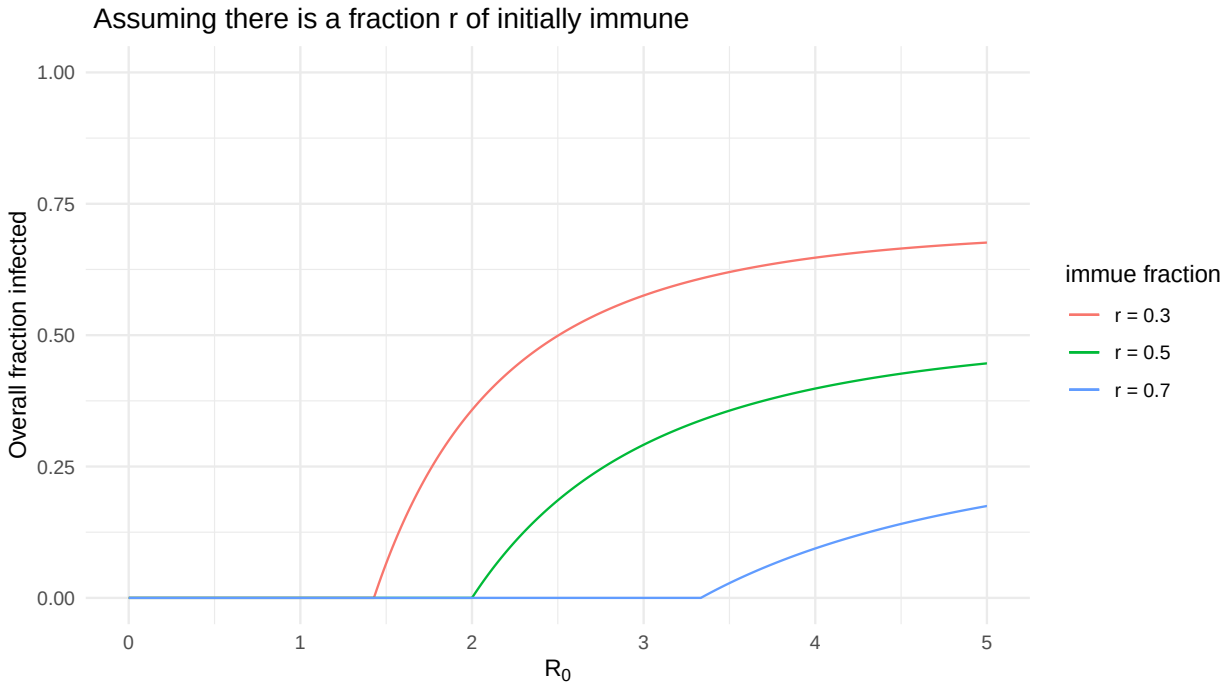
```r
################################################################################
# Function of R_0 solving for tau numerically, when there is initial immune
#  R_0 - reproduction number
#  r - fraction of initially immune
#  tau - overall final size
#  Return: tau
################################################################################
```

```r
tau_overall <- function(R_0, r) {
  final_eq <- function(tau) {
    ############################
    #    YOUR CODE HERE
    ############################
    # give the equation
    1 - tau - exp(-R_0 * (1 - r) * tau)
  }
  "
  Note: This final size equation only gives the fraction infected among
  those who are initially susceptible.
  "
  result1 <- uniroot(final_eq, lower = 0, upper = 1)$root
  result2 <- uniroot(final_eq, lower = 1e-12, upper = 1, extendInt = "yes")$root

  ######################################################
  # fill in here (minimum or maximum or others?)
  ######################################################
  result <- max(result1, result2)

  "
  Note: this result is the final size among initially susceptibles.
  then what is the overall fraction infected?
  "
  ##########################################################################
  # fill in here (here we want the OVERALL fraction infected!)
  ##########################################################################

  return(result * (1 - r))
}
```

Compute and plot τ against $R_0 \in [0, 5]$ for $r = 0.3,\ 0.5,\ 0.7$.

```r
#################################################################################
# Plot the overall final size of R_0 in [0,5]
# in three cases: r = 0.3, 0.5, 0.7
#################################################################################
# Create a vector of 10000 R0 values between 0.00001 and 5
R0_vec <- seq(from = 0, to = 5, length.out = 10000)
df_tau_r <- data.frame(R0 = R0_vec)

# Create a vector of the corresponding values of tau for r = 0.3
tau_vec <- as.vector(10000)
r <- 0.3
for (i in 1:10000) {
  R0 <- R0_vec[i]
  tau_vec[i] <- tau_overall(R_0 = R0, r = r)
}
df_tau_r <- cbind(df_tau_r, overall_infected.3 = tau_vec)

# Create a vector of the corresponding values of tau for r = 0.5
tau_vec <- as.vector(10000)
r <- 0.5
```

```r
for (i in 1:10000) {
  R0 <- R0_vec[i]
  tau_vec[i] <- tau_overall(R_0 = R0, r = r)
}
df_tau_r <- cbind(df_tau_r, overall_infected.5 = tau_vec)

############################
# YOUR CODE HERE
############################
# As in the codes for r = 0.3 and 0.5, create a vector of the corresponding
# values of tau for r = 0.7, and consider it as the column in the data frame
# "df_tau_r" , with name "overall_infected.7".

# Create a vector of the corresponding values of tau for r = 0.7
tau_vec <- as.vector(10000)
r <- 0.7
for (i in 1:10000) {
  R0 <- R0_vec[i]
  tau_vec[i] <- tau_overall(R_0 = R0, r = r)
}
df_tau_r <- cbind(df_tau_r, overall_infected.7 = tau_vec)

# Create a plot of tau against R0 in three cases
ggplot(df_tau_r) +
  geom_line(aes(x = R0, y = overall_infected.3, color = "r = 0.3")) +
  geom_line(aes(x = R0, y = overall_infected.5, color = "r = 0.5")) +
  geom_line(aes(x = R0, y = overall_infected.7, color = "r = 0.7")) +
  scale_color_discrete(name = "immue fraction") +
  labs(title = " Assuming there is a fraction r of initially immune") +
  ylab("Overall fraction infected") +
  xlab(expression("R"[0])) +
  ylim(0, 1) + theme_minimal()
```



Exercise 1.2 (Determinstic and stochastic SIR Model)

(a) Consider a continuous-time deterministic SIR model with parameter values. Create a function to compute the derivative of the ODE system for SIR Model.

```
################################################################################
# Function to compute the derivative of the ODE system for SIR Model:
# S(t): number of susceptible
# I(t): number of infectives
# ODE:
# dS(t) / dt = -beta / N * S(t) * I(t)
# dI(t) / dt = beta / N * S(t) * I(t) - gamma * I(t)
#
#  t - time
#  y - current state vector of the ODE at time t
#  parms - Parameter vector used by the ODE system
#
# Returns:
#  list containing dS(t)/dt and dI(t)/dt
################################################################################
# Parameter values
gamma <- 0.25
beta <- 0.75
# We assume a closed population of size
times <- seq(0, 100, length = 1000)

deter_sir <- function(t, y, parms) {
  beta <- parms[1]
  gamma <- parms[2]
```

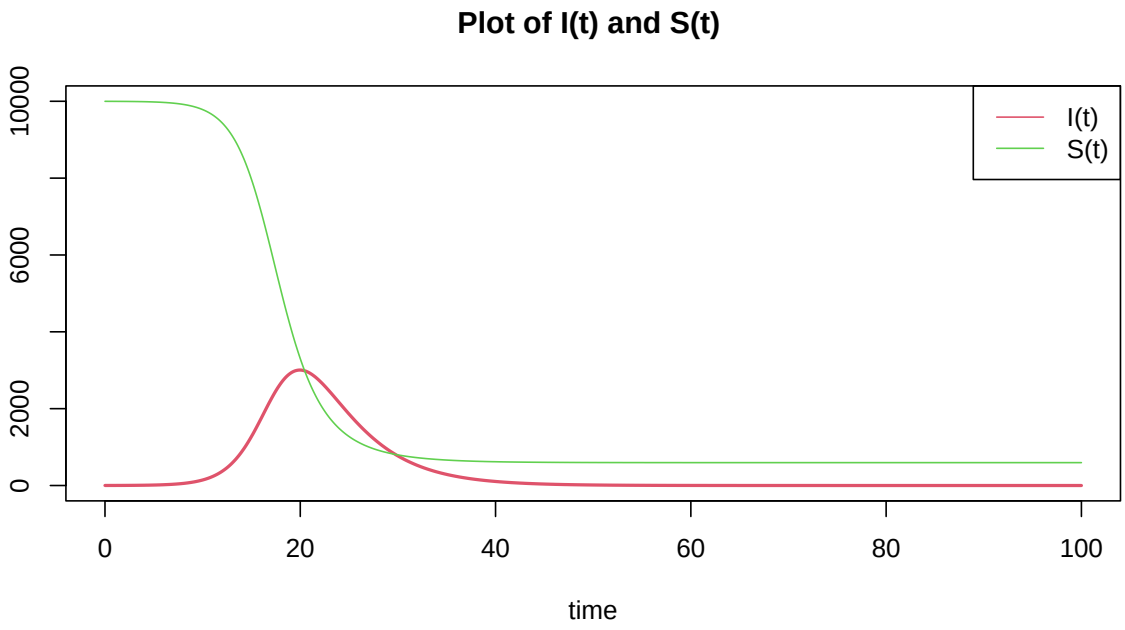
```
  N <- parms[3]
  S <- y[1]
  I <- y[2]
  return(list(c(
    S = -(beta * S * I) / N, # dS(t)/dt
    ############################
    # YOUR CODE HERE
    ############################
    # same for dI(t)/dt
    I = beta / N * S * I - gamma * I
  )))
}
```

Solve the SIR differential equation system with initial conditions: $S(0) = N - 1, I(0) = 1$:

```
##############################################
# YOUR CODE HERE, fill in the blank space
##############################################
# using lsoda(..) function

sim <- lsoda(
  y = c(10000 - 1, 1), # initial conditions for S ans I
  times = times, # times at which explicit estimates for y are desired
  func = deter_sir, # function computing values of derivatives in the ODE
  parms = c(beta, gamma, N = 10000) # vector/list of parameters used in function
)
```

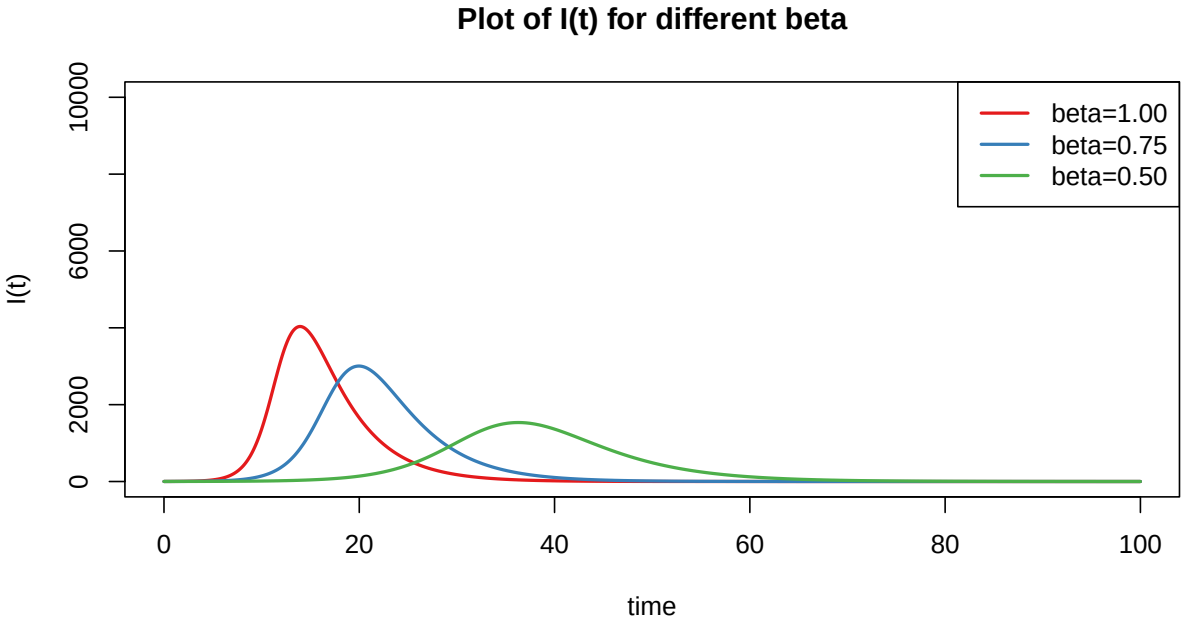
Plot the curves of $I(t)$ and $S(t)$ over time:

```
##############################################
# YOUR CODE HERE
##############################################

# simply use the function plot(...)
plot(times, sim[, 3], type = "l", lwd = 2,
     col = 2, xlab = "time", ylab = "",
     xlim = c(0, 100), ylim = c(0, 10000))
lines(times, sim[, 2], col = 3)
legend(x = "topright", c("I(t)", "S(t)"), col = c(2, 3), lty = 1)
title("Plot of I(t) and S(t) ")
```



(b) Now fix $\gamma = 0.25$, but choose different values of $\beta = 1, 0.75$ and 0.25. In each case, solve the SIR differential equation system with initial condition $(S(0) = N - 1, I(0) = 1)$. Plot the curves of $I(t)$ and compare them.

```
# set a vector of different values of beta
beta.grid <- c(1, 0.75, 0.5)

# summarize the I(t) for each case of beta
I <- sapply(
  beta.grid,
  function(beta) {
    ##############################################
    # YOUR CODE HERE, fill in the blank space
    ##############################################
    # again, using lsoda(..) function solve the ODE, compute the I(t)
    lsoda(y = c(10000 - 1, 1), times = times, func = deter_sir,
          parms = c(beta, gamma, N = 10000))[, 3]
  }
)

# plot the three curves of I(t)

pal <- brewer.pal(length(beta.grid), "Set1")
matplot(times, I, type = "l", lwd = 2, lty = 1, col = pal,
        xlab = "time", ylab = "I(t)", ylim = c(0, 10000))
legend(x = "topright", paste("beta=", sprintf("%.2f", beta.grid), sep = ""),
       lty = 1, lwd = 2, col = pal)
title("Plot of I(t) for different beta")
```



(c) Now the rate β depends on time when different measures take place. Create a function to compute the derivative of the ODE system for the SIR Model with beta(t).

```
sir_change <- function(t, y, parms) {
  beta0 <- parms[1]
  beta1 <- parms[2]
  beta2 <- parms[3]

  t1 <- parms[4]
  t2 <- parms[5]
  gamma <- parms[6]

  S <- y[1]
  I <- y[2]

  ################################################
  # YOUR CODE HERE, fill in the blank space
  ################################################
  # create a time-dependent rate \beta(t) as function of t

  beta <- function(t) {
    ifelse(t <= t1, beta0, ifelse(t > t2, beta2,
      beta1
    ))
  }

  return(list(c(
    S = -beta(t) / N * S * I, # list containing dS(t)/dt
    I = beta(t) / N * S * I - gamma * I
  ))) # and dI(t)/dt
```

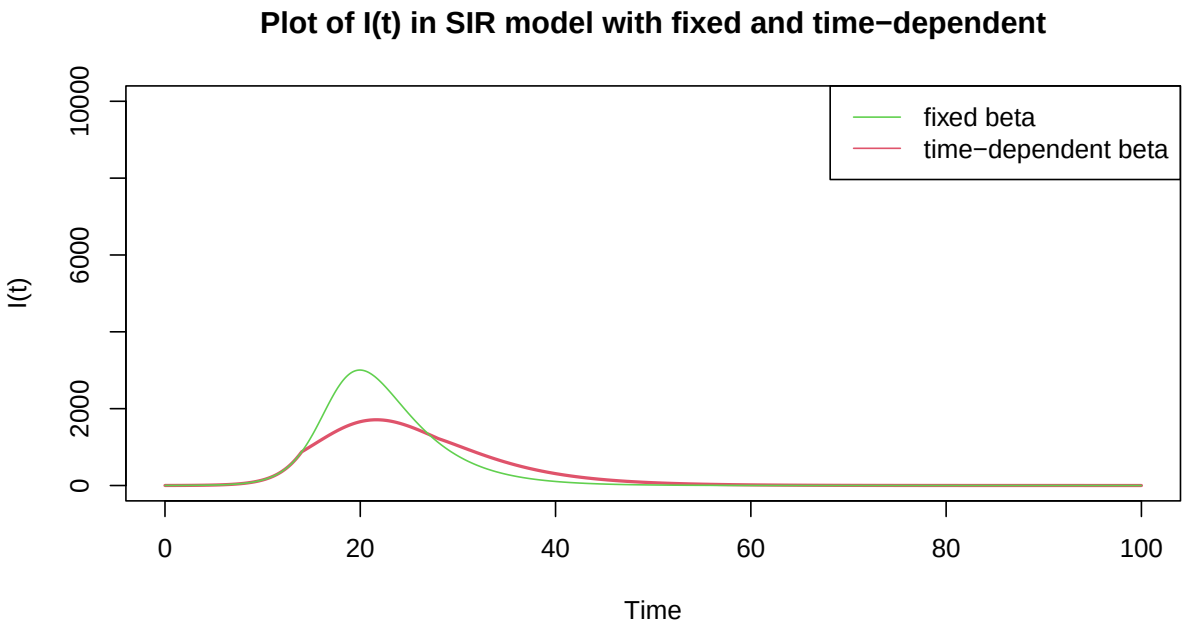
```
}

# set the paramter values
beta0 <- 0.75
beta1 <- 0.65 * 0.75
beta2 <- 0.75 * 0.75
t1 <- 14
t2 <- 28
gamma <- 0.25
N <- 10000
time <- seq(0, 100, length = 1000)
```

Solve the changed SIR differential equation system with initial conditions: $S(0) = N - 1, I(0) = 1$.

```
################################################
# YOUR CODE HERE, fill in the blank space
################################################

sol_SIRchange <- lsoda(
  y = c(N - 1, 1), # initial conditions for S ans I
  times = time, # times at which explicit estimates for y are desired
  func = sir_change, # function computing the values of derivatives in the ODE
  parms = c(beta0, beta1, beta2, t1, t2, gamma) # vector/list of parameters
)
```

Plot the curves of $I(t)$ for fixed beta and time-dependent beta.

```
################################################
# YOUR CODE HERE
################################################

# simply use the functions plot(...), lines(...)
plot(time, sol_SIRchange[, 3], type = "l", lwd = 2, col = 2,
     xlab = "Time", ylab = "I(t)", xlim = c(0, 100), ylim = c(0, N))
lines(times, sim[, 3], col = 3)
legend(x = "topright", c("fixed beta",
                         "time-dependent beta"), col = c(3, 2), lty = 1)
title("Plot of I(t) in SIR model with fixed and time-dependent")
```



(d) Here we turn our focus to stochastic SIR model in continuous time. Always assume that there are fraction $c = 10\%$ of initial infectives.

Simulate simple SIR model using the Gillespie-Algorithm.

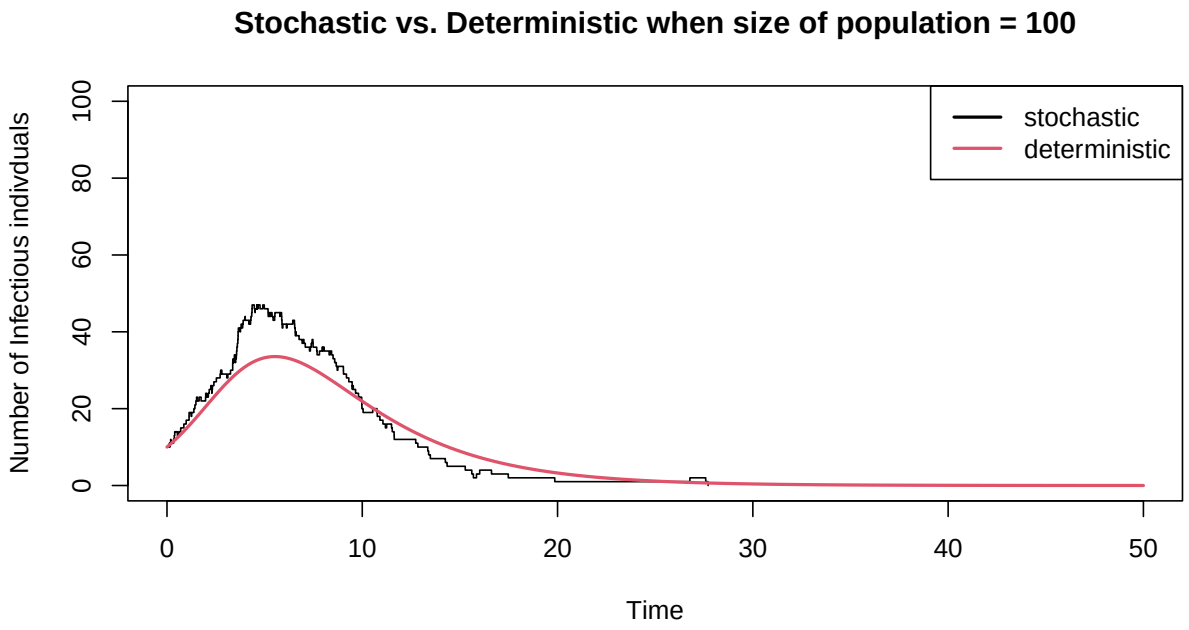
```
#########################################################################
# Params:
#  [0,T] - time horizon
#  beta - infection rate
#  gamma - recovery rate
#  n - initial number of susceptibles.
#  m - initial number of infectives
#########################################################################

stoch_SIR <- function(T, beta, gamma, n, m, N) {
  # Initialize (x= number of S, y=number of I, t=event time)
  x <- n
  y <- m
  z <- 1
  t <- 0
  N <- n + m
  # Possible events: Infection(S->I) and Removal(I->R)
  eventLevels <- c("S->I", "I->R")
  # Initialize result
  df_SIR <- data.frame(t = t, x = x, y = y, totalinfected = z, event = NA)
  # Loop until  time T or the epidemic stops(there is no infectives)
  while (t < T & (y > 0)) {
    ##############################################
    # YOUR CODE HERE, fill in the blank space
    ##############################################
```

```
    # Draw the waiting type for each possible event

    wait <- rexp(2, c(
      "S->I" = beta / N * x * y, # here insert the rate for "S->I"
      "I->R" = gamma * y # the rate for "I->R"
    ))
    "
    From those two rates, we draw two exponential
    random numbers for each possible event
    "

    # Determine which event occurs first
    # i.e. the event with the smaller or bigger random number?
    # which.min(wait) or which.max(wait)?

    i <- which.min(wait)

    # Record Event Time
    t <- t + wait[i]

    # Update the number of S and I according to the event type
    # if an infection occurs,
    if (eventLevels[i] == "S->I") {
      x <- x - 1  # the number of S will decrease by 1.
      # what happen to y, z?
      y <- y + 1

      z <- z + 1
    }
    # if an recovery occurs,
    if (eventLevels[i] == "I->R") {

      # what happen to x, y, z?
      y <- y - 1
    }
    # Store result
    df_SIR <- rbind(df_SIR, c(t, x, y, z, i))
  }
  # Re-code event type and return
  df_SIR$event <- factor(eventLevels[df_SIR$event], levels = eventLevels)

  return(df_SIR)
}
```

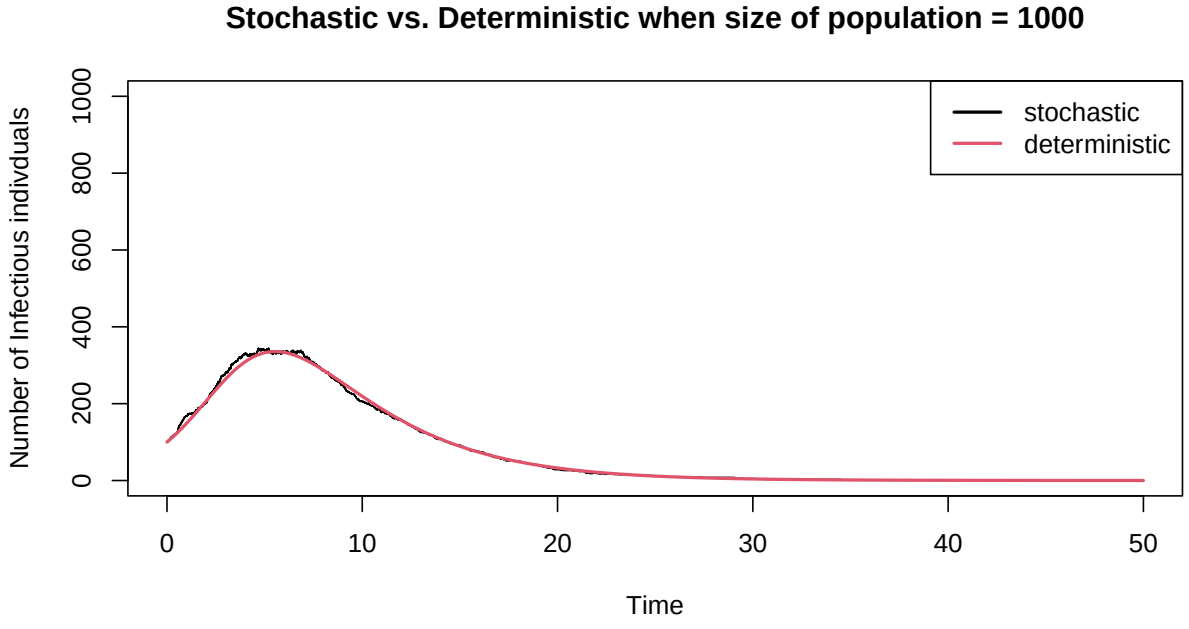
Plot $I(t)$ for one simulated stochastic epidemic and deterministic limit for different size of population $N = 100, 1000$ and 10000.

```
##############################################
# YOUR CODE HERE, fill in the blank space
##############################################

# set parameter values:
beta <- 0.75
```

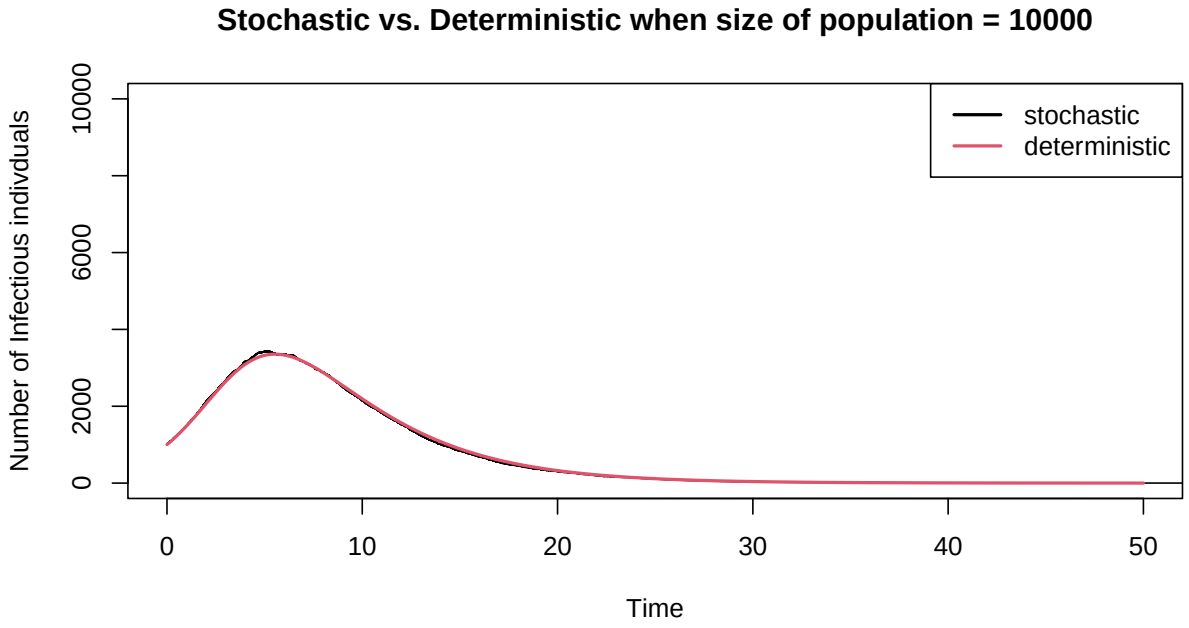
```r
gamma <- 0.25
Tmax <- 50 # time limit
c <- 0.1 # initial infectious fraction

compare_DS <- function(N) {

  # do one simulated stochastic epidemic
  traj <- stoch_SIR(T = Tmax, beta = beta, gamma = gamma,
                    n = N * (1 - c), m = N * c, N = N)

  # plot the stochastic curve of I
  plot(traj$t, traj$y,
    type = "s", ylim = c(0, N), xlab = "Time",
    ylab = "Number of Infectious indivduals", xlim = c(0, Tmax), col = 1
  )

  # Solve the deterministic ODE

  solution <- lsoda(
    y = c(N * (1 - c), N * c),
    times = seq(0, Tmax, length = 1000),
    func = deter_sir,
    parms = c(beta, gamma, N = N)
  )

  # add the deterministic curve of I

  lines(solution[, "time"], solution[, 3], col = 2, lwd = 2)
}


## Compare deterministic and stochastic for different N
set.seed(123)
compare_DS(N = 100)
legend(x = "topright", c("stochastic", "deterministic"),
       col = c(1, 2), lty = 1, lwd = 2)
title("Stochastic vs. Deterministic when size of population = 100")
```




```
#########################################################################
# YOUR CODE HERE,do the same for the other two cases  N=1000, 10000
#########################################################################
set.seed(123)
compare_DS(N = 1000)
legend(x = "topright", c("stochastic", "deterministic"),
       col = c(1, 2), lty = 1, lwd = 2)
title("Stochastic vs. Deterministic when size of population = 1000")
```



```r
set.seed(123)
compare_DS(N = 10000)
legend(x = "topright", c("stochastic", "deterministic"),
       col = c(1, 2), lty = 1, lwd = 2)
title("Stochastic vs. Deterministic when size of population = 10000")
```



(e) Let $\beta = 0.375$ and $\gamma = 0.25$.(implying that $R_0 = 1.5$), do 5000 simulations in three cases when the size of population $N = 500, 1000$ and 5000 with one initial infective.

Stochastic SIR model create a function final size.

```r
# total infected:
fsize_SIR <- function(T, beta, gamma, n, m, N) {
  # Initialize (x= number of S, y=number of I, z= y=number of R, t=event time)
  x <- n
  y <- m
  z <- 0
  t <- 0
  # size of population
  N <- n + m

  # Possible events: Infection(S->I) and Removal(I->R)
  eventLevels <- c("S->I", "I->R")

  # Loop until  time T or the epidemic stops(there is no infectives)
  while (t < T & (y > 0)) {
    ##################################################
    # YOUR CODE HERE,follow the codes in part d
    ##################################################
```

```
    # Draw the waiting type for each possible event
    wait <- rexp(2, c("S->I" = beta / N * x * y, "I->R" = gamma * y))
    # Determine which event occurs first
    i <- which.min(wait)
    # Record Event Time
    t <- t + wait[i]
    # Update the number of S and I according to the event type
    if (eventLevels[i] == "S->I") {
      x <- x - 1
      y <- y + 1
      z <- z + 1
    } # if infection
    if (eventLevels[i] == "I->R") {
      y <- y - 1
    } # if recovery
  }
  return(z) # return the number of recovered
}
```

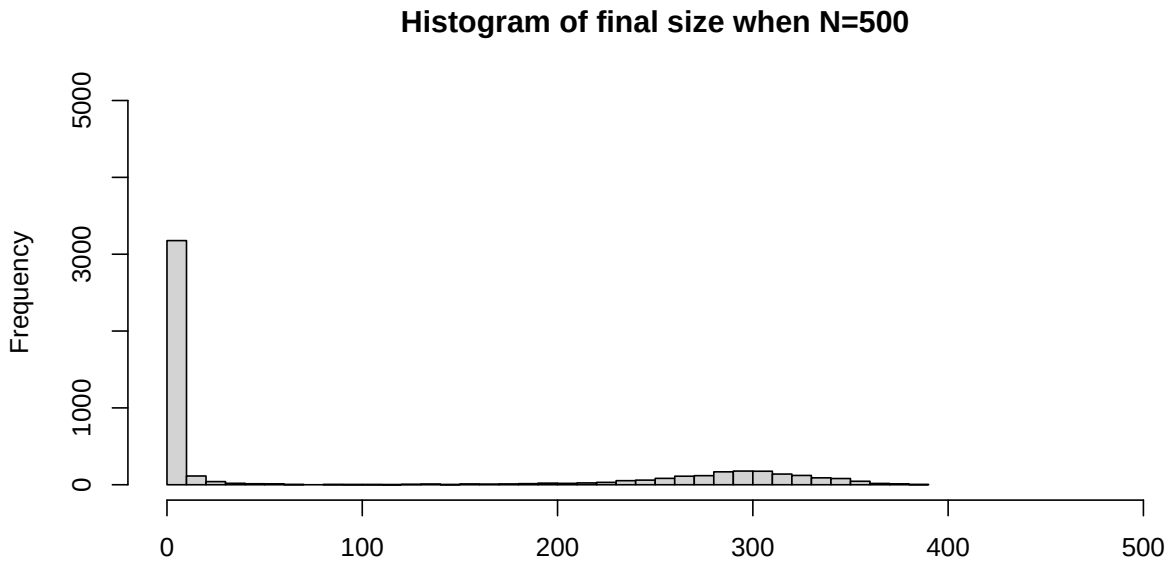
Do 5000 simulations with different size of community N.

```
################################################
# YOUR CODE, fill in the blank space
################################################
# set parameter values:
beta <- 0.375
gamma <- 0.25
Tmax <- 500 # time limit [0,500]
I0 <- 1
nSim <- 5000 # number of simulations

compare_hist <- function(N) {
  df_fsize <- data.frame(sim = NA, totalinfected = NA)

  for (i in 1:nSim) {
    z <- fsize_SIR(T = Tmax, beta = beta, gamma = gamma,
                   n = N - I0, m = I0, N = N)

    df_fsize <- rbind(df_fsize, c(i, z))
  }
  return(df_fsize$totalinfected)
}
```

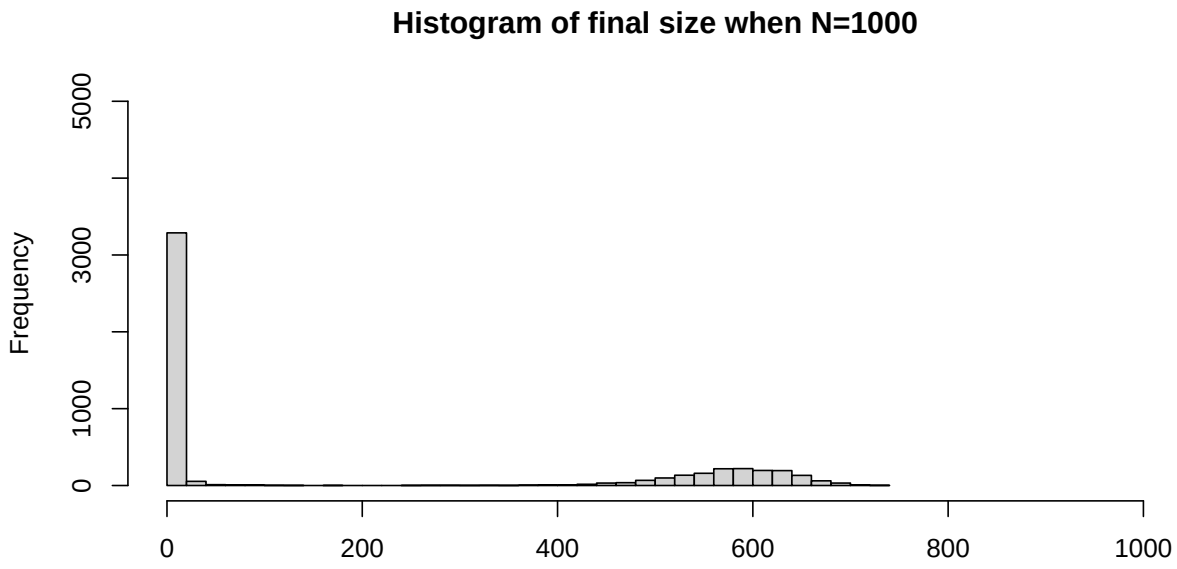
Make a histogram of the final size distribution in each case. Give your comments.

```
N <- 500

hist(compare_hist(N = 500),
  breaks = 50,
  main = "Histogram of final size when N=500",
  xlab = "",
  xlim = c(0, N),
```

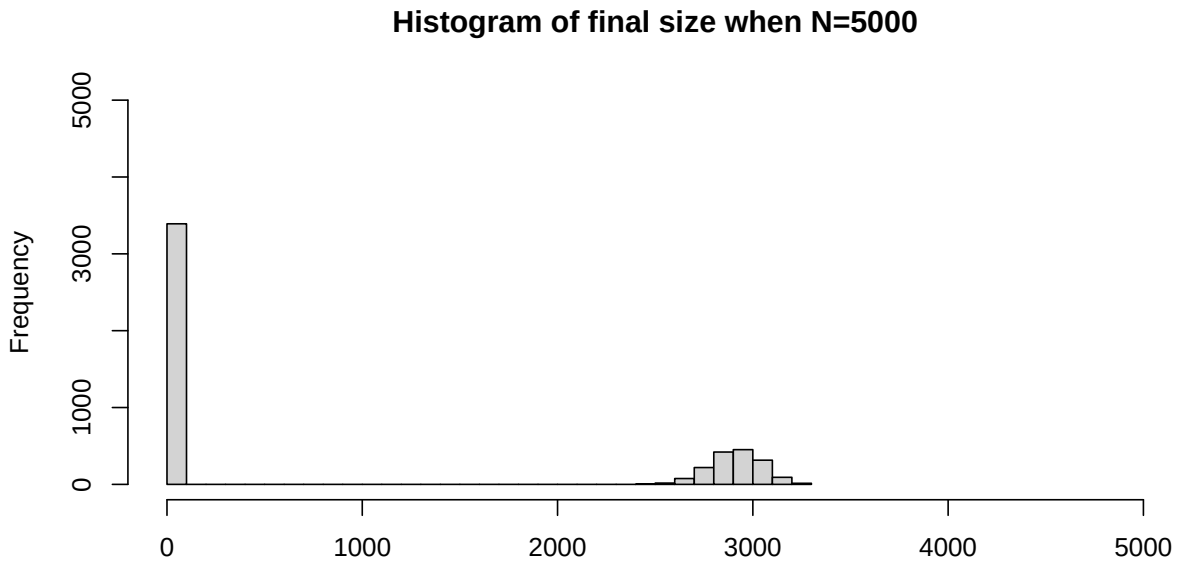
```
  ylim = c(0, nSim)
)
```




```
################################################
# YOUR CODE
################################################
# do the other two cases: N = 1000 and 5000
hist(compare_hist(N = 1000),
  breaks = 50, main = "Histogram of final size when N=1000",
  xlab = "", xlim = c(0, 1000), ylim = c(0, 5000)
)
```




```
hist(compare_hist(N = 5000),
  breaks = 45, main = "Histogram of final size when N=5000",
  xlab = "", xlim = c(0, 5000), ylim = c(0, 5000)
)
```

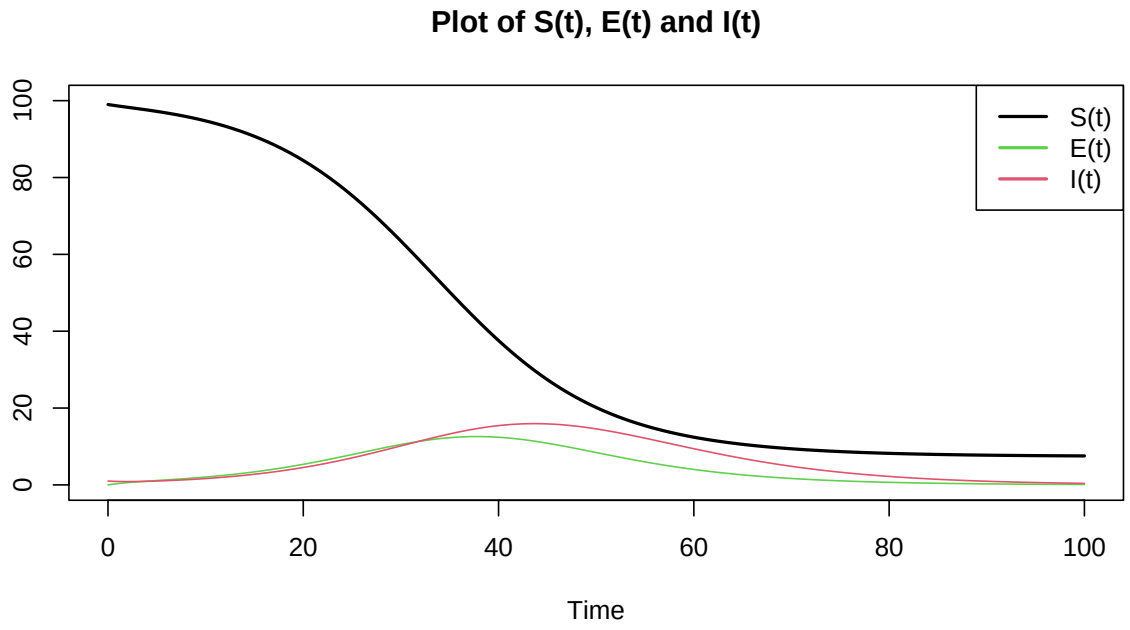


Exercise 1.3 (SEIR Model)

(a)

```
##############################################
# YOUR CODE HERE, fill in the blank space
##############################################
# do the same way as for SIR model, but with four states: S E I R

#############################################################################
# Function to compute the derivative of the ODE system for SEIR Model
#
#  t - time
#  y - current state vector of the ODE at time t
#  parms - Parameter vector used by the ODE system
#
# Returns:
#  list containing dS(t)/dt, dE(t)/dt, and dI(t)/dt
#############################################################################

seir <- function(t, y, parms, N) {
  beta <- parms[1]
  rho <- parms[2]
  gamma <- parms[3]
  N <- parms[4]

  S <- y[1]
  E <- y[2]
  I <- y[3]
  return(list(c(S = -beta / N * S * I, E = beta / N * S * I - rho * E,
                I = rho * E - gamma * I)))
}


# Parameter Values:
times <- seq(0, 100, length = 1000)
gamma <- 1 / 7
beta <- 0.4
rho <- 1 / 5
I0 <- 1
N <- 100

# Solve the ODE and plot
sol_seir <- lsoda(
  y = c(N - I0, 0, I0), # insert the corresponding initial conditions
  times = times,
  func = seir,
  parms = c(beta, rho, gamma, N)
)

plot(times, sol_seir[, 2],
  type = "l", lwd = 2, col = 1, xlab = "Time",
  ylab = "", xlim = c(0, 100), ylim = c(0, N)
)
```

```
lines(times, sol_seir[, 3], col = 3)
lines(times, sol_seir[, 4], col = 2)
legend(x = "topright", c("S(t)", "E(t)", "I(t)"),
       col = c(1, 3, 2), lty = 1, lwd = 2)
title("Plot of S(t), E(t) and I(t)")
```



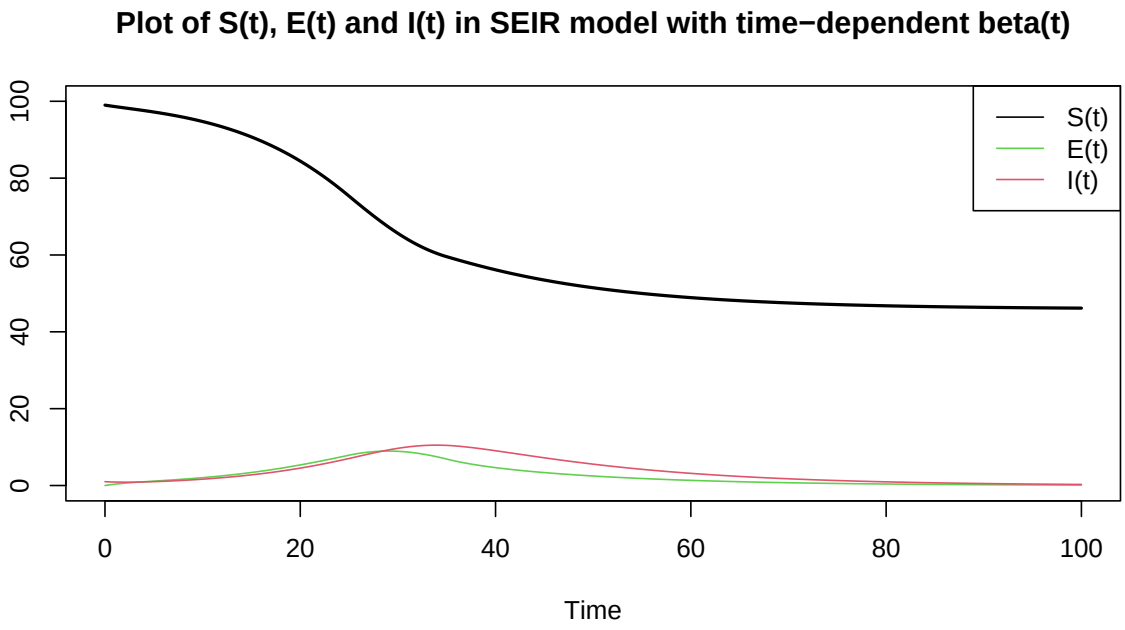
(b) Modify the SEIR model such that $\beta(t)$ becomes a time-dependent function. Create a function to compute the derivative of the ODE system for SEIR Model with changing point.

```
##############################################
# YOUR CODE HERE, fill in the blank space
##############################################

#############################################################################################
#
#  t - time
#  y - current state vector of the ODE at time t
#  parms - Parameter vector used by the ODE system
#
# Returns:
#  list containing dS(t)/dt, dE(t)/dt, and dI(t)/dt
#############################################################################################
seir_change <- function(t, y, parms, N) {
  beta0 <- parms[1]
  beta1 <- parms[2]
  t1 <- parms[3]
  w <- parms[4]
  rho <- 1 / 5
  gamma <- parms[5]
  N <- parms[6]
```

```
  S <- y[1]
  E <- y[2]
  I <- y[3]

  # write the time-dependent rate \beta(t) using ifelse(,)

  beta <- function(t) {
    ifelse(t <= t1 - w, beta0, ifelse(t > t1 + w, beta1,
      beta0 + (beta1 - beta0) / (2 * w) * (t - (t1 - w))
    ))
  }
  # write the return list containing dS(t)/dt, dE(t)/dt, and dI(t)/dt
  return(list(c(S = -beta(t) / N * S * I, E = beta(t) / N * S * I - rho * E,
                I = rho * E - gamma * I)))
}
# plot \beta(t):
# plot(time, b, type="l",xlab="Time",ylab="beta(t)")
# title("Plot of beta(t)")
```

Take $t_1 = 30, w = 5, \beta_0 = 0.4$ and $\beta_1 = 0.12$, solve the ODE system for the SEIR model with time-varying transmission rate in part (b) numerically using R command `lsoda` and plot $S(t), E(t)$ and $I(t)$ for $t \in [0, 100]$.

```
##############################################
# YOUR CODE HERE, fill in the blank space
##############################################
# set the parameter values: such as
beta0 <- 0.4
beta1 <- 0.12
t1 <- 30
w <- 5
gamma <- 1 / 7
N <- 100
I0 <- 1
time <- seq(0, 100, length = 1000)

# solve the ODE system for the changed SEIR model

sol_SEIRchange <- lsoda(
  y = c(N - I0, 0, I0),
  times = time, func = seir_change,
  parms = c(beta0, beta1, t1, w, gamma, N)
)

# plot curves of $S(t), E(t)$ and $I(t)$
plot(times, sol_SEIRchange[, 2],
  type = "l", lwd = 2, col = 1, xlab = "Time",
  ylab = "", xlim = c(0, 100), ylim = c(0, 100)
)
lines(times, sol_SEIRchange[, 3], col = 3)
lines(times, sol_SEIRchange[, 4], col = 2)
legend(x = "topright", c("S(t)", "E(t)", "I(t)"), col = c(1, 3, 2), lty = 1)
title("Plot of S(t), E(t) and I(t) in SEIR model with time-dependent beta(t)")
```



(d) Now for $N = 100, 1000$ and 10000, do one simulation of the stochastic SEIR epidemic starting from fraction infected 10% with exponentially distributed incubation period with mean 5 days and the above time-changing $\beta(t)$. Overlay it on the plot of the deterministic curve as done in Exercise 2.

Simulate a stochastic SEIR epidemic model.

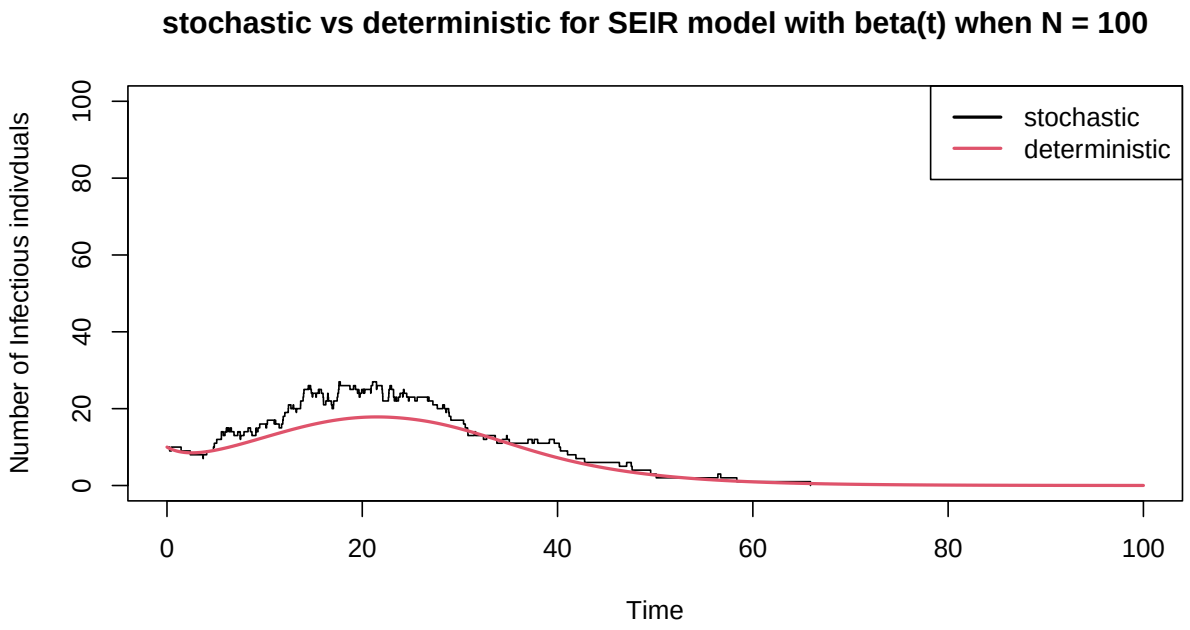
```
################################################
# YOUR CODE HERE, fill in the blank space
################################################

#########################################################################
#
# Params:
#  [0,T] - time horizon
#  beta - infection rate
#  gamma - recovery rate
#  rho - rate of (E -> I)
#  n - initial number of susceptibles.
#  m - initial number of infectives, here m=1
#  three events: "S->E","E->I","I->R"
#########################################################################


stoch_SEIR <- function(T, n, m, parms, N) {
  beta0 <- parms[1]
  beta1 <- parms[2]
  t1 <- parms[3]
  w <- parms[4]
  gamma <- parms[5]
  rho <- parms[6]
```

```r
  N <- parms[7]

  # write the function for beta(t)
  beta <- function(t) {
    ifelse(t <= t1 - w, beta0, ifelse(t > t1 + w, beta1,
      beta0 + (beta1 - beta0) / (2 * w) * (t - (t1 - w))
    ))
  }
  # Initialize (x= number of S, y=number of I, z=number of E, t=event time)
  x <- n
  y <- m
  z <- 0
  time <- 0
  # Possible events:
  eventLevels <- c("S->E", "E->I", "I->R")
  # Initialize result
  df_SEIR <- data.frame(time = time, x = x, y = y, z = z, event = NA)
  # Loop until  time T or the epidemic stops(there is no infectives)
  while (time < T & (y > 0)) {
    # Draw the waiting type for each possible event
    b <- beta(time)
    wait <- rexp(3, c("S->E" = b / N * x * y,
                      "E->I" = rho * z, "I->R" = gamma * y))
    # Determine which event occurs first
    i <- which.min(wait)
    # Record Event Time
    time <- time + wait[i]
    # Update the number of S, I, E according to the event type
    if (eventLevels[i] == "S->E") {
      x <- x - 1
      z <- z + 1
    }
    if (eventLevels[i] == "E->I") {
      z <- z - 1
      y <- y + 1
    }
    if (eventLevels[i] == "I->R") {
      y <- y - 1
    }
    # Store result
    df_SEIR <- rbind(df_SEIR, c(time, x, y, z, i))
  }
  # Re-code event type and return
  df_SEIR$event <- factor(eventLevels[df_SEIR$event], levels = eventLevels)

  return(df_SEIR)
}
```

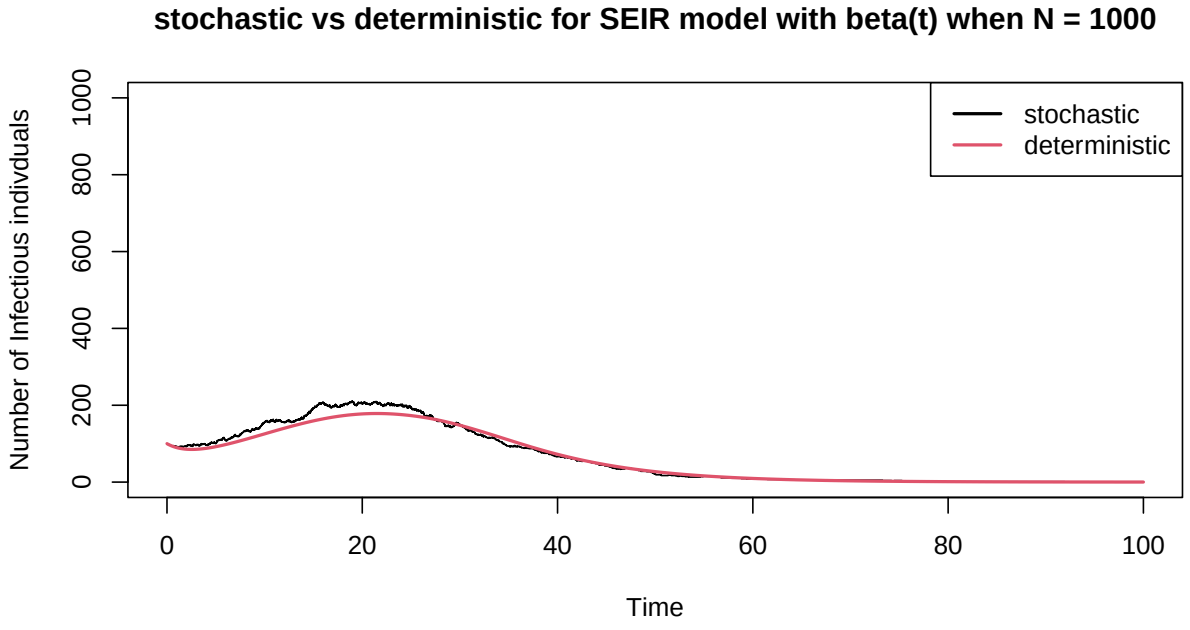
```r
##############################################
# YOUR CODE HERE, fill in the blank space
##############################################

# set paramter values:
```

```r
beta0 <- 0.4
beta1 <- 0.12
t1 <- 30
w <- 5
gamma <- 1 / 7
rho <- 1 / 5
c <- 0.1

compare_ds_seir <- function(N) {
  beta <- function(t) {
    ifelse(t <= t1 - w, beta0, ifelse(t > t1 + w, beta1,
      beta0 + (beta1 - beta0) / (2 * w) * (t - (t1 - w))
    ))
  }
  # Do  stochastic simulations
  set.seed(123)
  traj <- stoch_SEIR(T = 100, n = N * (1 - c), m = N * c,
                     parms = c(beta0, beta1, t1, w, gamma, rho, N))

  # plot stochastic I(t)

  plot(traj$time, traj$y,
    type = "s", ylim = c(0, N), xlab = "Time",
    ylab = "Number of Infectious indivduals", xlim = c(0, 100), col = 1
  )

  # Solve the deterministic ODE

  solution <- lsoda(
    y = c(N * (1 - c), 0, N * c), times = seq(0, 100, length = 1000),
    func = seir_change, parms = c(beta0, beta1, t1, w, gamma, N)
  )
  lines(solution[, "time"], solution[, 4], col = 2, lwd = 2)
  invisible()
}
```

```r
# plot stochastic and deterministic for different N
compare_ds_seir(N = 100)
title("stochastic vs deterministic for SEIR model with beta(t) when N = 100")
legend(x = "topright", c("stochastic", "deterministic"),
       col = c(1, 2), lty = 1, lwd = 2)
```




```
# do the same for other two cases
compare_ds_seir(N = 1000)
title("stochastic vs deterministic for SEIR model with beta(t) when N = 1000")
legend(x = "topright", c("stochastic", "deterministic"),
       col = c(1, 2), lty = 1, lwd = 2)
```



```
compare_ds_seir(N = 10000)
title("stochastic vs deterministic for SEIR model with beta(t) when N = 10000")
legend(x = "topright", c("stochastic", "deterministic"),
       col = c(1, 2), lty = 1, lwd = 2)
```