Exercise sheet 1
Prepare the below such that you are able to discuss it on Wednesday 3rd March

Exercise 1 Euler method

Euler's method to numerically solve a first order ODE of the form

$$\frac{\mathrm{d}\mathbf{y}(t)}{\mathrm{d}t} = \mathbf{f}(t, \mathbf{y}(t))$$

given the initial condition $\mathbf{y}(t_0) = \mathbf{y}_0$ works by iteratively advancing from t_n to $t_{n+1} = t_n + h$ by

$$\mathbf{y}_{t_{n+1}} = \mathbf{y}_{t_n} + h \cdot \mathbf{f}(t_n, \mathbf{y}_{t_n}),$$

where $h > 0$ is the step-length of the method.

a) Implement the SIR model in R with a function which takes inputs: a vector $(s(t), i(t))'$ and parameter vector $\theta = (\beta, \gamma)'$. The function should compute and return the derivative $(\mathrm{d}s(t)/\,\mathrm{d}t, \mathrm{d}i(t)/\,\mathrm{d}t)'$.

```
sir_f <- function(sit, theta)
```

Solution:

```r
sir_f <- function(sit, theta) {
  # Grab the relevant parts of the input vectors
  beta <- theta[1]
  gamma <- theta[2]
  S <- sit[1]
  I <- sit[2]
  # Return two-dimensional vector with derivatives
  return(c(- beta * S * I, beta * S * I - gamma * I))
}
```

b) Implement Euler's method to numerically solve the SIR ODE system as an R function. The function should take as inputs: a vector containing a grid of time values with the first element corresponding to $t_0 = 0$ and fixed distance h between two consecutive elements and a vector of length two containing $(s(t_0), i(t_0))'$, as well as the parameter vector $\theta = (\beta, \gamma)'$.

```
euler_f <- function(grid, si0, theta)
```

Your function should return a matrix with the grid of time values, and the solution $s(t)$ and $i(t)$ for each t.

Solution:

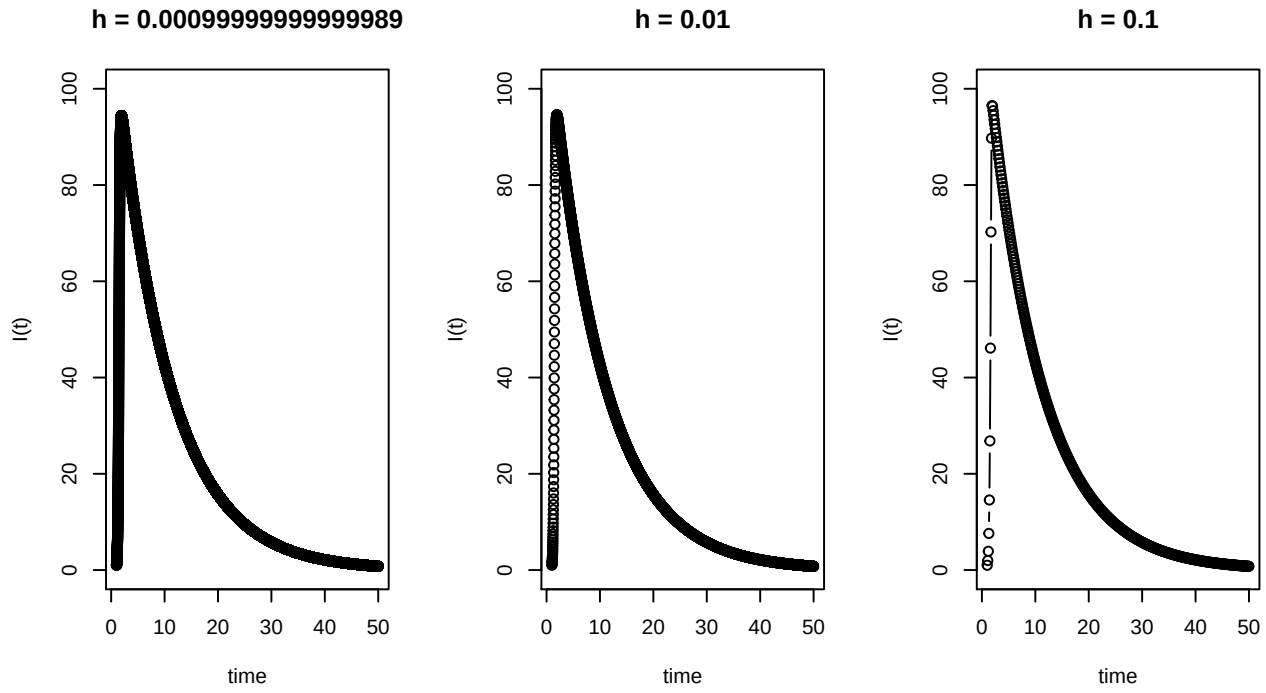
```r
euler_f <- function(grid, si0, theta) {
  # Empty matrix to be populated
  si <- matrix(NA, ncol = length(si0), nrow = length(grid))
  colnames(si) <- c("S(t)", "I(t)")
  # First row is initial values (S(t_0), I(t_0)) as given
  si[1, ] <- si0
```

```
  # (Lagged) difference between two gridpoints
  h <- diff(grid)[1]
  # Loop by advancing by the derivative
  for (i in 1 : (length(grid) - 1)) {
    # NB length subtracted one since first entry given as above
    si[i + 1, ] <- si[i, ] + h * sir_f(si[i, ], theta)
    # Using the function from the previous question to give us
    # the derivative at the time step
  }
  return(cbind("time" = grid, si))
}
```

c) Use Euler's method to solve the SIR ODE with $(s(0), i(0))' = (99, 1)'$ and $\theta = (0.1, 0.1)$ on $[0, 50]$. Try steps of different sizes; $h \in \{1, 0.1, 0.01\}$. Which effect does h appear to have? (*Hint*: A graphical representation may help visualise what is going on)

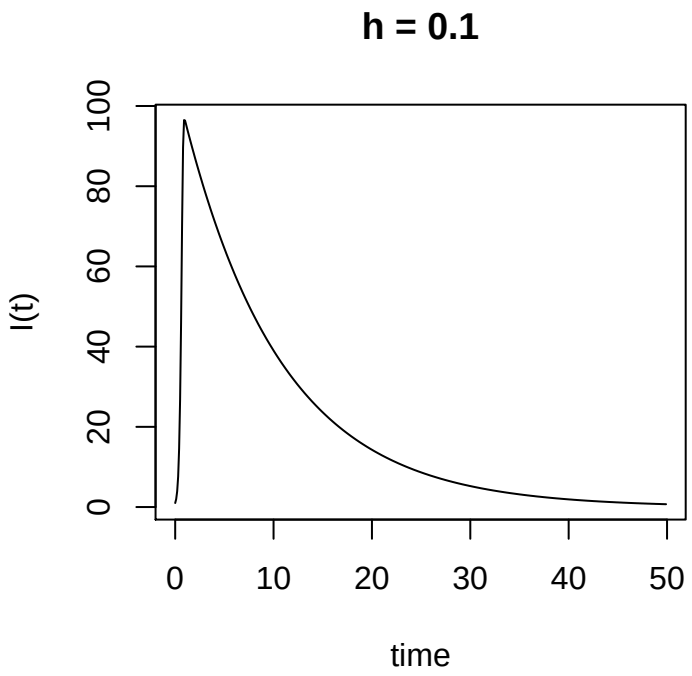
Solution:

```
par(mfrow = c(1, 3))
grid <- seq(1, 50, by = 0.001)
res <- euler_f(grid, si0 = c(99, 1), theta = c(beta = 0.1, gamma = 0.1))
plot(grid, res[, "I(t)"], type = "b", xlab = "time", ylab = "I(t)",
     ylim = c(0, 100), main = paste("h =", unique(diff(grid))[1]))
grid <- seq(1, 50, by = 0.01)
res <- euler_f(grid, si0 = c(99, 1), theta = c(beta = 0.1, gamma = 0.1))
plot(grid, res[, "I(t)"], type = "b", xlab = "time", ylab = "I(t)",
     ylim = c(0, 100), main = paste("h =", unique(diff(grid))[1]))
grid <- seq(1, 50, by = 0.1)
res <- euler_f(grid, si0 = c(99, 1), theta = c(beta = 0.1, gamma = 0.1))
plot(grid, res[, "I(t)"], type = "b", xlab = "time", ylab = "I(t)",
     ylim = c(0, 100), main = paste("h =", unique(diff(grid))[1]))
```



We see that the step-length h determines how far apart the points are. Alternatively, the inner workings of the `loop` in `euler_f` could have been implemented using (here illustrated for step-length $h = 0.1$)

```
# Step width of the Euler method
h <- 0.1
beta <- gamma <- 0.1
y <- matrix(NA, nrow = ceiling(50 / h), # Time frame is [0, 50]
            ncol = 3,
            dimnames = list(NULL, c("t","S","I")))
# Initial value
y[1, ] <- c(0, 99, 1)
# Loop
for (i in 2 : nrow(y)){ # Looping from time t = 1
  y[i, ] <- c(y[i - 1, "t"] + h, y[i - 1, c("S", "I")] +
                h * sir_f(sit = y[i - 1, c("S", "I")],
                          theta = c(beta, gamma)))
}
# Show Euler solve
plot(y[, "t"], y[, "I"], type = "l", xlab = "time", ylab = "I(t)",
     main = paste("h =", h))
```



d) Load the R package **deSolve** and read the documentation of the `lsoda` function.
Solution:

```
#install.packages("deSolve")
library(deSolve)
?lsoda
```

Note that `lsoda` says the input `func` requires a list: "The return value of `func` should be a list, whose first element is a vector containing the derivatives of `y` with respect to time, and whose

next elements are global values that are required at each point in `times`. The derivatives must be specified in the same order as the state variables `y`."

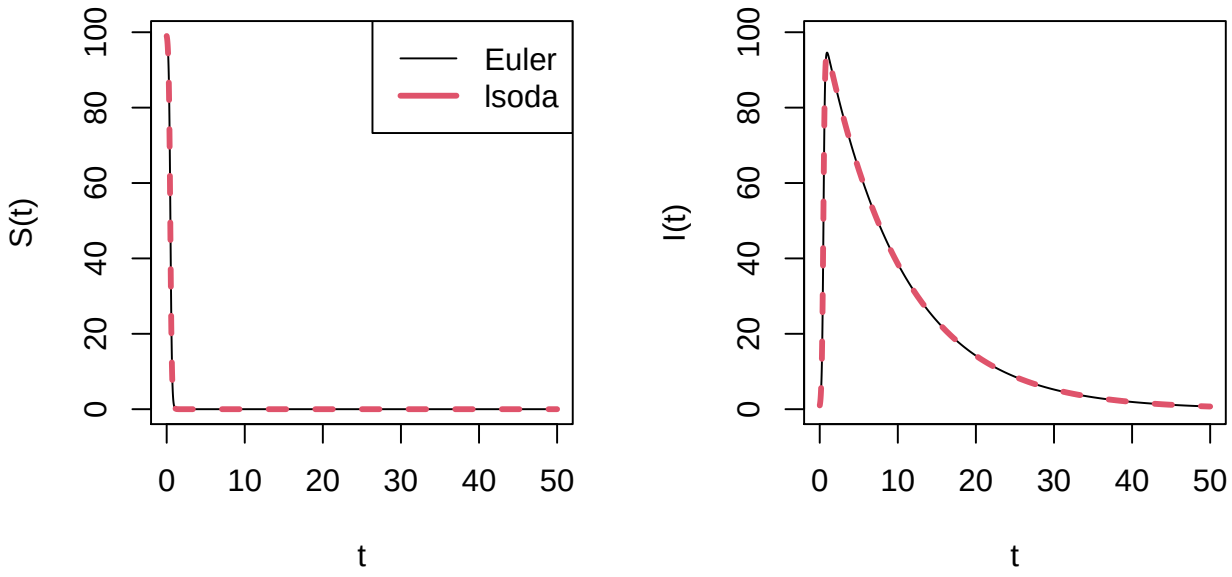
e) Write R code which applies `lsoda` to numerically solve the simple SIR model and compare the solution with your Euler method (input parameters are: $(s(0), i(0))' = (99, 1)'$ and $\theta = (0.1, 0.1)$)

`sir_l <- function(t, y, parms)`

(*Hint*: A graphical representation may help visualise what is going on)
Solution: Note the similarity of `sir_l` with the `sir_f` function but also note the additional argument `t`. Recall from the previous question that this is required for the `func` input of the `lsoda` ODE solver.
(*Hint*: A graphical representation may help visualise the results)

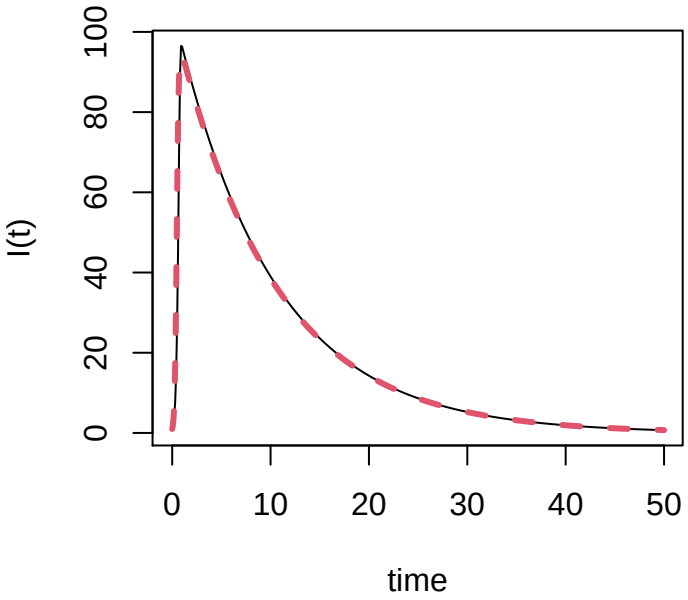
```
# SIR model implemented as required for lsoda
sir_l <- function(t, y, parms) {
  beta <- parms[1]
  gamma <- parms[2]
  S <- y[1]
  I <- y[2]
  # Returns a list rather than a vector -- difference compared to sir_f
  return(list(c(S = - beta * S * I, I = beta * S * I - gamma * I)))
}
# Function inputs
beta <- 0.1
gamma <- 0.1
grid <- seq(0, 50, by = 0.01)
# Our Euler function
res <- euler_f(grid, si0 = c(99, 1), theta = c(beta, gamma))
# Lsoda
res2 <- lsoda(y = c(99, 1), times = grid, func = sir_l,
              parms = c(beta, gamma))

# Plots of the two
par(mfrow = c(1, 2))
matplot(grid, cbind(res[, "S(t)"], res2[, 2]), type = "l",
        ylim = c(0, 99),
        lwd = c(1, 3), xlab = "t",
        ylab = "S(t)")
legend("topright", c("Euler", "lsoda"), col = 1 : 2, lwd = c(1, 3))
matplot(grid, cbind(res[, "I(t)"], res2[, 3]), type = "l",
        ylim = c(0, 99),
        lwd = c(1, 3), xlab = "t",
        ylab = "I(t)")
```



We see that our Euler implementation and the numerical `lsoda` approach yield similar results. We also see that the starting values of $(s(0), i(0))' = (99, 1)'$ with $\theta = (0.1, 0.1)$ leads to a set of epidemic curves with a rapid decrease in S; similarly a rapid increase in I. Alternatively, with `plot` and `lines` instead of `matplot`

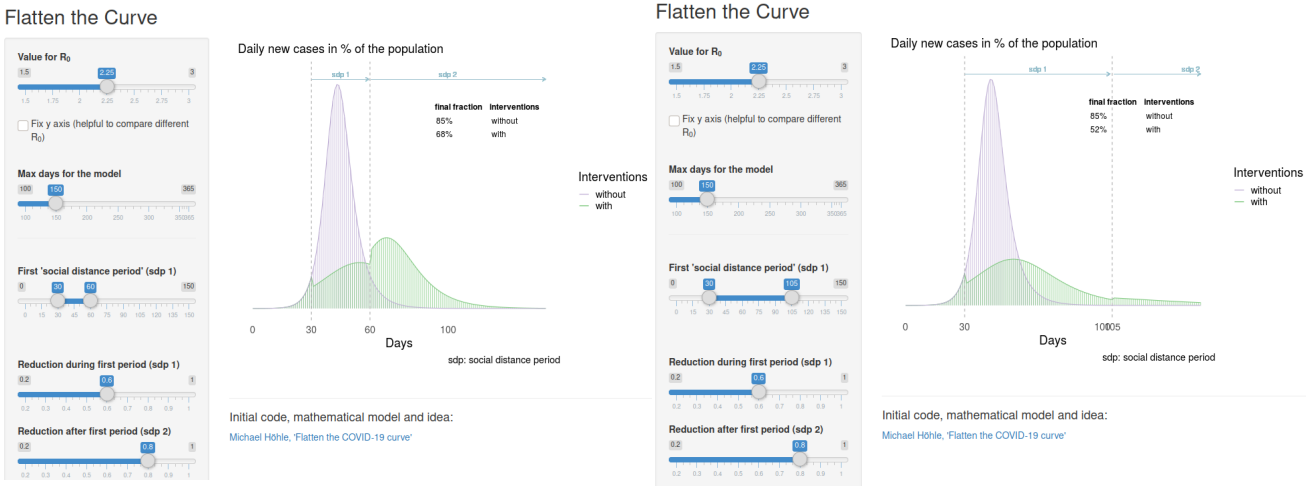
```
plot(y[, "t"], y[, "I"], type = "l", xlab = "time", ylab = "I(t)")
# Add lsoda (which uses a more advanced method)
lines(grid, lsoda(y = c(99, 1),
                  times = grid,
                  func = sir_l, parms = c(beta, gamma))[, 3],
      col = 2, lwd = 3, lty = 2)
```



Exercise 2 COVID-19

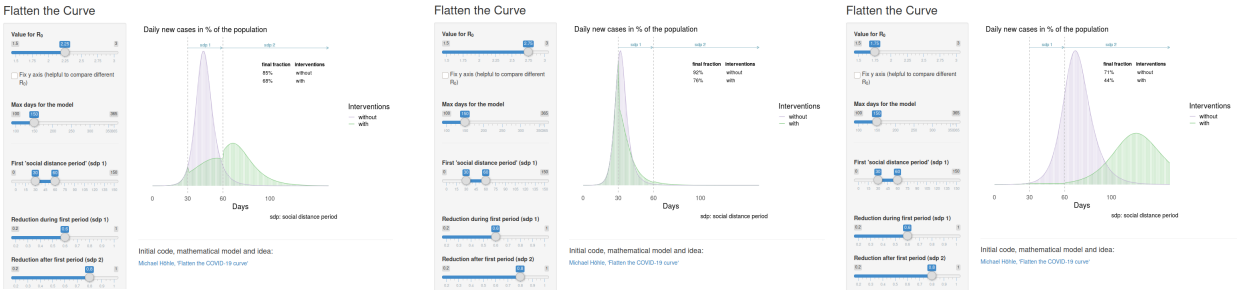
Read the blog post "Flatten the COVID-19 curve" and experiment with different containment strategies using the Shiny App located at `https://tinu.shinyapps.io/Flatten_the_Curve/`. Discuss the pros and cons of different strategies. Discuss limitations of the model when used to evaluate strategies.

a) With starting values ($R_0 = 2.25$, max days = 250, sdp1 $\in [30.60]$, reduction in first period = 0.6 and reduction in second period 0.8), what happens to the Final Fraction with and without interventions when the social distance period is allowed to be longer
Solution: Extending the first social distance period max days to 105 (up from 60) yields no change to the fraction without (it remains at 85) but decreases the fraction with (from 68 to 52). We see that the green curve "flattens" when the time frame for the intervention is larger.



b) What happens to them if R_0 is increased or decreased?
Solution: Increasing R_0 from 2.25 to 2.75 produces two very steep early peaks. The final fractions are 92 without and 76 with. Both epidemics seem to have finished by some 60 days with such a high reproduction number. The intervention does not seem to do much

in the first case. Decreasing R_0 to 1.75 instead delays the start of the peak and yields two curves which look different; the green curve (with interventions) is much wider and much less steep than the purple curve.



c) What happens when the reductions are the same, e.g. reduction after first period and reduction after second period are both 0.5? What happens when the first is larger than the second and when the second is larger than the first?
Solution: The green curve flattens because the values are reduced in both instances. As discussed in the exercise session, if the reduction is the same during *and* after socail distancing, the length of the social distancing period does not matter.

(*Hint*: Noting changes in different parameters may help, e.g. "increasing R_0 and decreasing `sdp1` has the effect of ..., keeping model parameters fixed but increasing the time interval has the effect of ...")

As background information you might want to read the blog post "Coronavirus: The Hammer and the Dance" by Tomas Pueyo.