Optimisation with the SEIRD model

Hui Jia Farm and Chon Lok Lei

2021-10-19

```r
library(comomodels)
library(ggplot2)
library(gridExtra)
library(glue)
library(dplyr)
library(tidyr)
```

Choose between plotting cached data or run optimisation

```r
# TRUE to plot saved data, FALSE to run optimisation
cached_bool <- TRUE
set.seed(0)
```

Introduction

This document shows an application of the SEIRD model class within the `como-models` package to the COVID-19 data. By fitting the SEIRD model to the COVID-19 data, we can use the model to learn (such as) the transmission rate and the death rate of the epidemic. We can also make predictions of the pandemic using the fitted model. However, due to limited information and presence of uncertainty, the data may not be able to confine or identify a single set of model parameters for the data; that is, multiple sets of parameters can fit the model to the data whilst giving same objective value (in our example, it is the log-likelihood value). This is known as an issue of identifiability in the model parameters.

In this vignette, we show how optimisation can be done with the `como-models` package, and how the identifiabilty of parameters in the SEIRD model can be assessed. We also demonstrate that one has to be caution when intepreting optimisation results. Throughout the vignette, we assume that there is no unreported cases.

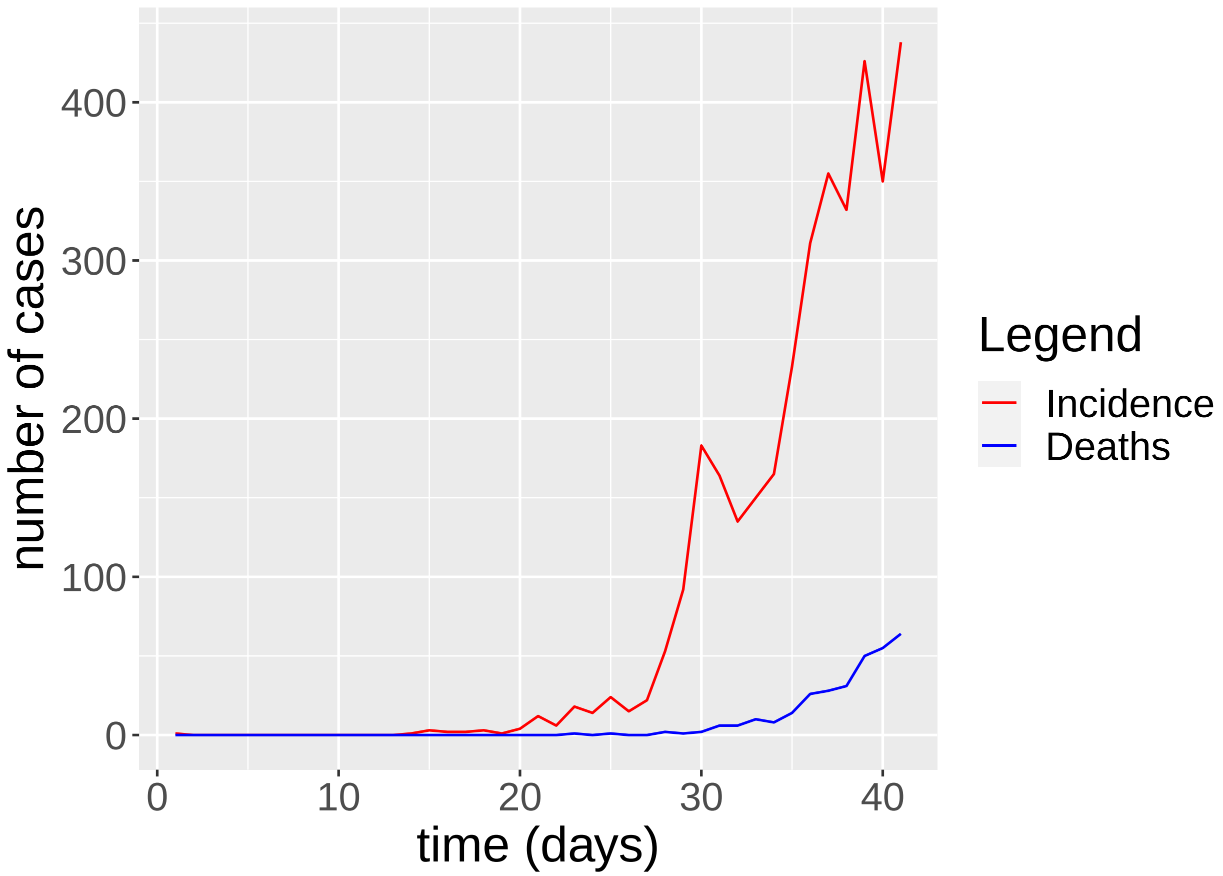
Optimisation on real London COVID-19 data

First, we will try to optimise the model by fitting to the London COVID-19 data. The time period of data we used is set to be the period before any interventions was implemented, lockdown in this case. Each interventions can be thought as a change of the SEIRD model parameters. Since we are fitting only one set of parameters, constant throughout the whole period, we use the London COVID-19 data only up to the implementation of lockdown. The population size for London is assumed to be 9×10^6; it is the estimated population of London in 2020 extracted from Ref. [london_data].

```r
# Load London COVID-19 data
df_london = read.csv(’data/London_data.csv’, header = TRUE)

# Extracting cases and deaths and renaming columns
cases_deaths_name_london = c(’newCasesBySpecimenDate’, ’newDailyNsoDeathsByDeathDate’)
df_london = df_london[cases_deaths_name_london]
names(df_london)[names(df_london) == "newCasesBySpecimenDate"] <- "DailyCases"
names(df_london)[names(df_london) == "newDailyNsoDeathsByDeathDate"] <- "DailyDeaths"

# Set population size of London
population_size <- 9e6
```

We first inspect the daily incidences and deaths in London, over the period of 1st January 2020 to 22 March 2020, where 23 March 2020 is the start of the lockdown.

```r
# Extract COVID-19 data prior to lockdown
London = df_london[42:82, 1:2] # Start of lockdown (23/03/2020) is 83rd day
rownames(London) = seq(length=nrow(London))
London$time <- seq(length=nrow(London))

# Visualise the data
ggplot() +
  geom_line(data=London, aes(x = time, y = DailyCases, color="Incidence")) +
  geom_line(data=London, aes(x = time, y = DailyDeaths, color="Deaths")) +
  scale_color_manual(name = "Legend",
                     values = c("Incidence" = "red", "Deaths" = "blue")) +
  labs(x = "time (days)", y = "number of cases") +
  theme(text = element_text(size = 20))
```



The figure above shows the incidences and deaths in London, starting from 1st January 2020 to 22 March 2020.

In this example, we will perform maximum likelihood estimation; we first define a (log-)likelihood function to be maximised. We will use a Poisson likelihood to model (the noise of) the data. Note that it takes the average of the log-likelihood values of incidences and deaths. With L as the likelihood function, N as the size of the data, k as the actual data and λ as estimator of the SEIRD model, the log-likelihood function is

$$\log L = \sum_{i=1}^{N} (k_i \log \lambda_i - \lambda_i)$$

```
# Define the log-likelihood function
RealData_LogLikelihoodFn <- function(parameters, model=SEIRD(),
                                     inc_numbers=0, death_numbers=0) {

  # Set up parameters
  transmission_para <- list(beta=parameters[1],
                            kappa=parameters[2],
                            gamma=parameters[3],
                            mu=parameters[4])
  init_cond <- list(S0=parameters[5],
                    E0=parameters[6],
                    I0=parameters[7],
                    R0=parameters[8])
  transmission_parameters(model) <- transmission_para
  initial_conditions(model) <- init_cond

  # Simulate model
  times <- seq(0, length(inc_numbers), by = 1)
  out_df <- run(model, times)

  # Organise data and scale it with size of population
  data_wide <- spread(out_df$changes, compartment, value)
  data_wide$Incidence <- abs(data_wide$Incidence) * population_size
  data_wide$Deaths <- abs(data_wide$Deaths) * population_size

  # Computation of log-likelihood function
  logIncidence <- log(data_wide$Incidence[-1])
  incidence_likelihood <- sum(inc_numbers * logIncidence - data_wide$Incidence[-1])
  logDeaths <- log(data_wide$Deaths[-1])
  death_likelihood <- sum(death_numbers * logDeaths - data_wide$Deaths[-1])

  (incidence_likelihood + death_likelihood)/2
}
```

Optimisation

The optimisation of the SEIRD model is repeated 100 times, with different initial values for all transmission parameters. Due to the heuristic approach of Nelder-Mead method, each run of optimisation will return different sets of optimised parameters.

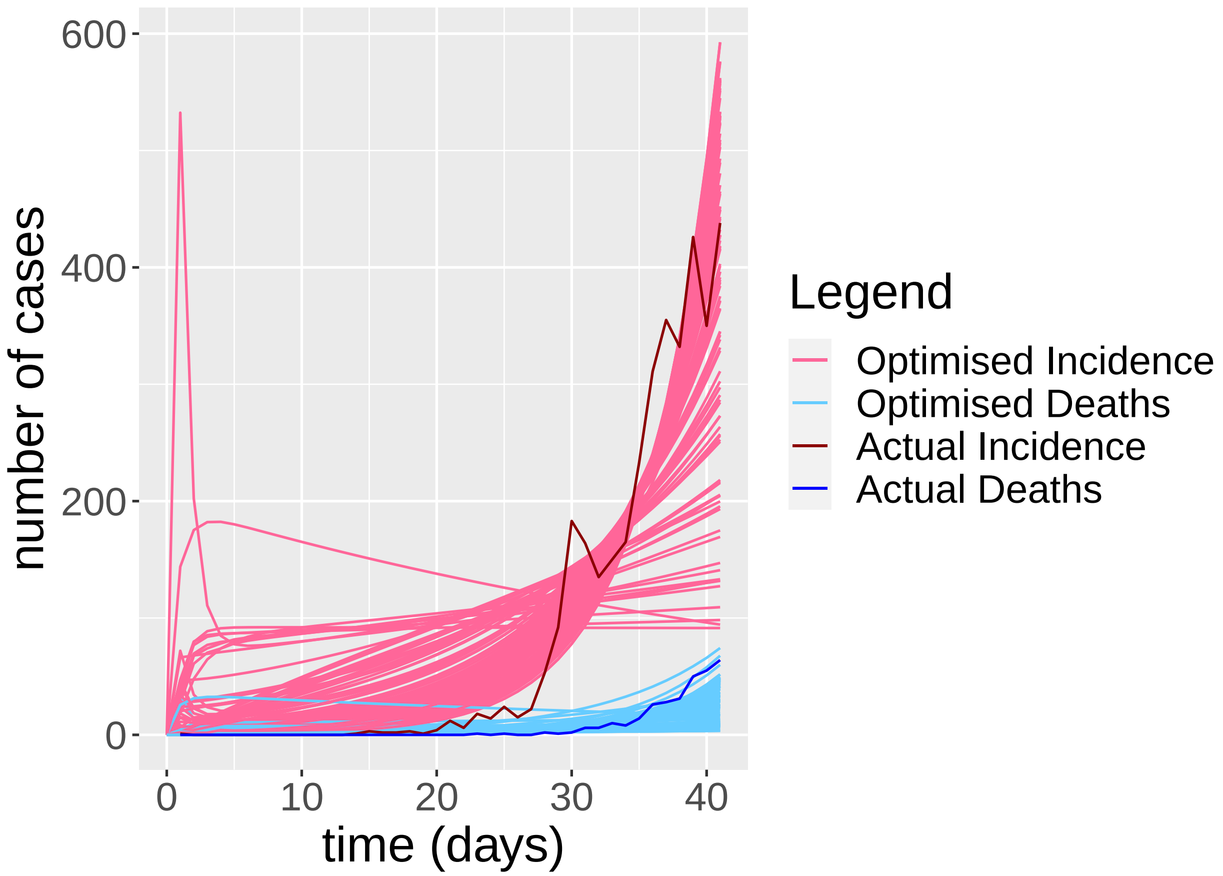
```
# Set up conditions for optimisation
init_conds_guess<- c(1-3e-7, 1e-7, 1e-7, 1e-7)
constraint_ui <- rbind(diag(8), -diag(8)[5:8,],
                       c(rep(0, 4), rep(1,4)),
                       c(rep(0, 4), rep(-1,4)))
constraint_ci <- c(rep(0, 8), rep(-1, 4), 0.99, -1.01)
repeat_num <- 100

# Run optimisation 100 times with different initial transmission parameters
for (repeats in 1:repeat_num){
    trans_params_guess <- runif(4, min = 0, max = 1)
    result <- constrOptim(c(trans_params_guess, init_conds_guess),
                          RealData_LogLikelihoodFn, ’NULL’, constraint_ui,
                          constraint_ci, method = "Nelder-Mead",
                          control=list(fnscale=-1, reltol=1e-6),
                          inc_numbers = London$DailyCases,
                          death_numbers = London$DailyDeaths)

    # Create data frame to save initial guesses and optimised parameters
    if (repeats == 1){
    optimised_para <- data.frame("beta_opt" = result$par[1], "kappa_opt" = result$par[2],
                                 "gamma_opt" = result$par[3], "mu_opt" = result$par[4],
                                 "S0_opt" = result$par[5], "E0_opt" = result$par[6],
                                 "I0_opt" = result$par[7], "R0_opt" = result$par[8],
                                 "obj_fn" = result$value)
    initial_guesses <- data.frame("beta_init" = trans_params_guess[1],
                                  "kappa_init" = trans_params_guess[2],
                                  "gamma_init" = trans_params_guess[3],
                                  "mu_init" = trans_params_guess[4])
    } else {
      optimised_para[nrow(optimised_para) + 1,] <- c(result$par, result$value)
      initial_guesses[nrow(initial_guesses) + 1,] <- c(trans_params_guess)
    }
}

opt_init_df <- merge(optimised_para, initial_guesses, by = 0)

# save optimised parameters
write.csv(opt_init_df, "data/London_parameters_100_optimisations.csv", row.names = FALSE)
```

```
# load saved results
opt_init_df <- read.csv("data/London_parameters_100_optimisations.csv")
repeat_num <- nrow(opt_init_df)
```

Visualising optimised result

```
# Define function to plot data and optimised trajectory
optimisation_plot <- function(parameters_df, time_length_scale=1, plot_real_data=TRUE){

    # Extract optimised parameters from data frame
    optimised_para <- subset(parameters_df, select=c(beta_opt, kappa_opt, gamma_opt,
                                                     mu_opt, S0_opt, E0_opt,
                                                     I0_opt, R0_opt))
    colnames(optimised_para) <- c("beta", "kappa", "gamma", "mu", "S0", "E0", "I0", "R0")

    # Simulate the model for all optimised parameters
    for (repeats in 1:nrow(parameters_df)){

        my_model <- SEIRD()

        transmission_parameters(my_model) <- optimised_para[repeats, 1:4]
        initial_conditions(my_model) <- optimised_para[repeats, 5:8]

        times <- seq(0, length(London$DailyCases) * time_length_scale, by = 1)
        out_df <- run(my_model, times)

        cases <- out_df$changes
        cases$value <- cases$value * population_size
        cases$repeat_id <- rep(repeats, length(cases$time))

        if (repeats == 1){
          cases_repeats <- cases
        } else{
          cases_repeats <- rbind(cases_repeats, cases)
        }
    }
```

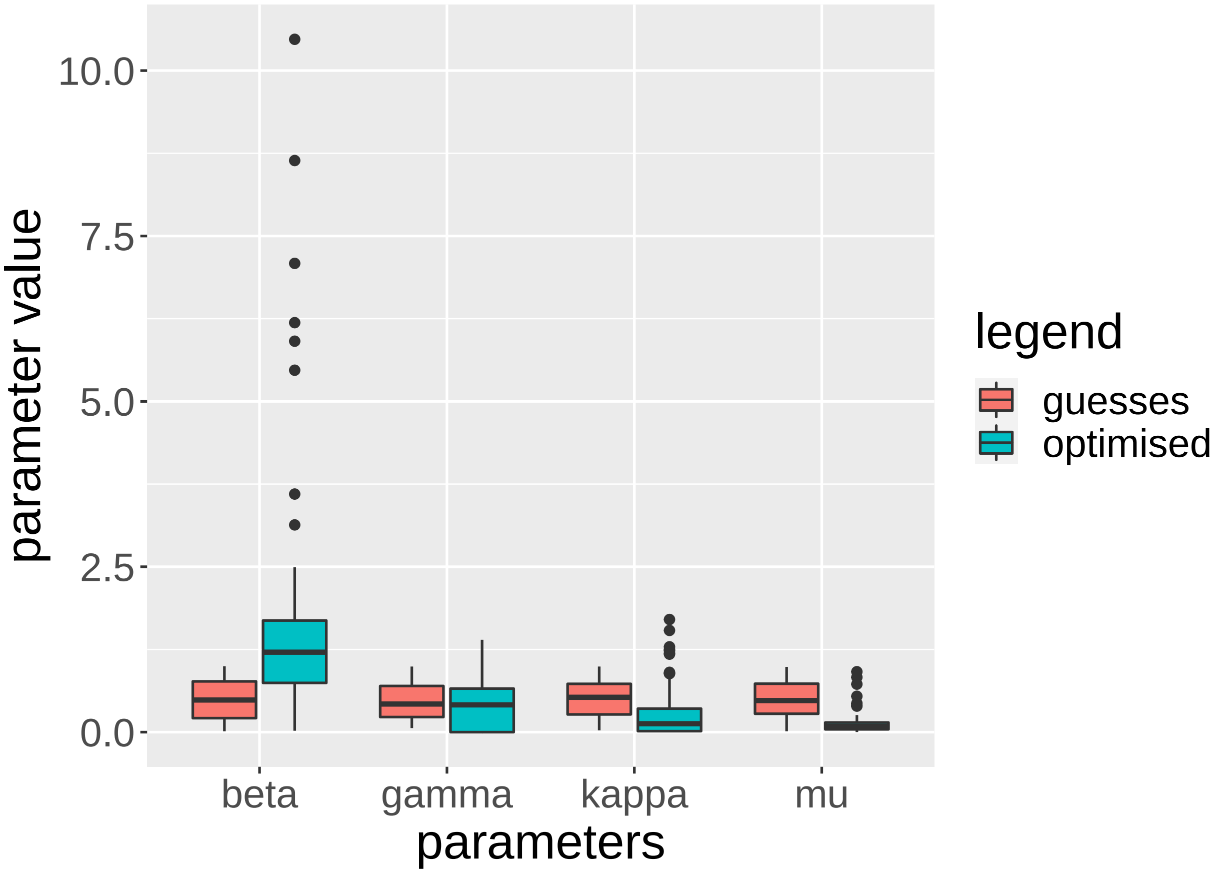
```r
    # Visualise simulated trajectory
    cases_repeats <- spread(cases_repeats, compartment, value)
    b <- ggplot()
    # Plot optimised cases
    for (i in 1:nrow(parameters_df)){
      data_set <- cases_repeats[cases_repeats$repeat_id == i,]
      b <- b + geom_line(data=data_set, aes(x = time, y = Incidence,
                                            color="Optimised Incidence")) +
      geom_line(data=data_set, aes(x = time, y = Deaths, color="Optimised Deaths"))
    }
    # Plot actual data
    if (plot_real_data == TRUE){
        b <- b + geom_line(data=London, aes(x = time, y = DailyCases,
                                            color="Actual Incidence")) +
                geom_line(data=London, aes(x = time, y = DailyDeaths,
                                           color="Actual Deaths"))
    }
    b <- b + scale_color_manual(name = "Legend",
                                values = c("Optimised Incidence" = "#FF6699",
                                           "Optimised Deaths" = "#66CCFF",
                                           "Actual Incidence" = "darkred",
                                           "Actual Deaths" = "blue")) +
        labs(x = "time (days)", y = "number of cases") +
        theme(text = element_text(size = 20))
    print(b)
}

optimisation_plot(opt_init_df)
```



The figure above shows the 100 fitted models on the actual incidences and deaths. It can be observed from the figure that different runs of optimisation returns different results. Some of the fits even have their peaks in the first 10 days of the pandemic.

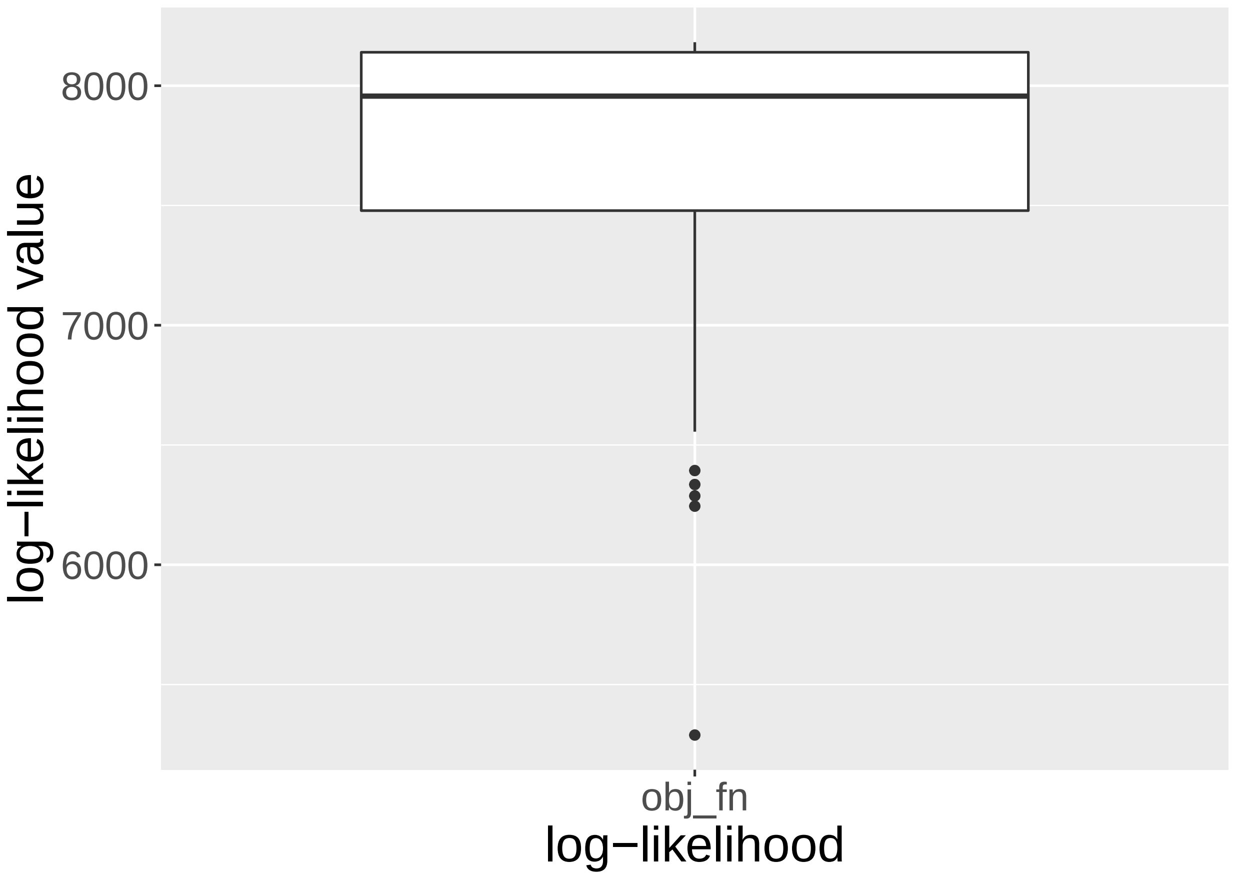
Box plot of estimated variables

```
# Organise data for box plot
plotting_data <- subset(opt_init_df, select=c(beta_opt, kappa_opt, gamma_opt, mu_opt))
colnames(plotting_data) <- c("beta", "kappa", "gamma", "mu")
plotting_data$index <- seq(1, repeat_num)
plotting_data$legend <- rep("optimised", repeat_num)
initial_guesses <- subset(opt_init_df,
                          select=c(beta_init, kappa_init, gamma_init, mu_init))
colnames(initial_guesses) <- c("beta", "kappa", "gamma", "mu")
initial_guesses$index <- seq(1, repeat_num)
initial_guesses$legend <- rep("guesses", repeat_num)
data <- rbind(plotting_data, initial_guesses)
data <- gather(data, key="parameters", value="value", -c(index, legend))

# Plot initial guesses and optimised parameters from optimisation
# in box plot format
ggplot(data, aes(x=parameters, y=value, fill=legend)) +
    geom_boxplot() +
    labs(y = "parameter value") +
    theme(text = element_text(size = 20))
```



The box plot compares the summary statistics of prior and posterior of transmission parameters, which are β, κ, γ and μ.

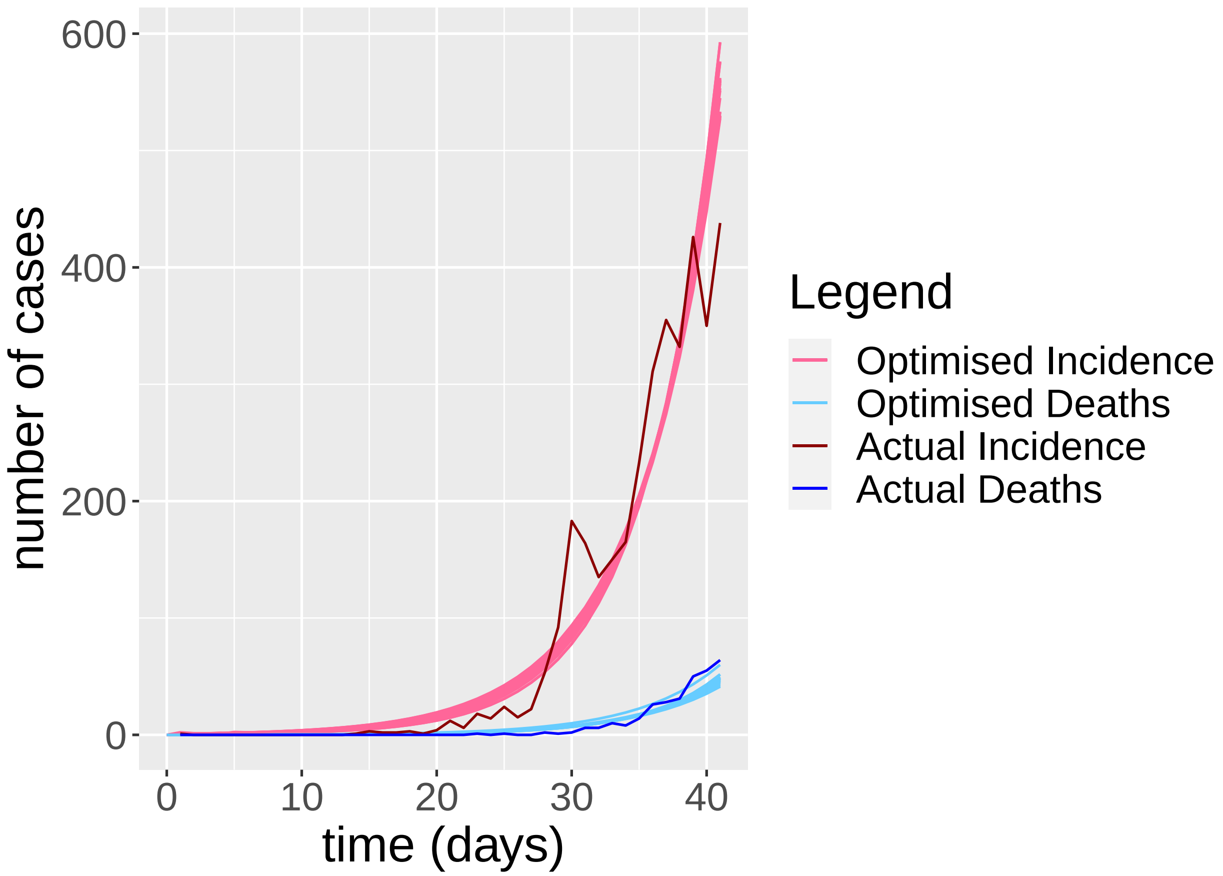
```
# Create box plot for log-likelihood values of all optimisations
plotting_data <- subset(opt_init_df, select=c(obj_fn))
plotting_data$index <- seq(1, repeat_num)
data <- gather(plotting_data, key="likelihood_value", value="value", -c(index))
ggplot(data, aes(x=likelihood_value, y=value)) +
    geom_boxplot() +
    labs(x = "log-likelihood", y = "log-likelihood value") +
    theme(text = element_text(size = 20))
```



The box plot shows a range of log-likelihood values achieved from different runs of optimisation. Note that the Nelder-Mead algorithm used here is a local optimisation algorithm, which does not guarantee a convergence to the global optimum. The result confirms that some runs of optimisation perform better than others, which can happen when the optimising algorithm reaches a local optimum.

After sorting the obtained paramters according to the error value, we choose the top 20 optimisation with highest log-likelihood values as our final maximum-likelihood estimates.

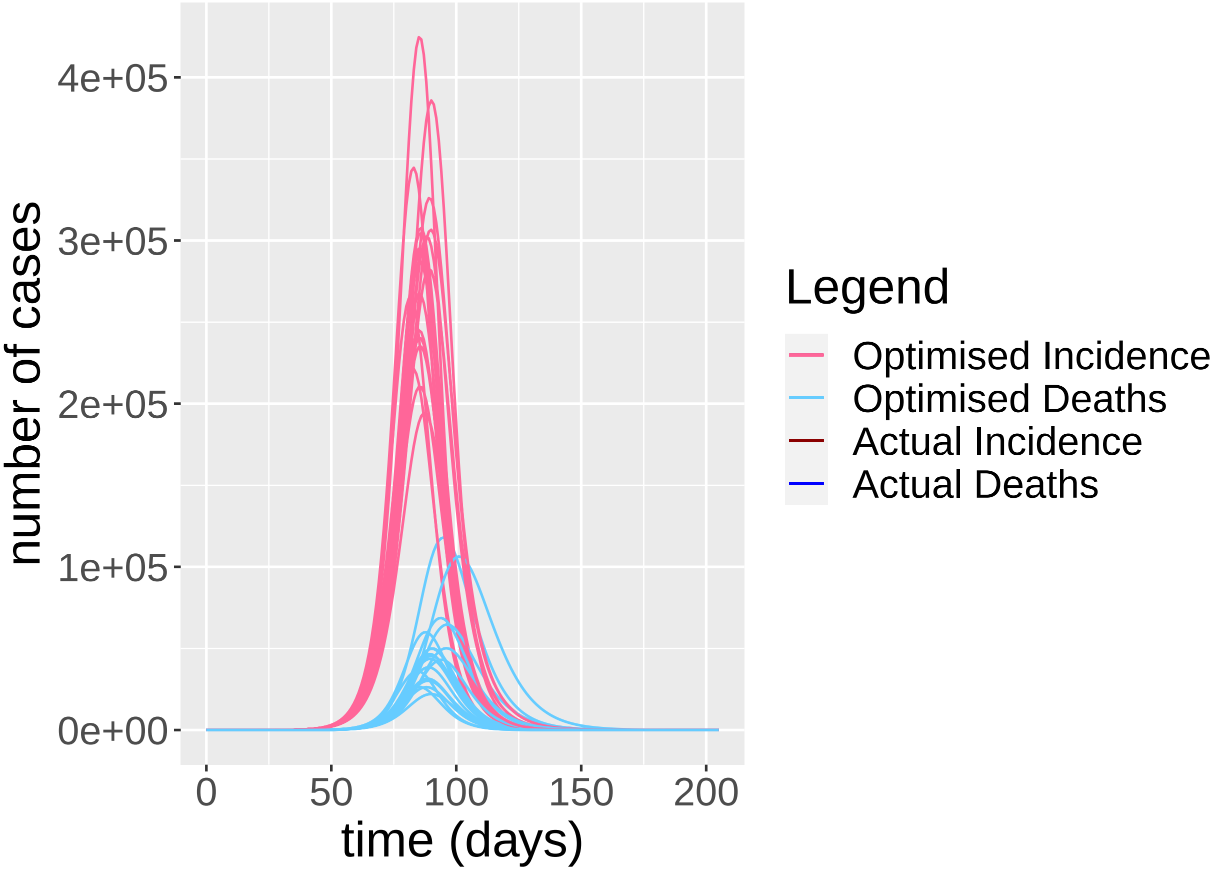
```r
# Sort data in order of highest log-likelihood value
chosen_repeats <- 20
opt_init_df <- opt_init_df[order(-opt_init_df$obj_fn),]
top_opt_init_df <- opt_init_df[1:chosen_repeats,]

# Plot trajectories of optimisation with highest log-likelihood value
optimisation_plot(top_opt_init_df)
```



The figure aboe shows the 20 optimisations that achieve highest log-likelihood value.

Projection of the epidemic using the fitted SEIRD model

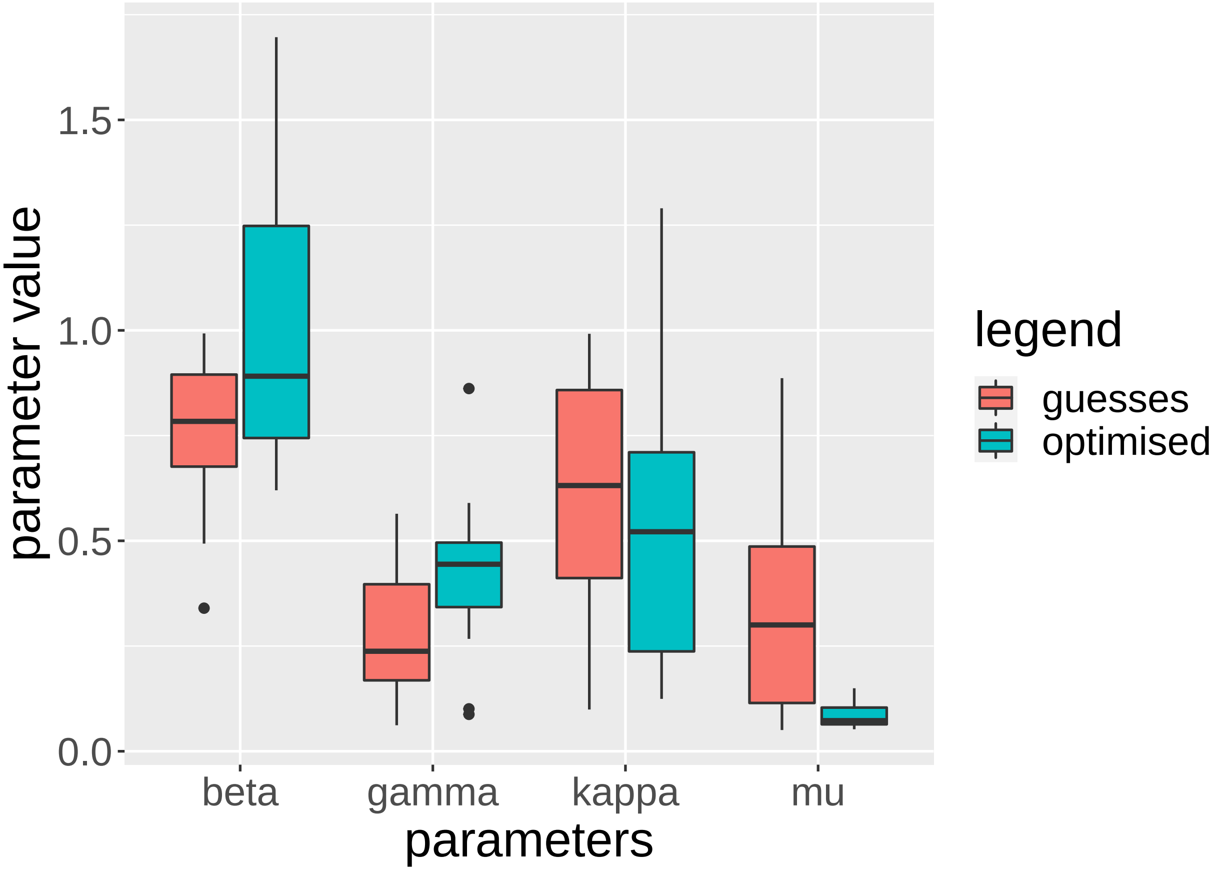
```
# Plot prediction of the epidemic
optimisation_plot(top_opt_init_df, time_length_scale = 5, plot_real_data = FALSE)
```



The figure above shows that even when the optimisation fits the real data well during early stages of the pandemic, (very) different outcomes are predicted. Given the estimated London population, the maximum difference in peak values of all projections is more than 2×10^5 for incidences and almost 1×10^5 for deaths.

Box plot of the estimated variables and log-likelihood value

```r
# Organise data for box plot
plotting_data <- subset(top_opt_init_df, select=c(beta_opt, kappa_opt, gamma_opt, mu_opt))
colnames(plotting_data) <- c('beta', 'kappa', 'gamma', 'mu')
plotting_data$index <- seq(1, chosen_repeats)
plotting_data$legend <- rep("optimised", chosen_repeats)
initial_guesses <- subset(top_opt_init_df, select=c(beta_init, kappa_init,
                                                    gamma_init, mu_init))
colnames(initial_guesses) <- c("beta", "kappa", "gamma", "mu")
initial_guesses$index <- seq(1, chosen_repeats)
initial_guesses$legend <- rep("guesses", chosen_repeats)
data <- rbind(plotting_data, initial_guesses)
data <- gather(data, key="parameters", value="value", -c(index, legend))

# Plot initial guesses and optimised parameters from optimisation
# in box plot format
ggplot(data, aes(x=parameters, y=value, fill=legend)) +
    geom_boxplot() +
    labs(y = "parameter value") +
    theme(text = element_text(size = 20))
```



The box plot shows the range of optimised parameters of optimisations with top 20 highest log-likelihood values. Despite achieving the largest objective value, the transmission parameters, except μ, display a distribution of optimised values, with large standard deviation. This shows that different sets of transmission parameters can achieve similar log-likelihood values. With the London COVID-19 data, the SEIRD model cannot identify the parameters uniquely.

Synthetic data studies and profile likelihood

To assess the identifiability issue observed in optimisations of the real data, we perform a profile likelihood analysis over some *synthetic data*. The profile likelihood is a series of maximum likelihood value obtained by fixing a parameter of interest to a range of values and optimising the remaining parameters.

Generating synthetic data

The mean value of the synthetic data is simulated with values that provide sufficient time points before the peak of the pandemic, which qualitatively replicates the London COVID-19 data shown above.

```
# Set up SEIRD model
model <- SEIRD()
simulating_para <- list(beta=9.1e-1, kappa=8.7e-1, gamma=5.3e-1, mu=9.7e-2,
                        S0=9.999e-1, E0=1e-7, I0=1e-7, R0=1e-6)
tranmission_para_name <- c("beta", "kappa", "gamma", "mu")
initial_conditions_name <- c("S0", "E0", "I0", "R0")
transmission_parameters(model) <- simulating_para[tranmission_para_name]
initial_conditions(model) <- simulating_para[initial_conditions_name]

# Simulate the model to create synthetic data
times <- seq(0, 150, by = 1)
out_df <- run(model, times)
```

```
model <- SEIRD()
simulating_para <- list(beta=9.1e-1, kappa=8.7e-1, gamma=5.3e-1, mu=9.7e-2,
                        S0=9.999e-1, E0=1e-7, I0=1e-7, R0=1e-6)
```

Adding noise to the synthetic data

The synthetic data is then generated by drawing samples from a Poisson distribution with λ as the incidences and deaths of model simuated value, as shown in the equations below. $C(t)$ and $D(t)$ are the number of incidences and deaths at time t respectively; $C^*(t)$ and $D^*(t)$ are the number of incidences and deaths with Poisson noise.

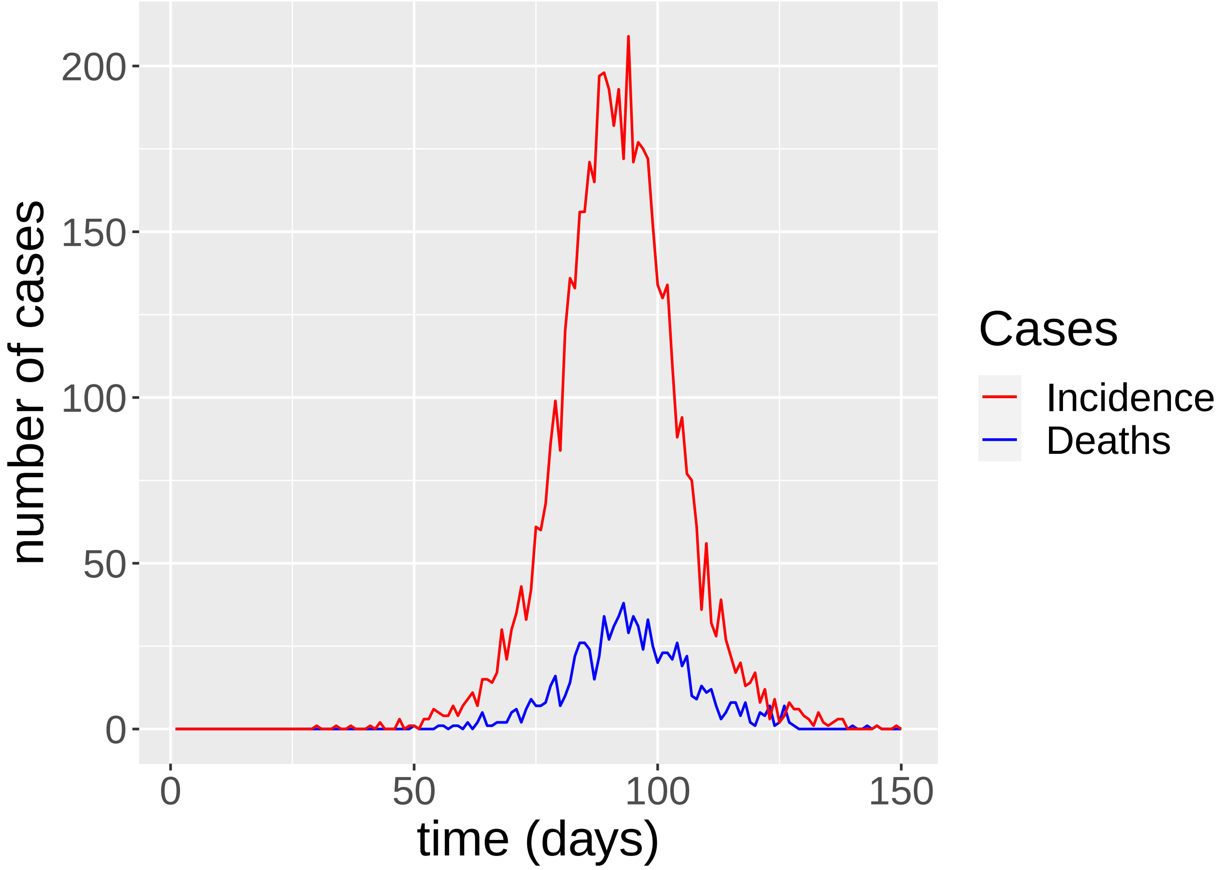
$$C^*(t) \sim \mathsf{Pois}(C(t))$$
$$D^*(t) \sim \mathsf{Pois}(D(t))$$

```
population_size <- 1e4

# add Poisson noise to synthetic data
testing_data <- spread(out_df$changes, compartment, value)
testing_data$Incidence <- testing_data$Incidence * population_size
testing_data$Deaths <- testing_data$Deaths * population_size
inc_noise <- rpois(1, testing_data$Incidence[1])
death_noise <- rpois(1, testing_data$Deaths[1])
for (i in 2:length(testing_data$Incidence)){
  inc_rand <- rpois(1, testing_data$Incidence[i])
  inc_noise <- c(inc_noise, inc_rand)

  death_rand <- rpois(1, testing_data$Deaths[i])
  death_noise <- c(death_noise, death_rand)
}
testing_data$IncNoise <- inc_noise
testing_data$DeathNoise <- death_noise
testing_data <- testing_data[-1,]

# save results
write.csv(testing_data, "data/synthetic_data.csv", row.names = FALSE)
```

```
# load saved data
population_size <- 1e4
testing_data <- read.csv("data/synthetic_data.csv")
```

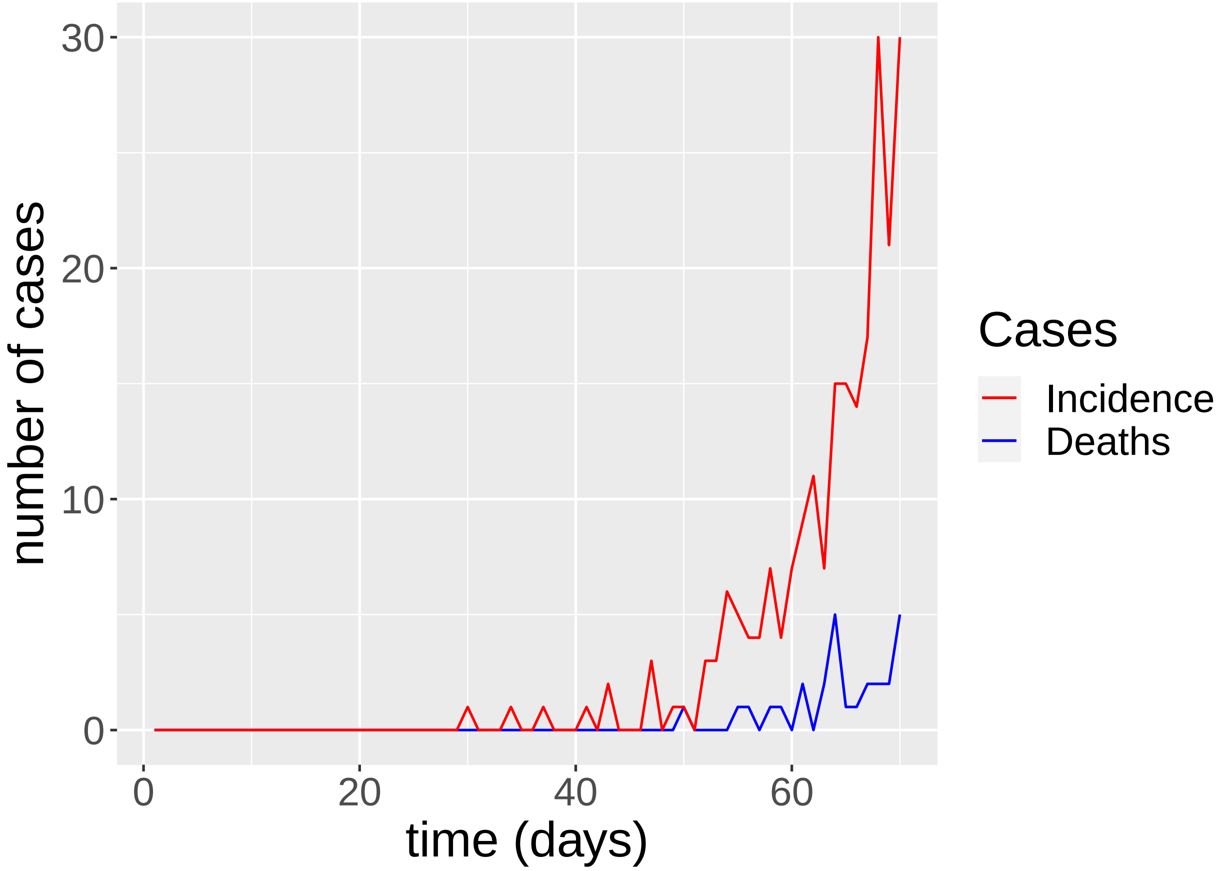
Visualising the synthetic data

```
# Plot synthetic data
ggplot() +
  geom_line(data=testing_data, aes(x = time, y = DeathNoise, color = "Deaths")) +
  geom_line(data=testing_data, aes(x = time, y = IncNoise, color = "Incidence")) +
  scale_color_manual(name = "Cases", values = c("Incidence" = "red", "Deaths" = "blue")) +
  labs(x = "time (days)", y = "number of cases") +
  theme(text = element_text(size = 20))
```



The figure above shows the synthetic data simulated for the full pandemic.

```
# Extract data prior to peak
testing_data_short <- testing_data[1:70,]

# Visualise data
ggplot() +
  geom_line(data=testing_data_short, aes(x = time, y = DeathNoise, color = "Deaths")) +
  geom_line(data=testing_data_short, aes(x = time, y = IncNoise, color = "Incidence")) +
  scale_color_manual(name = "Cases", values = c("Incidence" = "red", "Deaths" = "blue")) +
  labs(x = "time (days)", y = "number of cases") +
  theme(text = element_text(size = 20))
```



The first 70 times points (days) of the simulated data is used, giving roughly 40 times points since the first case, which is similar to the London COVID-19 data.

Again, a Poisson log-likelihood function is used to fit the SEIRD model to the simulated data. The profile likelihood is then constructed for the transmission parameters, while the initial conditions are fixed at values used to generate the simulated data.

```
# Define log-likelihood function to allow fixing of parameters
LogLikelihoodFn <- function(parameters, model=SEIRD(), inc_numbers=0, death_numbers=0,
                    profile_parameters=profile_parameters, fixed_parameter=FALSE,
                    fixed_parameter_value=0) {

  # Set up parameters for the SEIRD model
  optimising_para <- simulating_para
  for (i in 1:length(profile_parameters)){
      optimising_para[profile_parameters[i]] <- parameters[i]
  }

  # Fixing parameter of interest
  if (fixed_parameter != FALSE){
    optimising_para[names(optimising_para) == fixed_parameter] <-
            fixed_parameter_value}

  transmission_parameters(model) <- optimising_para[c('beta', 'kappa', 'gamma', 'mu')]
  initial_conditions(model) <- optimising_para[c('S0', 'E0', 'I0', 'R0')]
  # Simulate model
  times <- seq(0, length(inc_numbers), by = 1)
  out_df <- run(model, times)

  # Organise and scale data
  data_wide <- spread(out_df$changes, compartment, value)
  data_wide$Incidence <- abs(data_wide$Incidence) * population_size
  data_wide$Deaths <- abs(data_wide$Deaths) * population_size

  # Calculate log-likelihood value
  logIncidence <- log(data_wide$Incidence[-1])
  incidence_likelihood <- sum(inc_numbers * logIncidence - data_wide$Incidence[-1])
  logDeaths <- log(data_wide$Deaths[-1])
  death_likelihood <- sum(death_numbers * logDeaths - data_wide$Deaths[-1])

  (incidence_likelihood + death_likelihood)/2
}
```

Profile likelihood of β and γ

In some cases, identifiability issues arise when there are more parameters describing the model than the information can infer. In other words, parameters cannot be uniquely identified with limited data. When the number of parameters increases, the parameters might become unidentifiable. Here, we first create a profile likelihood for only two of the four transmission parameters in the SEIRD model.

Setting ranges for profile likelihood

```
# Set interested parameters for profile likelihood
profile_parameters <- c(’beta’, ’gamma’)

# Set range of values for interested parameters
beta_range <- c(seq(simulating_para$beta, 0.4, by = -0.02),
                seq(simulating_para$beta, 1.5, by = 0.02))
gamma_range <- c(seq(simulating_para$gamma, 0.1, by = -0.02),
                 seq(simulating_para$gamma, 1.1, by = 0.02))

# Create a data frame for the range of values
range_transmission <- data.frame(parameter=rep(’beta’, length(beta_range)),
                                 fixed_value=beta_range)
range_transmission <- rbind(range_transmission, data.frame(
  parameter=rep(’gamma’, length(gamma_range)), fixed_value=gamma_range))
```

Optimisation

```
# Define function to run the optimisations for profile likelihood
profilelikelihood_opt <- function(profile_parameters, range_transmission,
                                  likelihoodfn, synthetic_data, reltol=1e-6){
  # Set up structure
  previous_param <- rep(0, length(profile_parameters))
  profile_likelihood <- data.frame(parameter=character(), fixed_value=double(),
                                 likelihood_value=double())
  profile_likelihood$parameter <- as.character(profile_likelihood$parameter)
  for (fixed_param in profile_parameters){
    profile_likelihood[paste(’optim_’, fixed_param, sep = "")] <- double()
  }

  # Set up constraints
  constraint_ui <- rbind(diag(length(profile_parameters)))
  constraint_ci <- c(rep(0,length(profile_parameters)))

  # Run optimisation for all values in the range defined above
  set.seed(0)
  for (param_name in profile_parameters){
    err_value <- 0
    fixed_values <- range_transmission %>% filter(parameter == param_name)
    fixed_values <- fixed_values$fixed_value
    for (i in 1:length(fixed_values)){
      if (param_name == ’beta’ & fixed_values[i] == simulating_para$beta){
        init_guess <- simulating_para[profile_parameters]
      } else if (param_name == ’kappa’ & fixed_values[i] == simulating_para$kappa){
      init_guess <- simulating_para[profile_parameters]
      } else if (param_name == ’gamma’ & fixed_values[i] == simulating_para$gamma){
      init_guess <- simulating_para[profile_parameters]
      } else if (param_name == ’mu’ & fixed_values[i] == simulating_para$mu){
      init_guess <- simulating_para[profile_parameters]
      } else {
        init_guess <- previous_param}
    init_guess <- as.numeric(init_guess)
    result <- constrOptim(init_guess, likelihoodfn, ’NULL’, constraint_ui,
                          constraint_ci, method = "Nelder-Mead",
                          control=list(fnscale=-1, reltol=reltol),
                          model=model,
                          inc_numbers = synthetic_data$IncNoise,
                          death_numbers = synthetic_data$DeathNoise,
                          profile_parameters = profile_parameters,
                          fixed_parameter = param_name,
                          fixed_parameter_value = fixed_values[i])

    # Save the log-likelihood value and value of optimised parameters
    err_value <- append(err_value, result$value)
    previous_param <- result$par
    profile_likelihood[
      nrow(profile_likelihood) + 1,] <- c(
        param_name, fixed_values[i],
        result$value, result$par)
    }
  }
  ind <- seq(2, 3 + length(profile_parameters), by = 1)
  profile_likelihood[ ,ind] <- apply(profile_likelihood[ ,ind], 2,
                                     function(x) as.numeric(x))
  profile_likelihood
}

# Run optimisations to create profile likelihood
```

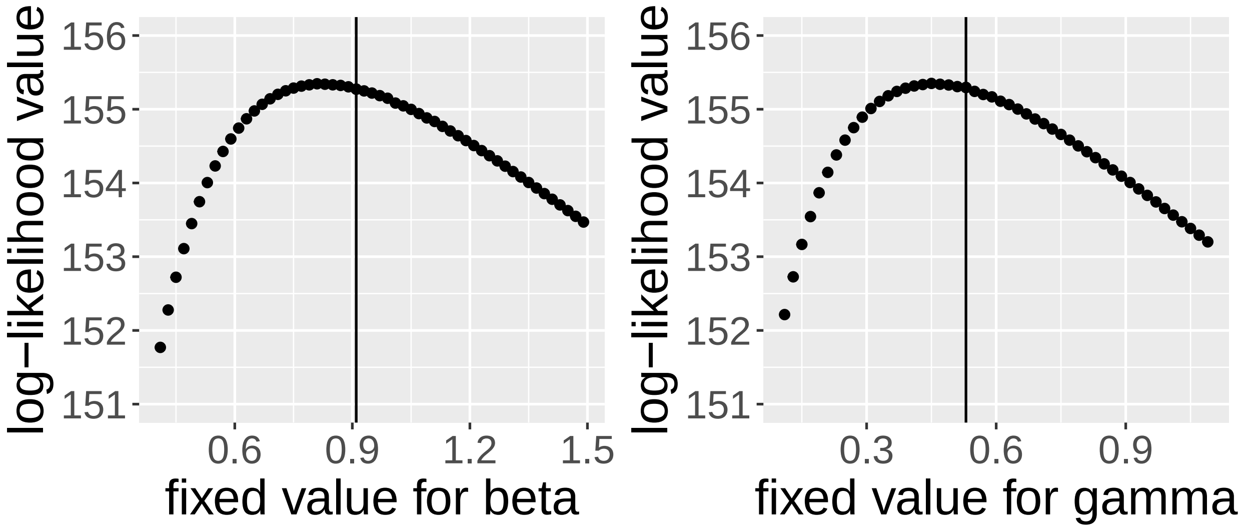
```
profile_likelihood <- profilelikelihood_opt(profile_parameters, range_transmission,
                                LogLikelihoodFn, testing_data_short)

# Save results
write.csv(profile_likelihood, "data/synthetic_halftrend_2param_profilelikelihood.csv",
          row.names = FALSE)
```

While working through the optimisations for different fixed values of parameter of interest, the previously optimised parameters were used as the initial guesses for the next round of optimisation. To ease the optimisation challenges for the profile likelihood, the parameters for the profile likelihood were sweeped from the actual parameter value to the minimum, then from the actual parameter value to the maximum.

```
# Load results
profile_likelihood <- read.csv("data/synthetic_halftrend_2param_profilelikelihood.csv")
```

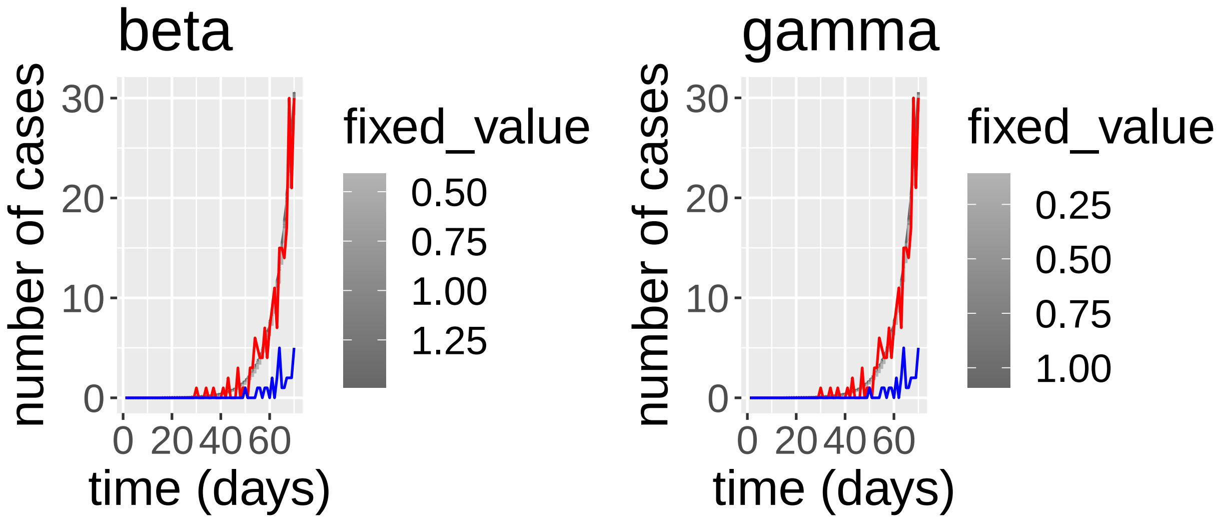
Profile likelihood

```
# Create function to plot the profile likelihood
profilelikelihood_plot <- function(profile_likelihood, profile_parameters){

    # Get minimum and maximum of all log-likelihood values
    ymin <- floor(min(profile_likelihood$likelihood_value))
    ymax <- ceiling(max(profile_likelihood$likelihood_value))

    # Plot the log-likelihood values for each parameters of interest
    profile_plot <- vector(’list’, length(profile_parameters))
    for (i in 1:length(profile_parameters)){
        plot_data <- profile_likelihood %>% filter(parameter == profile_parameters[i])
        profile_plot[[i]] <- ggplot(plot_data, aes(x = fixed_value,
                                                   y = likelihood_value)) +
            geom_vline(xintercept = as.numeric(simulating_para[profile_parameters[i]])) +
            geom_point() + ylim(ymin, ymax) +
            labs(x = paste("fixed value for", profile_parameters[i]),
                 y = "log-likelihood value") +
            theme(text = element_text(size = 20))
    }
    grid.arrange(grobs=profile_plot, nrow = length(profile_parameters)%/%2, ncol = 2)
}

profilelikelihood_plot(profile_likelihood, profile_parameters)
```



The figure above shows the log-likelihood value obtained by fixing the parameter of interest and optimising the remaining tranmission parameters. The actual parameter value is indicated by the vertical line. The profile likelihood shows a single peak for both β and γ, which implies that the parameters are identifiable when the SEIRD model has only two transmission parameters. However, there is a bias when using the maximum likelihood estimate (i.e. the parameter value with the maximum log-likelihood is not the same as the actual parameter value used to simulate the data). The bias is due to the Poisson noise added to the simulated data.

```
init_cond <- list(S0=9.999e-1,
                  E0=1e-7,
                  I0=1e-7,
                  R0=1e-6)

# Define function to plot the fitted trajectory against synthetic data
profilelikelihood_fit <- function(profile_likelihood, profile_parameters, synthetic_data){

    count <- 0
    trans_param <- list(beta=simulating_para$beta, kappa=simulating_para$kappa,
                        gamma=simulating_para$gamma, mu=simulating_para$mu)

    for (param_name in profile_parameters){
      temp <- profile_likelihood %>% filter(parameter == param_name)
      for (i in 1:nrow(temp)){
        model <- SEIRD()

        # Set up parameters from data frame of optimised parameters
        for (param_interest in profile_parameters){
            trans_param[names(trans_param) == param_interest] <-
                temp[i, paste('optim_', param_interest, sep = "")]
        }
        trans_param[names(trans_param) == param_name] <- temp[i,2]
        transmission_parameters(model) <- trans_param
        initial_conditions(model) <- init_cond

        # Run simulation
        times <- seq(0, length(synthetic_data$Incidence), by = 1)
        out_df <- run(model, times)
        # Organise simulated data
        fitted_cases <- out_df$changes
        fitted_cases <- spread(fitted_cases, compartment, value)
        fitted_cases$Incidence <- fitted_cases$Incidence * population_size
        fitted_cases$Deaths <- fitted_cases$Deaths * population_size
        fitted_cases <- fitted_cases[-1,]
        fitted_cases$parameter <- rep(param_name, length(fitted_cases$Incidence))
        fitted_cases$fixed_value <- rep(temp$fixed_value[i],
                                        length(fitted_cases$Incidence))

        if (count == 0){
          total_cases <- fitted_cases
        } else{
          total_cases <- rbind(total_cases, fitted_cases)
        }
        count <- count + 1
      }
    }

    # Plot all profile likelihoods
    total_cases <- transform(total_cases, fixed_value=as.numeric(fixed_value))
    optimised_cases <- vector(’list’, length(profile_parameters))
    for (i in 1:length(profile_parameters)){
        plotting_data <- total_cases %>% filter(parameter == profile_parameters[i])
        optimised_cases[[i]] <- ggplot(plotting_data,
                        aes(x = time, y = Incidence, color=fixed_value)) +
        geom_line() +
        scale_color_gradient(low = "grey70", high = "grey40") +
        guides(colour = guide_colorbar(reverse=TRUE)) +
        geom_line(data=synthetic_data, aes(x = time, y = IncNoise), color=’red’) +
        geom_line(data=synthetic_data, aes(x = time, y = DeathNoise), color=’blue’) +
        labs(x = "time (days)", y = "number of cases") +
        ggtitle(profile_parameters[i]) +
        theme(text = element_text(size = 20))
    }
    grid.arrange(grobs=optimised_cases, nrow = length(profile_parameters)%/%2, ncol = 2)
}

profilelikelihood_fit(profile_likelihood, profile_parameters, testing_data_short)
```



The figure above shows the model simulations using the parameters identified in the profile likelihood.

Profile likelihood of all transmission parameters

The same procedure is repeated with all the transmission parameters (instead of two out of the four): β, κ, γ and μ.

Setting ranges for profile likelihood

```r
# Set interested parameters for profile likelihood
profile_parameters <- c('beta', 'kappa', 'gamma', 'mu')

# Set range of values for interested parameters
beta_range <- c(seq(simulating_para$beta, 0.4, by = -0.02),
                seq(simulating_para$beta, 1.5, by = 0.02))
kappa_range <- c(seq(simulating_para$kappa, 0.6, by = -0.02),
                 seq(simulating_para$kappa, 1.2, by = 0.02))
gamma_range <- c(seq(simulating_para$gamma, 0.2, by = -0.02),
                 seq(simulating_para$gamma, 0.8, by = 0.02))
mu_range <- c(seq(simulating_para$mu, 0.02, by = -0.005),
              seq(simulating_para$mu, 0.2, by = 0.005))

# Create a data frame for the range of values
range_transmission <- data.frame(parameter=rep('beta', length(beta_range)),
                                 fixed_value=beta_range)
range_transmission <- rbind(range_transmission, data.frame(
  parameter=rep('kappa', length(kappa_range)), fixed_value=kappa_range))
range_transmission <- rbind(range_transmission, data.frame(
  parameter=rep('gamma', length(gamma_range)), fixed_value=gamma_range))
range_transmission <- rbind(range_transmission, data.frame(
  parameter=rep('mu', length(mu_range)), fixed_value=mu_range))
```

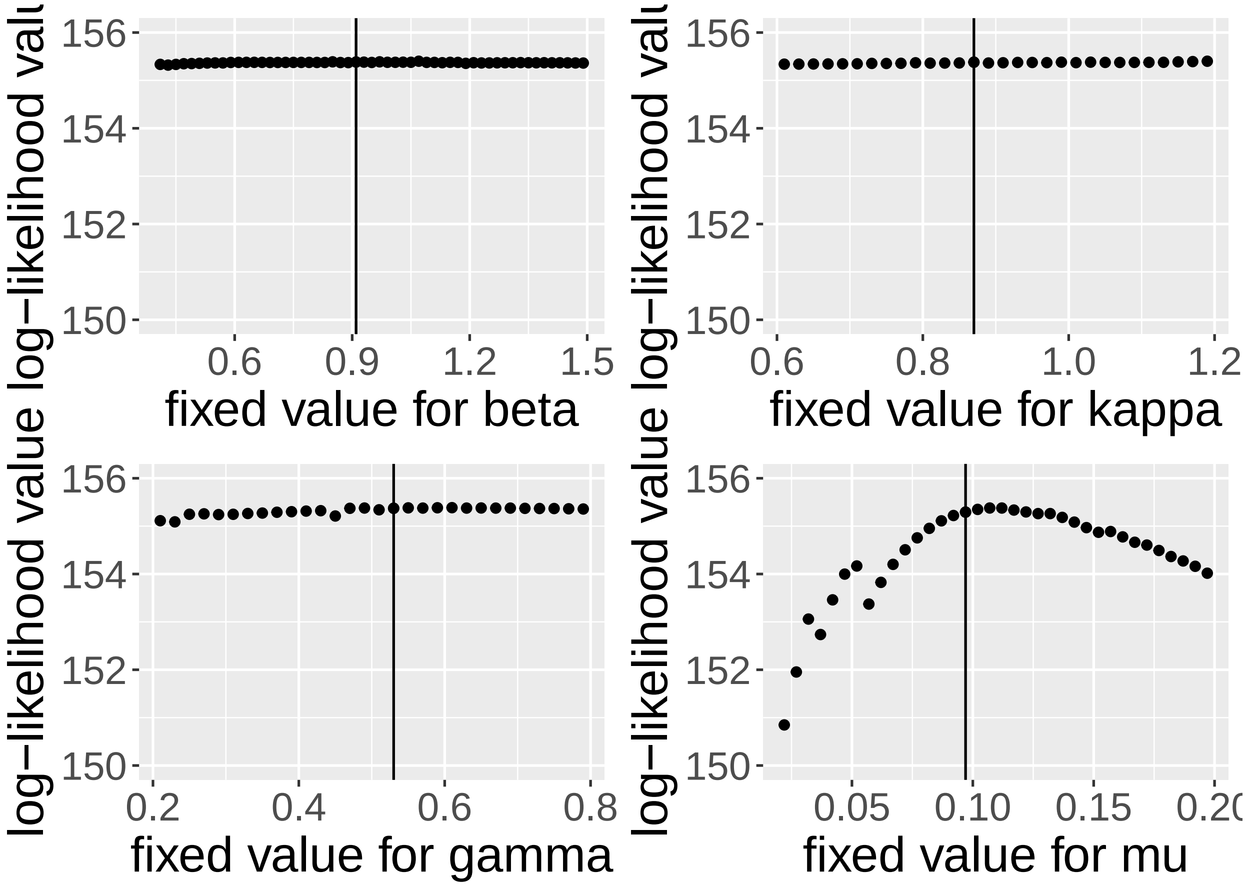
Optimisation

```
# Run optimisations to create profile likelihood
profile_likelihood <- profilelikelihood_opt(profile_parameters, range_transmission,
                                  LogLikelihoodFn, testing_data_short)
# Save results
write.csv(profile_likelihood, "data/synthetic_halftrend_4param_profilelikelihood.csv",
          row.names = FALSE)
```

```
# Load results
profile_likelihood <- read.csv("data/synthetic_halftrend_4param_profilelikelihood.csv")
```

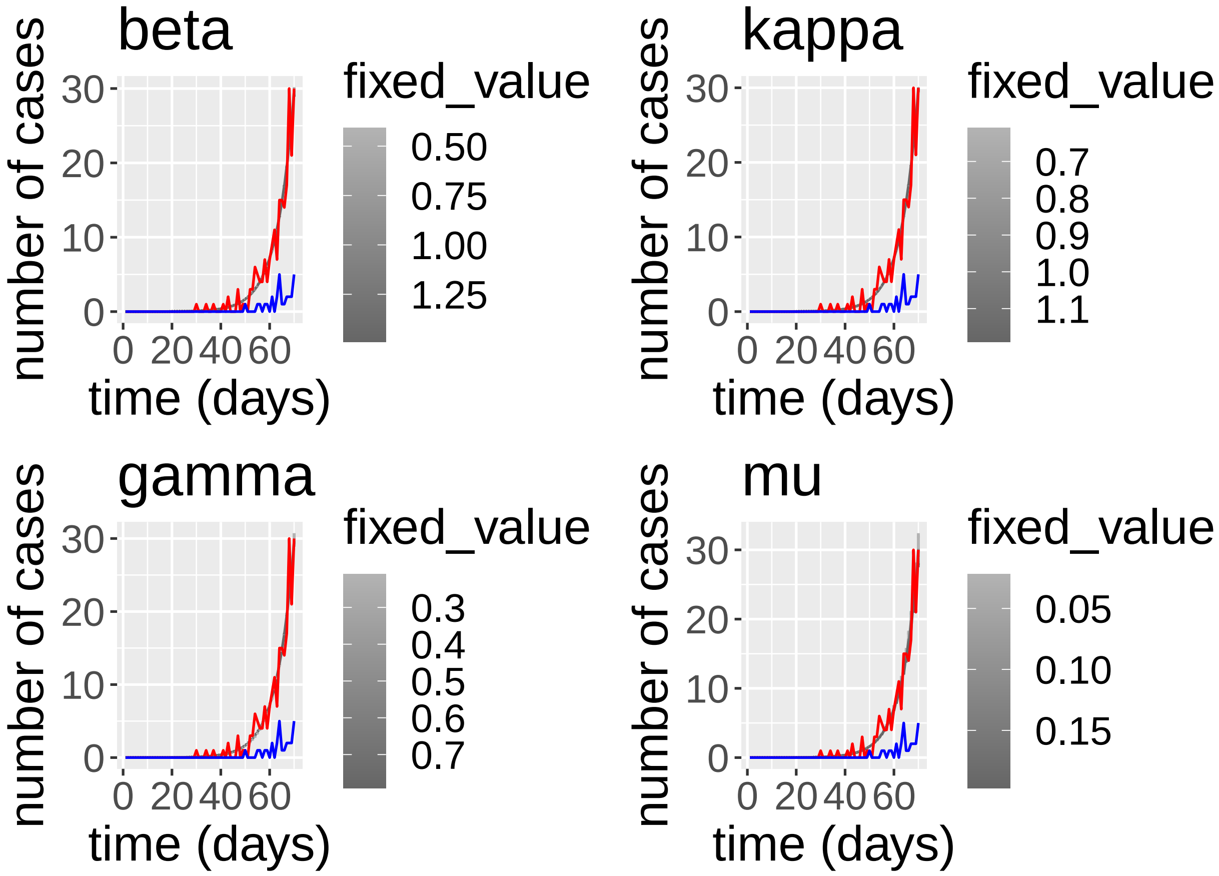
Visualisation

```
# Plot profile likelihood
profilelikelihood_plot(profile_likelihood, profile_parameters)
```



The profile likelihood results show that β, κ and γ are not identifiable, i.e. they have similar log-likelihood values for different fixed values. For example, fixing $\beta = 1.2$ would give a fit of a similar log-likelihood value as fixing $\beta = 0.9$. Therefore, it could not be (uniquely) identified which β value gives the best fit (i.e. the maximum likelihood). In this case, only μ is considered to be identifiable. μ is the death rate, which can be thought as changing other parameters cannot compensate μ, as μ uniquely describes the number of deaths (which is a part of the data).

```r
# Plot fitted trajectory to check optimisation
profilelikelihood_fit(profile_likelihood, profile_parameters, testing_data_short)
```



Given the profile likelihood shows a similar likelihood value, it may be expected that all the model (forward) simulations perform equally well.

Profile likelihood with more information

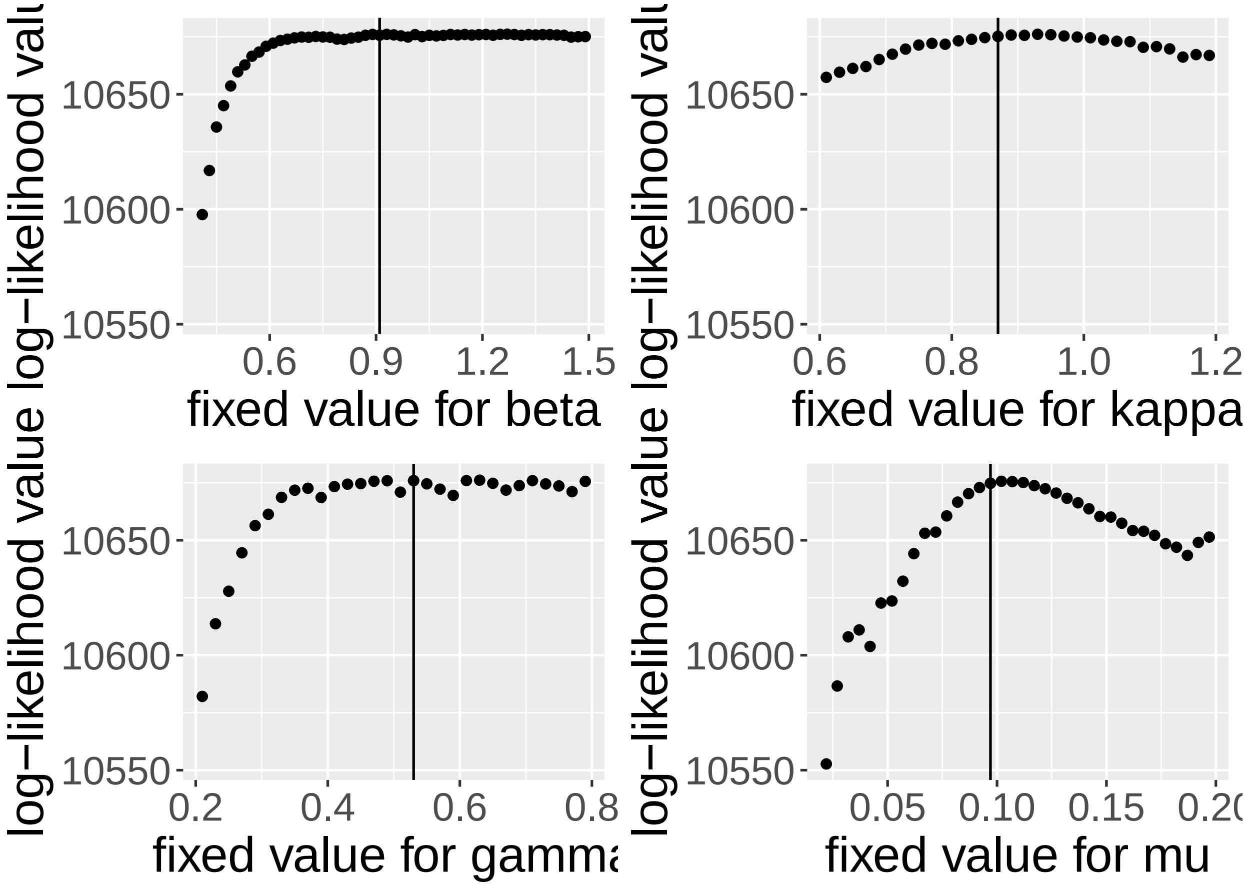
When more information is provided to the optimisation, the identifiability issue could be improved. Here, we show an example by constructing the profile likelihood using synthetic data with double the time points, which covers the first wave of the pandemic.

```r
# Run optimisations to create profile likelihood
profile_likelihood <- profilelikelihood_opt(profile_parameters, range_transmission,
                                            LogLikelihoodFn, testing_data, reltol=1e-4)

# Save results
write.csv(profile_likelihood, "data/synthetic_fulltrend_4param_profilelikelihood.csv",
          row.names = FALSE)
```

```r
# Load results
profile_likelihood <- read.csv("data/synthetic_fulltrend_4param_profilelikelihood.csv")
```

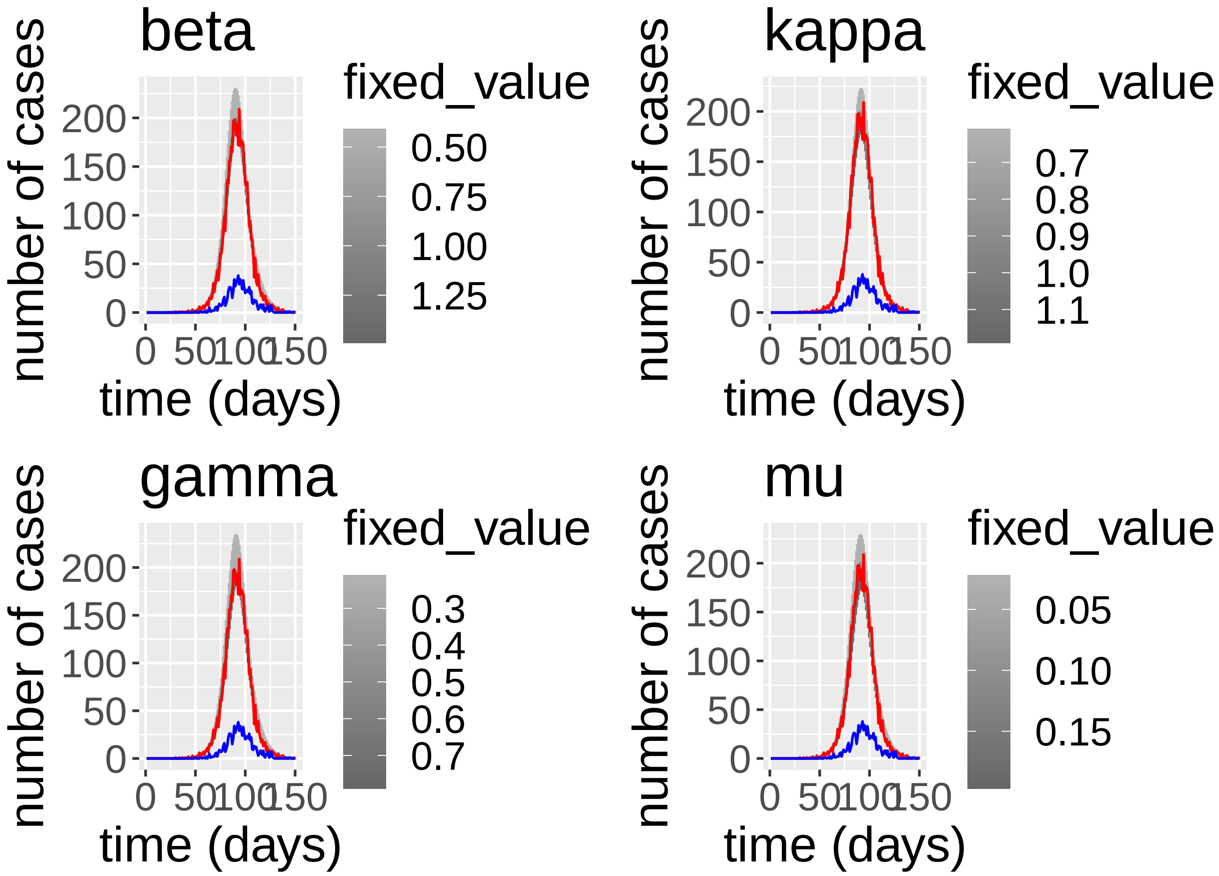
Visualisation

```
# Plot profile likelihood
profilelikelihood_plot(profile_likelihood, profile_parameters)
```



With more data points, μ is still identifiable, observed by a curve with single peak in the profile likelihood. κ is considered to be marginally identifiable, with gradual decrease from the peak of the curve. On the other hand, β and γ are thought to be identifiable up to a certain range; the log-likelihood value for β decreases when β is smaller than 0.6, while the log-likelihood value for γ decreases when γ is smaller than 0.38.

```
# Plot fitted trajectory to check optimisation
profilelikelihood_fit(profile_likelihood, profile_parameters, testing_data)
```



The figure above shows that some of the fits do not perform as well as the others, which may be expected from the profile likelihood.

Conclusion

We showed that there are identifiability issues with the parameters in the SEIRD model by showing that multiple sets of parameters can fit the SEIRD model to the pre-intervention London COVID-19 data. These sets of parameters predicted different scenarios for the COVID-19 incidences and deaths, which is not helpful when planning interventions or preparation for estimated cases. We can further understand this identifiability problem by creating profile likelihood for the parameters. We started constructing the profile likelihood with only 2 transmission parameters, namely β and γ, the infectious rate and the recovery rate. The profile likelihood showed that parameters in this simplified SEIRD model are identifiable. When more parameters are included in the profile likelihood, 3 out of the 4 parameters become unidentifiable. However, the identifiability of the parameters were improved when more data is provided. This concludes, at least for the SEIRD model, that large amount of parameters will give rise to identifiability problem, but extra information can improve the situation. Therefore, we have to practice caution when interpreting optimised parameters from models.

References

[1] Office for National Statistics. (2020) Census Ouput Area population estimates - London, England.