



Fitting Models in R

SPARKLE short course ‘Mathematical Modelling of Infectious Diseases’

Presenter: Rahmat Sagara, Oxford University Clinical Research Unit Indonesia

Summary

A combination of theory and practice to extend the concepts of maximum likelihood and other model fitting methodologies to the context of transmission dynamic models.

This session is divided into four parts:

Part 1: Thinking about parameter fitting - how to measure fit and how to find the best fit

Part 2: Apply measures of distance to our data: least squares and maximum likelihood

Part 3: Parameter estimation techniques with maximum likelihood

Part 4: Confidence intervals derived from likelihood methods

Model fitting

The first exercise in this session is to fit a model “by eye”. We hope that by the end of the exercise you will be thinking to yourself “there must be an easier and more robust way to do this”. This of course motivates the rest of the session. First open Shiny application `Least_Squares_App.Rmd`. (App can be accessed online [here](#)). In this application the dots represent the data and the line represents predicted values based on a mathematical model (in this case a simple cubic equation). This application allows you to explore how changing model parameters changes SSQ (sum of squares of residuals). It is intended to give you an intuitive feel for how a model fitting approach that aims to minimize the SSQs works.

Try to find the best fitting line by adjusting the three parameters with the sliders. Check your answer with the application.

Questions

What are the best SSQs (residuals)?

What disadvantages might there be of fitting models by hand like this?

Can you think of a simple algorithm for searching over all the parameters and finding those that give the best fitting model?

In the example we have been minimizing the sum of squares of the residuals. What would be the effect if we decided to minimize the sums of the absolute values of the residuals? Would you expect to get the same answer?

Distance Function

In this section we will use the sum of squares distance to see how far our model results are from the data. To do so, we will use the model developed in the last session and the initial data in March 2020. To determine our parameters of interest we will minimise our sum of squares and assess model fit to data.

In this practical we are going to estimate some of the parameters of our base SEIR COVID-19 model by fitting it to our incidence data. What are the unknown parameters of our base model?

In this exercise let us assume a known latent period and infectious period and use the basic reproduction number, \mathcal{R}_0 , and the initial size of the infected population, $I(t = 0)$, as the free parameters we will calibrate using our data.

To begin, create a new R script, save the file as `fitting_models.R` into your working directory, and copy the code chunk below:

```
# SPARKLE Modelling Short course

#####
## FITTING MODELS IN R ##
#####
if (!require(pacman)) install.packages("pacman")
library(pacman)

pacman::p_load(dplyr, deSolve, ggplot2)
```

Filtering the data

When calibrating each of these parameters, one thing we must be careful of is to account for the introduction of various interventions as these will likely have a significant impact on the parameter values (the reproduction number in particular). Therefore, in the code provided below I first isolate the “pre-intervention” era (which we defined in the last session as the first wave between 7th and 26th March 2020). We will use this truncated dataset to estimate the parameters, \mathcal{R}_0 and `initial_infected_population`. The code below should let you view the data we are using to fit the model.

```
# Data imports and filtering
first_wave <- read.csv("first_wave_COVID.csv", colClasses = c("Date", "numeric", "numeric"))
```

```

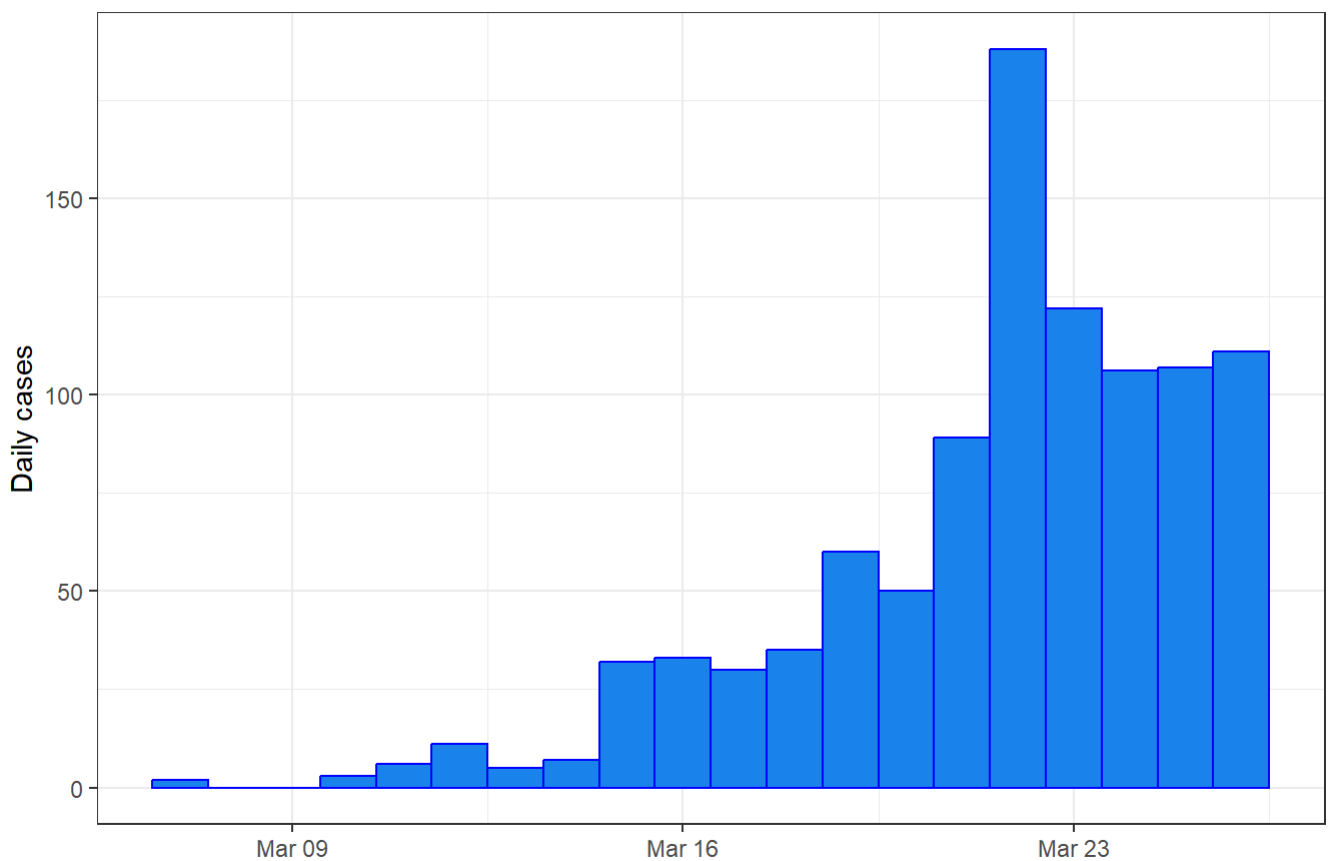
# Time window
start_date <- as.Date("2020-03-07")
end_date   <- as.Date("2020-03-26")

# Filter data to capture period prior to interventions. This has already been done in the last
uncontrolled_period <- first_wave %>%
  filter(Date >= start_date, Date <= end_date)

# Plot the filtered data
ggplot(uncontrolled_period) +
  geom_col(aes(x = Date, y = Cases), width = 1, fill = "dodgerblue2", colour = "blue") +
  ylab("Daily cases") +
  xlab("") +
  ggtitle("Uncontrolled first wave of COVID-19 in X-Country 1st to 24th March, 2020") +
  theme_bw()

```

Uncontrolled first wave of COVID-19 in X-Country 1st to 24th March, 2020



We'll re-use the functions from the last session

```

## Define model as per the previous session
# Time window
times <- seq(start_date, end_date, by = 1)

# Model parameters
parameters <- c(
  R0 = 4,

```

```

    latent_period = 5,
    infectious_period = 6
  )

# Initial conditions
Total_population <- 6.6e7 # Population of X-Country
Initial_exposed <- 0
Initial_infectious <- 20 # Initial infectious seed
Initial_recovered <- 0
Initial_susceptible <- Total_population - Initial_exposed - Initial_infectious - Initial_recovered

# Compartments
state <- c(
  Susceptible = Initial_susceptible,
  Exposed     = Initial_exposed,
  Infectious  = Initial_infectious,
  Recovered   = Initial_recovered
)

# Model function
COVID.base <- function(t, state, parameters) {
  with(as.list(c(state, parameters)), {

    # Calculate the total population size
    Total_population <- Susceptible + Exposed + Infectious + Recovered

    # Calculate the average force of infection imposed on each susceptible individual
    force_of_infection <- R0 * Infectious / (Total_population * infectious_period)

    # Calculate the net (instantaneous) change in each state variable
    Susceptible_change <- -force_of_infection * Susceptible
    Exposed_change     <- force_of_infection * Susceptible - Exposed / latent_period
    Infectious_change  <- Exposed / latent_period - Infectious / infectious_period
    Recovered_change   <- Infectious / infectious_period

    # Return net changes as list
    return(list(
      c(
        Susceptible_change,
        Exposed_change,
        Infectious_change,
        Recovered_change
      )
    ))
  })
}

solve.base.model <- function(y_ = state,
                             times_ = times,
                             func. = COVID.base,
                             parms_ = parameters) {

  out <- ode(
    y = y_,
    times = as.numeric(times_ - times_[1]),

```

```

    func = func.,
    parms = parms_
  )

# Calculate the prevalence, incidence and cumulative incidence (for comparison with data)
out <- as.data.frame(out) %>%
  mutate(
    Prevalence      = Exposed + Infectious,
    Incidence       = Exposed * (1/parms_["latent_period"]),
    Cumulative_incidence = cumsum(Incidence) + Incidence[1],
    Population      = Susceptible + Exposed + Infectious + Recovered,
    #Prevalence     = (Exposed + Infectious)/Population,
    Date = times_
  )

return(out)
}

```

Solving the model for a specific set of parameters

Now we use the model developed in the last session and solve the model for an initial set of parameters, an initial state and using dates from 7th - 26th March, 2020

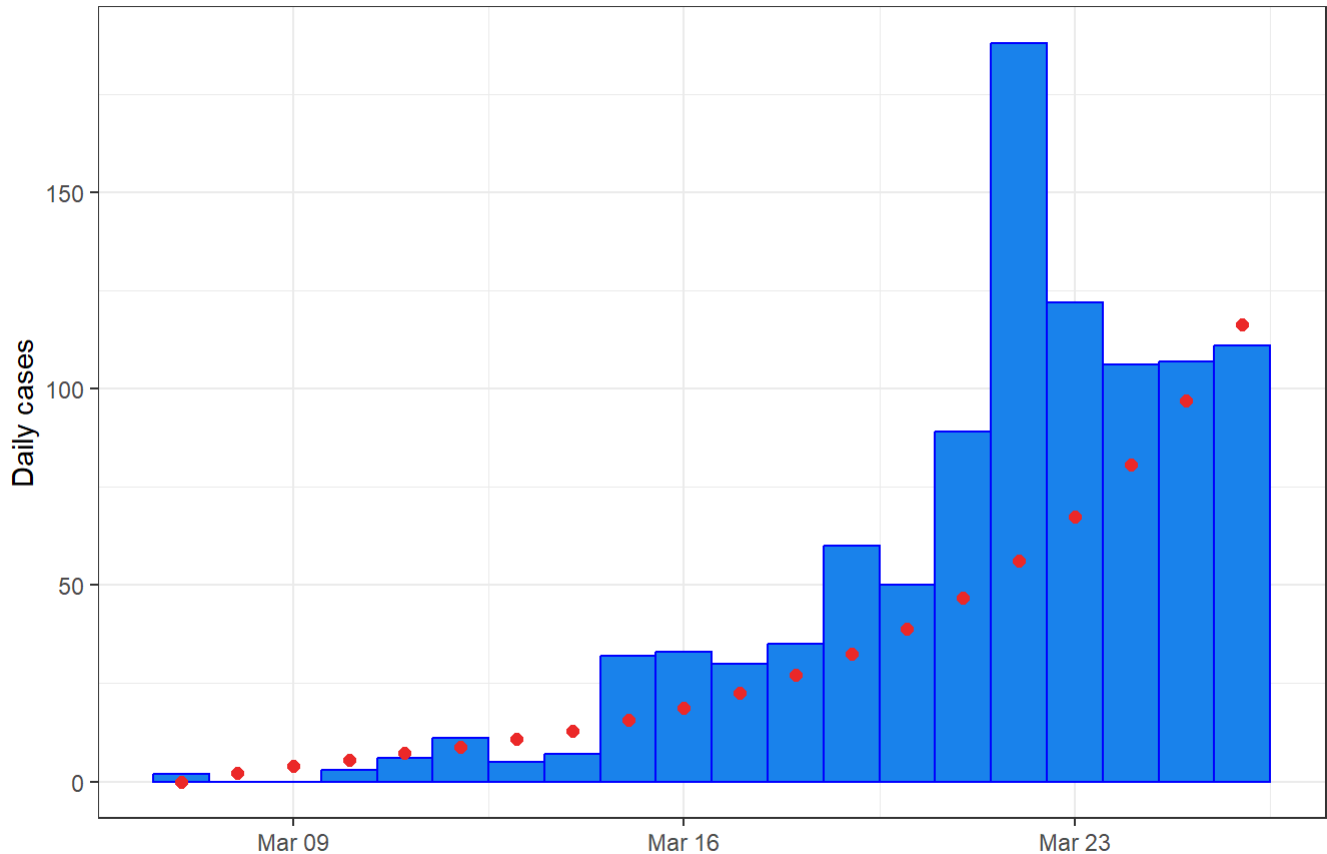
```

# now we use the model developed in the last session and solve it
# for an initial set of parameters, an initial state and using times
# from just 7th - 26th of March 2020
out_init <- solve.base.model(
  y_ = state,
  times_ = times,
  func. = COVID.base,
  parms_ = parameters
)

# Plot the filtered data and the first model "guess"
ggplot(uncontrolled_period) +
  geom_col(aes(x = Date, y = Cases), width = 1, fill = "dodgerblue2", colour = "blue") +
  geom_point(data = out_init, aes(x = Date, y = Incidence), size = 2, colour = "firebrick2") +
  ylab("Daily cases") +
  xlab("") +
  ggtitle("X-Country's First Wave, Jan-Jul 2020") +
  theme_bw()

```

X-Country's First Wave, Jan-Jul 2020



Calculating “distance” between the model and data

We can use the model output, stored in `out_init` to assess model fit. We can use sum of squares of the difference between the data and the model.

```
# find the sum of the residuals squared for day 2 until day 24
SSQ_initial <- sum((uncontrolled_period$Cases[-1] - out_init$Incidence[-1])^2)
SSQ_initial
```

```
[1] 24516.09
```

If we were to minimise `SSQ_initial`, given our parameters \mathcal{R}_0 and $I(t=0)$ we can argue that we have found our parameters of interest.

Transforming the parameters

Before launching into a minimisation of our SSQ function across all parameter space for the values of \mathcal{R}_0 and $I(t=0)$, we should pause and plan our search carefully.

We will use the `optim()` function to minimize our function, but we should be aware of the fact that this function searches over all conceivable values of the parameters (i.e., from $-\infty$ to ∞).

What are the feasible ranges of values for our two parameters: \mathcal{R}_0 and $I(t = 0)$?

With this in mind, it would be sensible to restrict our search to only those values of parameters that are feasible. Not only should this increase the speed of the algorithm (hopefully leading to the desired answer more quickly) but may also help us avoid any computational errors that might arise which will kill our search prematurely. For this purpose, we introduce a set of transformed parameters, that is parameters that exist in the full interval $-\infty$ to ∞ , but can be mapped back to our feasible regions for our parameters.

Transforming variables

`optim()` is going to pass parameter values into our function on the interval $(-\infty, \infty)$. Is there a function that will take those parameter values and put them on the interval $[0, \infty)$?

If $x \in (-\infty, \infty)$, then $\exp(x) \in [0, \infty)$. Therefore if we know R_0 and $I(t = 0)$, we would use `log` to transform back into the `optim()` space. If we want the opposite we can use `exp`.

```
# transform the parameters
# The search / optimization algorithm we employ searches over
# the range (-Inf, Inf) for each variable. In our application
# we are only interested in solutions in the range (0, Inf)
# for R0 and I(0). Therefore, it
# would be a good idea to apply a transformation to our
# search space so that we are not wasting time exploring
# infeasible regions of parameter space.
initial_parameters <- c(
  "R0" = 4,
  "Initial_infectious" = 20
)
```

So now we are in a position to define our function that we wish to optimise. The code is below:

```
#### Fitting routine ####

# We have selected R0 and I(0) as our free parameters
# We need a function that accepts R0 and I(0) as arguments,
# solves the model using these inputs, and calculates the
# sum of squares of the data given these parameters

# Sum of Square Function
SSQ_function <- function(new_parameters,
                          data = uncontrolled_period$Cases[-1],
                          state_base = state,
                          times_ = times,
                          func. = COVID.base,
                          parms_base = parameters) {

  R0 <- new_parameters["R0"]
  Initial_infectious <- new_parameters["Initial_infectious"]

  # Calculate updated susceptible population
```

```

Initial_susceptible <- state_base["Susceptible"] + state_base["Infectious"] - Initial_infectious

# Overwrite baseline parameters with proposed parameters
parms_base["R0"] <- R0
state_base["Susceptible"] <- Initial_susceptible
state_base["Infectious"] <- Initial_infectious

# Solve model with updated parameters
out <- solve.base.model(
  state_base,
  times_,
  func.,
  parms_base
)
return(
  sum((uncontrolled_period$Cases[-1] - out$Incidence[-1])^2)
)
}

```

We can check it works as intended:

```

SSQ_initial_from_function <- SSQ_function(initial_parameters)
SSQ_initial_from_function

```

```
[1] 24516.09
```

Yay, we are getting the same result.

With our `SSQ_function` now set up, we can use the `optim` function to determine the optimal parameters, that is the parameter values that minimize the sum of squares.

We will use the Nelder-Mead search algorithm. This is a search algorithm that allows rapid exploration of the parameter space for the optimal parameters.

```

#### Optimal parameters ####

# Use the optim function to determine the parameters that
# minimize the negative log-likelihood (i.e., maximize
# the likelihood)
# We will use the Nelder-Mead optimization solver
optim_NM <- optim(
  par      = initial_parameters,
  fn       = SSQ_function,
  control  = list(maxit = 500),
  method   = "L-BFGS-B",
  hessian  = TRUE
)

```

Once the search is complete, inspect the solution:

```
# Check for convergence (always code 0 for NM)
optim_NM$convergence # 0 - converged; 1 - failed to converge
```

```
[1] 0
```

```
# Inspect solution
optim_NM$par
```

```
      R0 Initial_infectious
2.986699      73.852109
```

What are the initial values of the two fitted parameters?

But have we improved our SSQ from the initial guess we had?

```
# Inspect the optimal sum of squared residuals
optimum_SSQ <- optim_NM$value
optimum_SSQ
```

```
[1] 15100
```

How does the SSQ score at the optimal value compare with our initial SSQ score (`SSQ_initial`)?

Satisfied that we have found the optimal solution, let's now solve the model using these parameter values and compare the solution with our observed data:

```
#### Optimal model solution ####

# Create the optimal parameter and state vectors
optimum_parameters <- optim_NM$par
optimum_parameters
```

```
      R0 Initial_infectious
2.986699      73.852109
```

```
optimal_parameters <- parameters
optimal_parameters["R0"] <- optimum_parameters["R0"]

optimal_initial_state <- state
optimal_initial_state["Susceptible"] <- state["Susceptible"] + state["Infectious"] - optimum_p
optimal_initial_state["Infectious"] <- optimum_parameters["Initial_infectious"]

# Solve the model given the optimal parameters and initial conditions
optimal_solution <- solve.base.model(
  y_ = optimal_initial_state,
  times_ = times,
  func. = COVID.base,
```

```

  parms = optimal_parameters
)

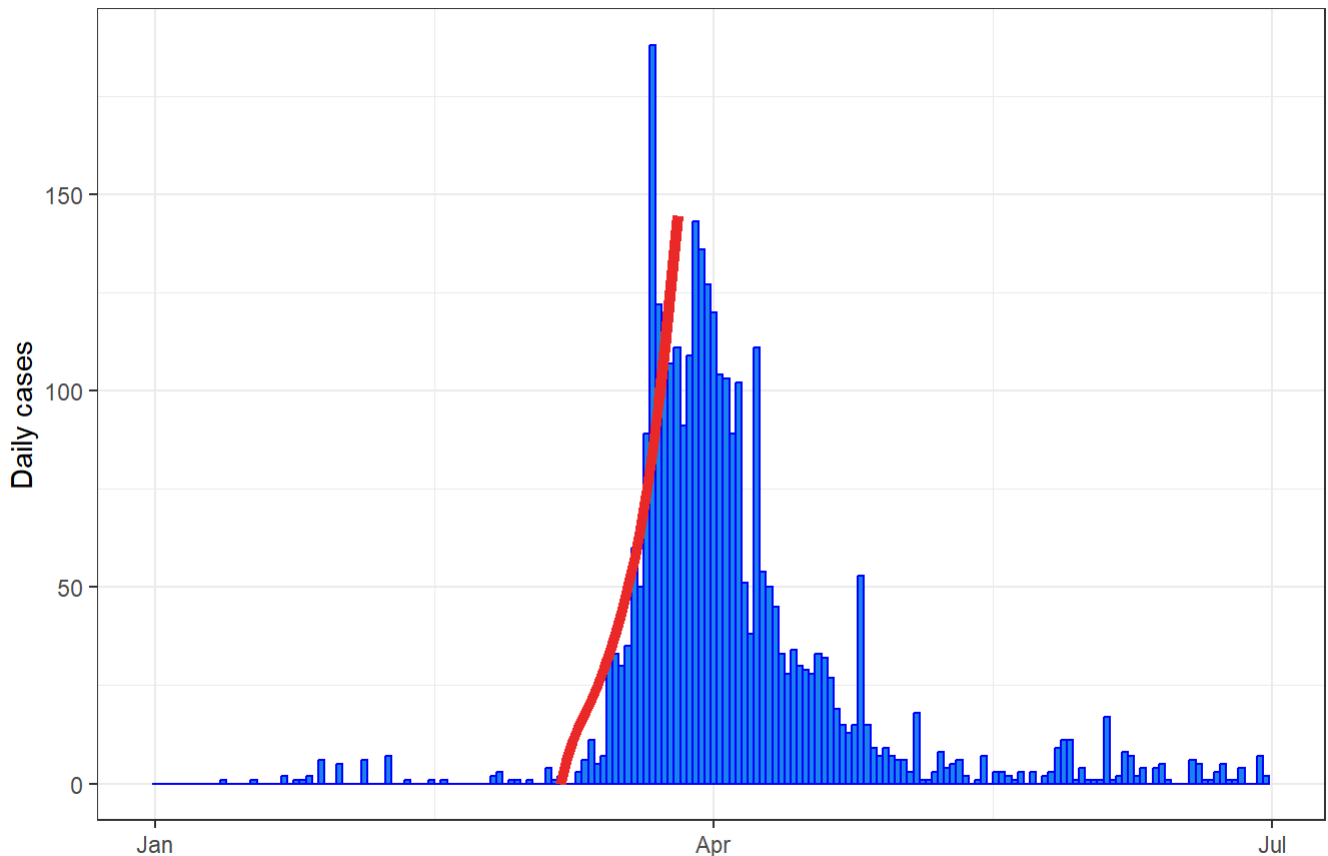
# Plot the optimal solution
ggplot(first_wave) +
  geom_col(aes(x = Date, y = Cases), width = 1, fill = "dodgerblue2", colour = "blue") +
  geom_line(data = optimal_solution, aes(x = Date, y = Incidence), size = 2, colour = "firebrick",
  ylab("Daily cases") +
  xlab("") +
  ggtitle("Fit to unmitigated period") +
  theme_bw()

```

Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.

! Please use `linewidth` instead.

Fit to unmitigated period

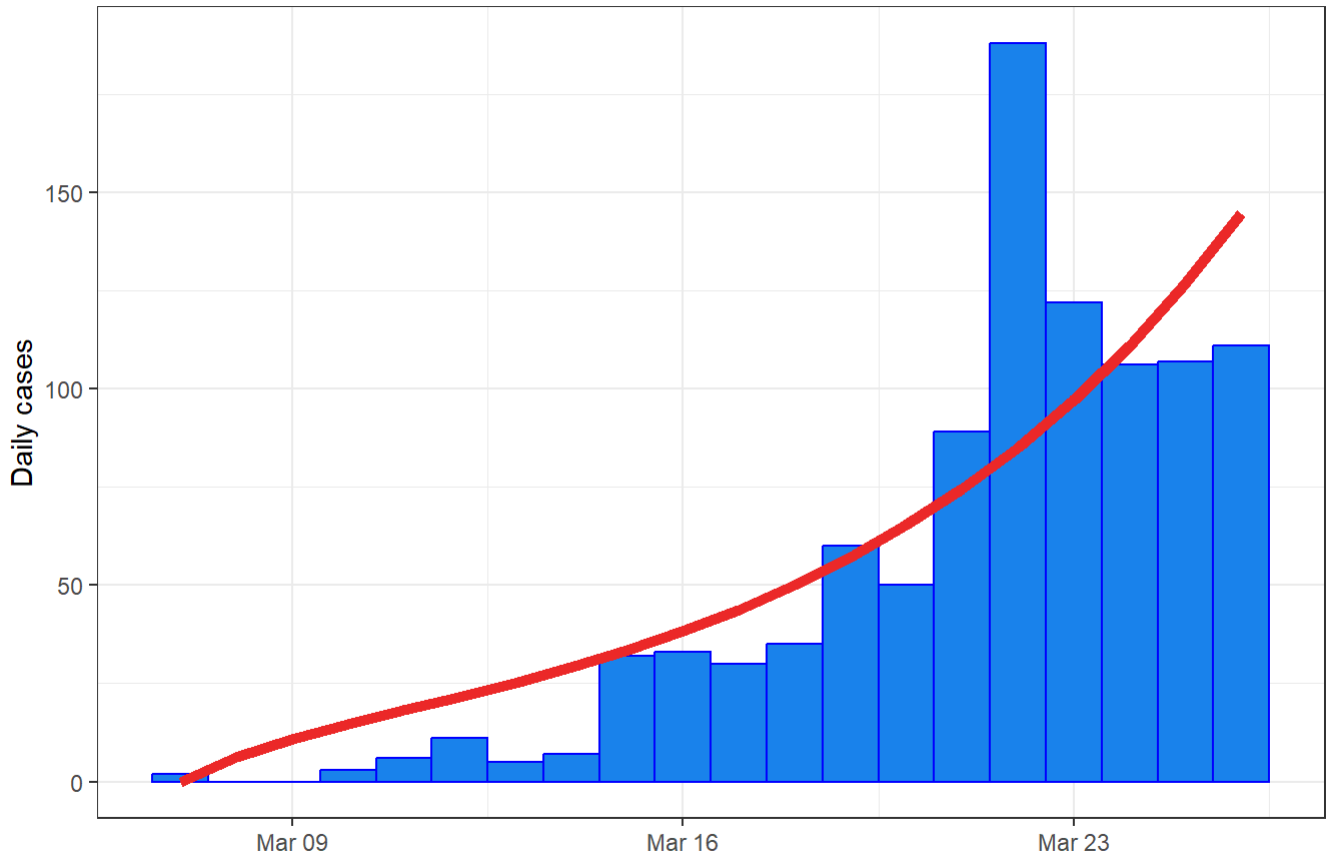


```

ggplot(uncontrolled_period) +
  geom_col(aes(x = Date, y = Cases), width = 1, fill = "dodgerblue2", colour = "blue") +
  geom_line(data = optimal_solution, aes(x = Date, y = Incidence), size = 2, colour = "firebrick",
  ylab("Daily cases") +
  xlab("") +
  ggtitle("Fit to unmitigated period") +
  theme_bw()

```

Fit to unmitigated period



How does this fit compare to the observed data?

Likelihood Functions

In the previous section you learnt of one way to measure how far the model results are from data. When you minimise SSQ, you are performing a [Least Squares Fit](#). This is drawing the line of best fit through all of your data points (obviously you might minimise your SSQ further by changing your choice of line, but then we run into overfitting problems).

In the literature, a much more common way to fit parameters is to find the [maximum likelihood](#). But what is the likelihood?

Definition: Likelihood function

The likelihood is a function that describes how **likely** your observed data is given your model and its parameters.

Why would we want to use the likelihood over SSQ?

You might have noticed that the model fit in the last section was not... very good. It appears to just go through the middle of the data and not describe any one point very well. This is because our model wanted to minimise the sum of squared error! In reality, it makes more sense to consider that each data point is an observation around which there is uncertainty that can be represented by a statistical distribution.

So now the question is *how do we determine what our likelihood function should be?*

This is not easy to answer in general, but what we can teach is easy ways to construct likelihood functions.

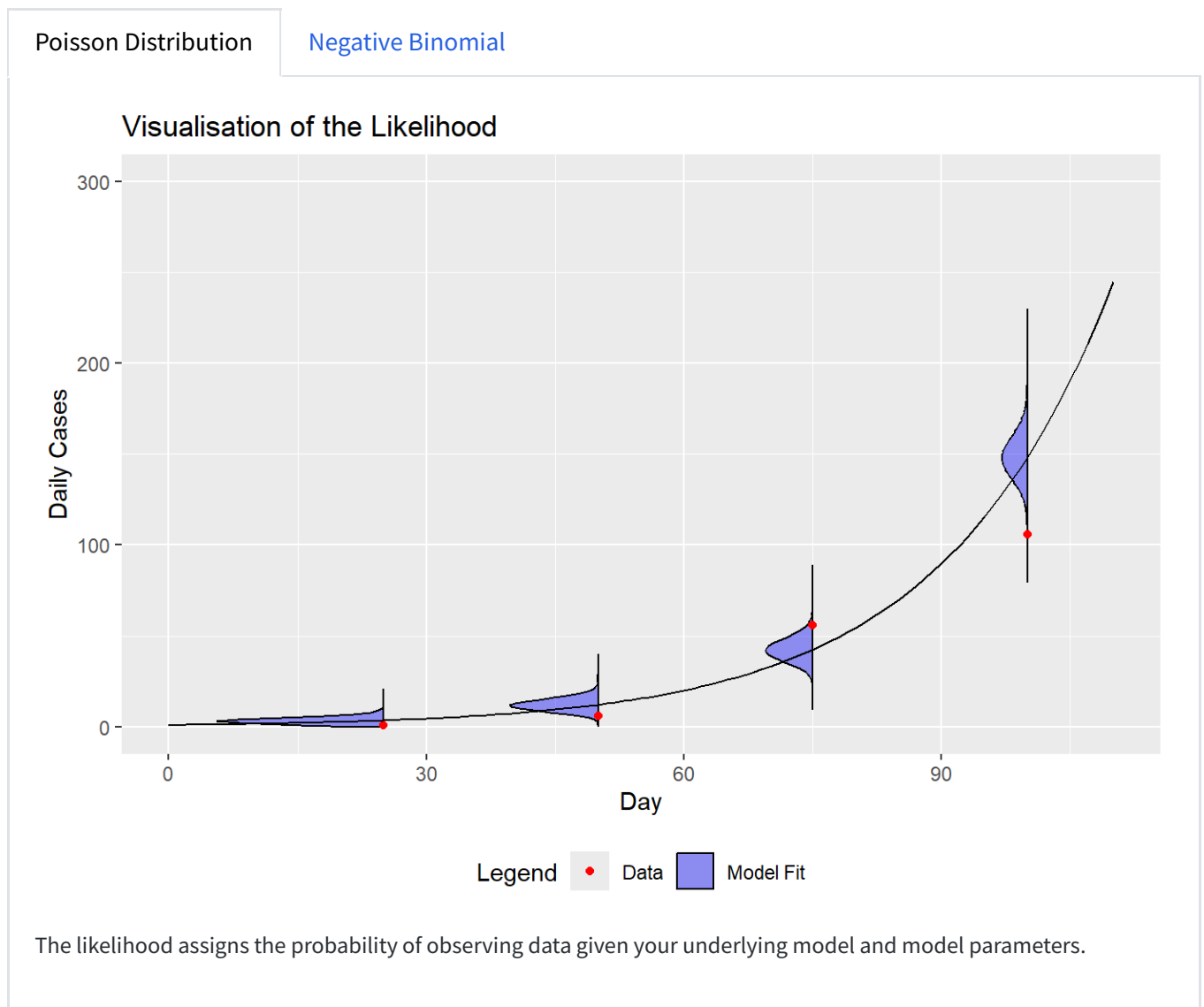
As the likelihood is the probability of data given your parameters, it is typically defined as,

$$p(\vec{D}|\vec{\theta})$$

where \vec{D} is the vector of all data points and $\vec{\theta}$ is a vector containing the model parameters. In words, we could say that the probability of observing our data given our parameters is the same as observing the first data point and the second data point and the third data point... so on given our parameters. So let us express our likelihood as,

$$p(\vec{D}|\vec{\theta}) = \prod_{i=1}^N p(D_i|\vec{\theta}),$$

where D_i is the i th data point in the vector of data. Let us look at the figure below for a visualisation of the likelihood.



When using the likelihood, it is much more common for people to work with log-likelihood. The reason for this, is that with a large number of data points you very quickly run out of computational precision (each probability will be less than one and computers cannot store super small numbers well). From now on we will work with a

log-likelihood, which means all products will become sums. That is,

$$\log p(\vec{D}|\vec{\theta}) = \sum_{i=1}^N \log p(D_i|\vec{\theta}),$$

Covid-19 Likelihood function

Let us calculate the *likelihood* of our observed data given our COVID model (which of course is specific to the current parameter values that we have used). To do this, a common practice is to assume that the observed daily incidence (stored in the dataframe `uncontrolled_period`) is a Poisson-distributed random variable with mean given by our model-predicted incidence (stored in the dataframe `out_init`). Under this model the (log-)likelihood of each daily observation can be calculated using the `dpois()` function; and the total (log-)likelihood for all of the observations is simply the sum of all these:

$$\log p(\text{Cases}|\vec{\theta}) = \sum_{i=1}^N \log \text{Poisson}(c_i|\mu = \text{Model}(\theta, i)),$$

where `Cases` is the vector containing our case data, c_i is the number of cases on the i th day, `Model(θ, i)` is the model output given our parameters on day i , and μ is the mean of the Poisson distribution.

There are many possible distributions that can be used to develop the likelihood. You may wish to investigate some of the density distributions available in R, which you can do by typing `?distributions` in your console.

Another distribution, which allows us to account for high variability in the data is the negative binomial distribution. We could use the output from our last function call to calculate the likelihood of our observed data given our COVID model using `dnbinom()` function in much the same way as the Poisson distribution. Those with a particular interest in statistics and probability distributions may wish to experiment and to consider the pros and cons of different choices. For the remainder of this exercise, however, we will continue with the Poisson distribution which has the attribute of having only one parameter to estimate, the mean.

Changing distributions

The decision on what distribution to use within your likelihood function is a *Model Choice*. Sometimes, multiple distributions may suit your need. Be aware that each distribution has different assumptions.

Parameter estimation with maximum likelihood

With our likelihood function defined mathematically, let us now start thinking about how to set it up in R. We again want to use the `optim` function to determine the optimal parameters, however, `optim()` will solve for the minimum as opposed to the maximum (we want the maximum likelihood). Because of this, we shall put a minus sign in front of our likelihood, that way finding the minimum of the new function will actually calculate the maximum that we want!

So now we are equipped to fit our likelihood function, using code like below:

```
#### Fitting routine ####

# We have selected R0 and I(θ) as our free parameters
# We need a function that accepts R0 and I(θ) as arguments,
# solves the model using these inputs, and calculates the
```

```

# negative log-likelihood of the data given these parameters

negative_log_likelihood <- function(new_parameters,
                                   data = uncontrolled_period$Cases[-1],
                                   state_base = state,
                                   times_ = times,
                                   func. = COVID.base,
                                   parms_base = parameters) {

  # parameters
  R0 <- new_parameters["R0"]
  Initial_infectious <- new_parameters["Initial_infectious"]

  # Calculate updated susceptible population
  Initial_susceptible <- state_base["Susceptible"] + state_base["Infectious"] - Initial_infectious

  # Overwrite baseline parameters with proposed parameters
  parms_base["R0"] <- R0
  state_base["Susceptible"] <- Initial_susceptible
  state_base["Infectious"] <- Initial_infectious

  # Solve model with updated parameters
  out <- solve.base.model(
    state_base,
    times_,
    func.,
    parms_base
  )

  return(-sum(dpois(
    x = data,
    lambda = out$Incidence[-1],
    log = TRUE
  )))
}

```

Let us now calculate our optimal parameters!

```

#### Optimal parameters ####

# Use the optim function to determine the parameters that
# minimize the negative log-likelihood (i.e., maximize
# the likelihood)
# We will use the Nelder-Mead optimization solver
optim_NM <- optim(
  par = initial_parameters,
  fn = negative_log_likelihood,
  control = list(maxit = 500),
  method = "L-BFGS-B",
  lower = 0,

```

```
hessian = TRUE
)
```

Once the search is complete, inspect the solution:

```
# Check for convergence (always code 0 for NM)
optim_NM$convergence # 0 - converged; 1 - failed to converge
```

```
[1] 0
```

```
# Inspect solution
optim_NM$par
```

```
      R0 Initial_infectious
3.970841      30.352094
```

What are the initial values of the two fitted parameters?

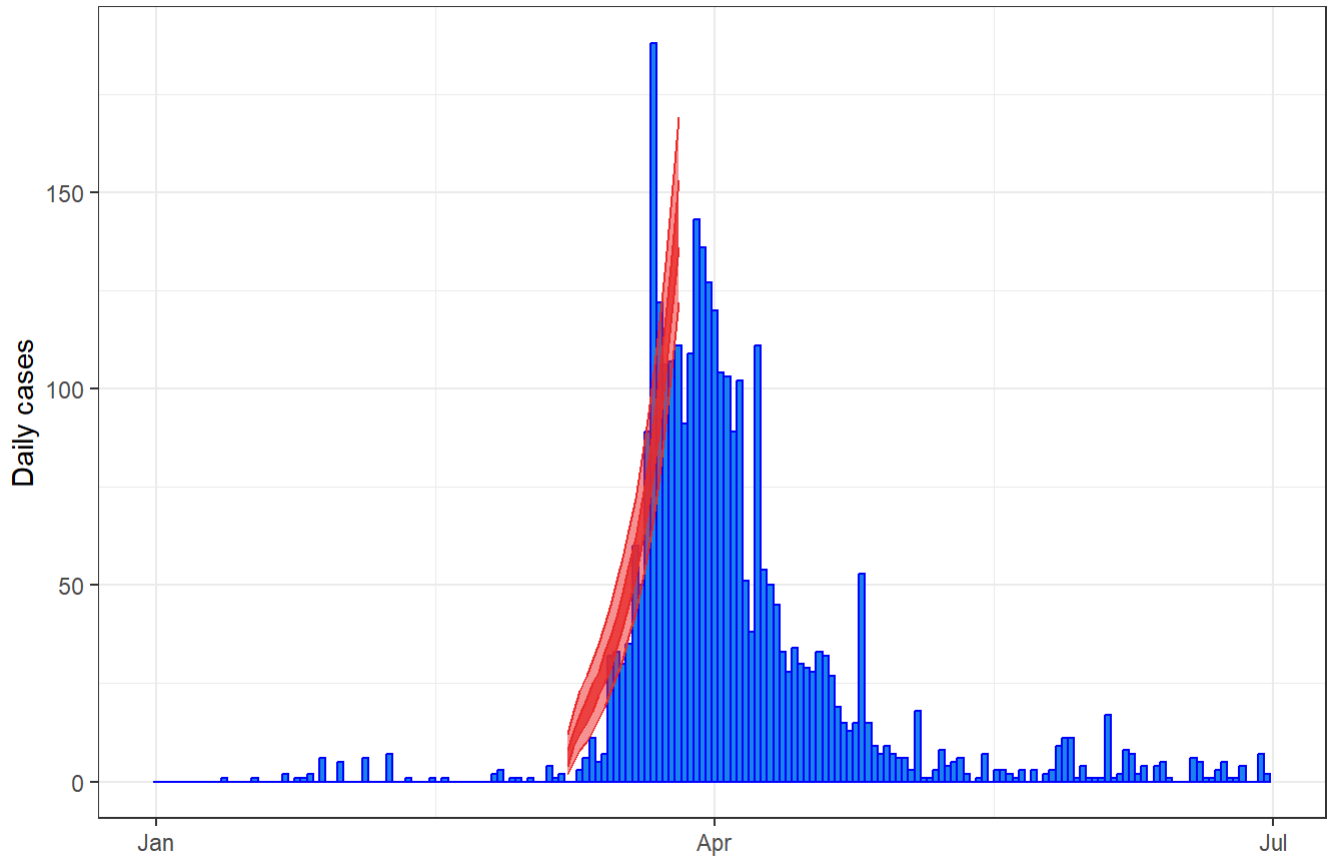
```
optimum_parameters <- optim_NM$par
optimum_parameters
```

```
      R0 Initial_infectious
3.970841      30.352094
```

```
# Calculate the observational confidence intervals
optimal_solution <- optimal_solution %>%
  mutate(
    lower50 = qpois(p = 0.25, lambda = Incidence), # 50% confidence interval (i.e., 25 - 75th
    upper50 = qpois(p = 0.75, lambda = Incidence),
    lower95 = qpois(p = 0.025, lambda = Incidence), # 95% confidence interval (i.e., 2.5 - 97.
    upper95 = qpois(p = 0.975, lambda = Incidence)
  )

# Plot confidence intervals as ribbons around the central estimates
ggplot(first_wave) +
  geom_col(aes(x = Date, y = Cases), width = 1.0, fill = "dodgerblue2", colour = "blue") +
  geom_ribbon(data = optimal_solution[-1, ], aes(x = Date, ymin = lower50, ymax = upper50), fi
  geom_ribbon(data = optimal_solution[-1, ], aes(x = Date, ymin = lower95, ymax = upper95), fi
  ylab("Daily cases") +
  xlab("") +
  ggtitle("Fit to unmitigated period (with observational uncertainty)") +
  theme_bw()
```

Fit to unmitigated period (with observational uncertainty)



Are there any differences between these results and the fitted parameters in the previous section?

Why might this be?

If we chose a different distribution, could we get a different answer?

Bonus code

Copy and paste this code to perform a grid search - getting the negative log likelihood across a sequence of each parameter value.

```
# Specify search grid
R0_vec <- seq(from = 0.1, to = 10, by = 0.1)
initial_infectious_vec <- seq(from = 1, to = 50, by = 1)

# the function that will perform grid search

nll.grid <- function(R0_vec, initial_infectious_vec) {
  parameter_grid <- expand.grid(
    R0 = R0_vec,
```

```

    initial_infectious = initial_infectious_vec
  )
  nll <- data.frame(matrix(nrow = nrow(parameter_grid), ncol = 1))
  colnames(nll) <- "NLL"
  for (row_of_params in seq(1, dim(parameter_grid)[1])) {
    transformed_params <- log(c(
      "R0" = parameter_grid[row_of_params, 1],
      "Initial_infectious" = parameter_grid[row_of_params, 2]
    ))
    nll[row_of_params, ] <- negative_log_likelihood(transformed_params)
  }
  return(data.frame(
    Reproduction_number = parameter_grid["R0"],
    Initial_infectious = parameter_grid["initial_infectious"],
    nll
  ))
}

# call the function that will calculate likelihood surface
nll_grid_results <- nll.grid(R0_vec, initial_infectious_vec)

# Plot likelihood surface
nll_plot <- nll_grid_results %>%
  ggplot2::ggplot(aes(
    x = R0,
    y = initial_infectious,
    fill = NLL,
    z = NLL
  )) +
  geom_tile() +
  geom_contour(breaks = c(100, 110, 120, 130, 200, 500, 1000, 4000)) +
  ylab("I(0)") +
  xlab("R0") +
  ggtitle("Likelihood surface") +
  theme_bw()

# Extract optimum from grid search
grid_optimum <- nll_grid_results[which.min(nll_grid_results$NLL), ]
grid_optimum

```

	R0	initial_infectious	NLL
2940	4	30	144.9431

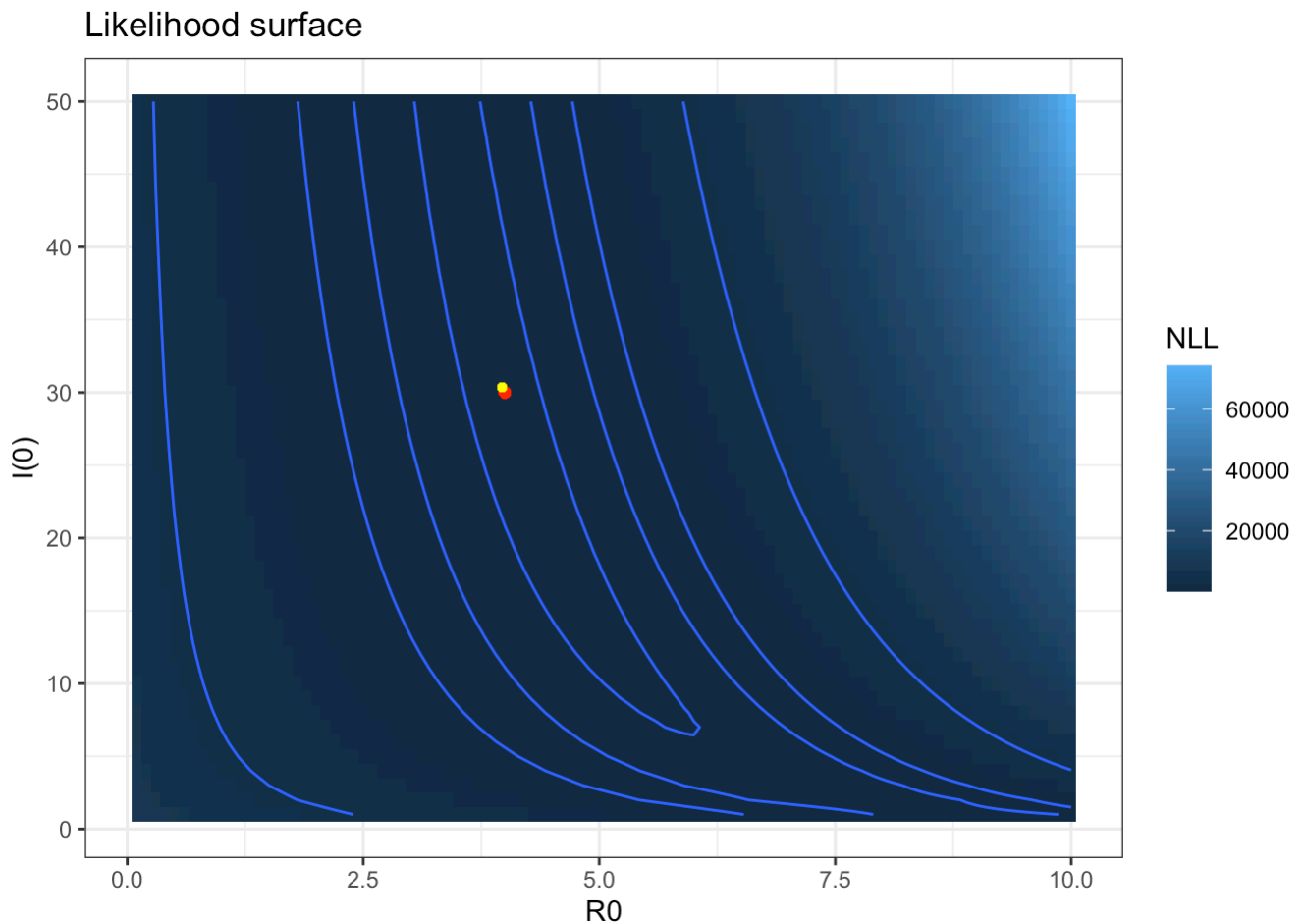
```

# Superimpose optimum on likelihood surface plot
nll_plot <- nll_plot +
  geom_point(aes(
    x = grid_optimum$R0,
    y = grid_optimum$initial_infectious
  ),
  colour = "red",
  )
# Superimpose optimal point to negative-log-likelihood plot

```

```
nll_plot <- nll_plot + geom_point(aes(
  x = optimum_parameters["R0"],
  y = optimum_parameters["Initial_infectious"]
),
colour = "yellow",
size = 1
)

print(nll_plot)
```



Why might a grid search not be the best option for finding optimum parameter values ?

Calculating confidence intervals

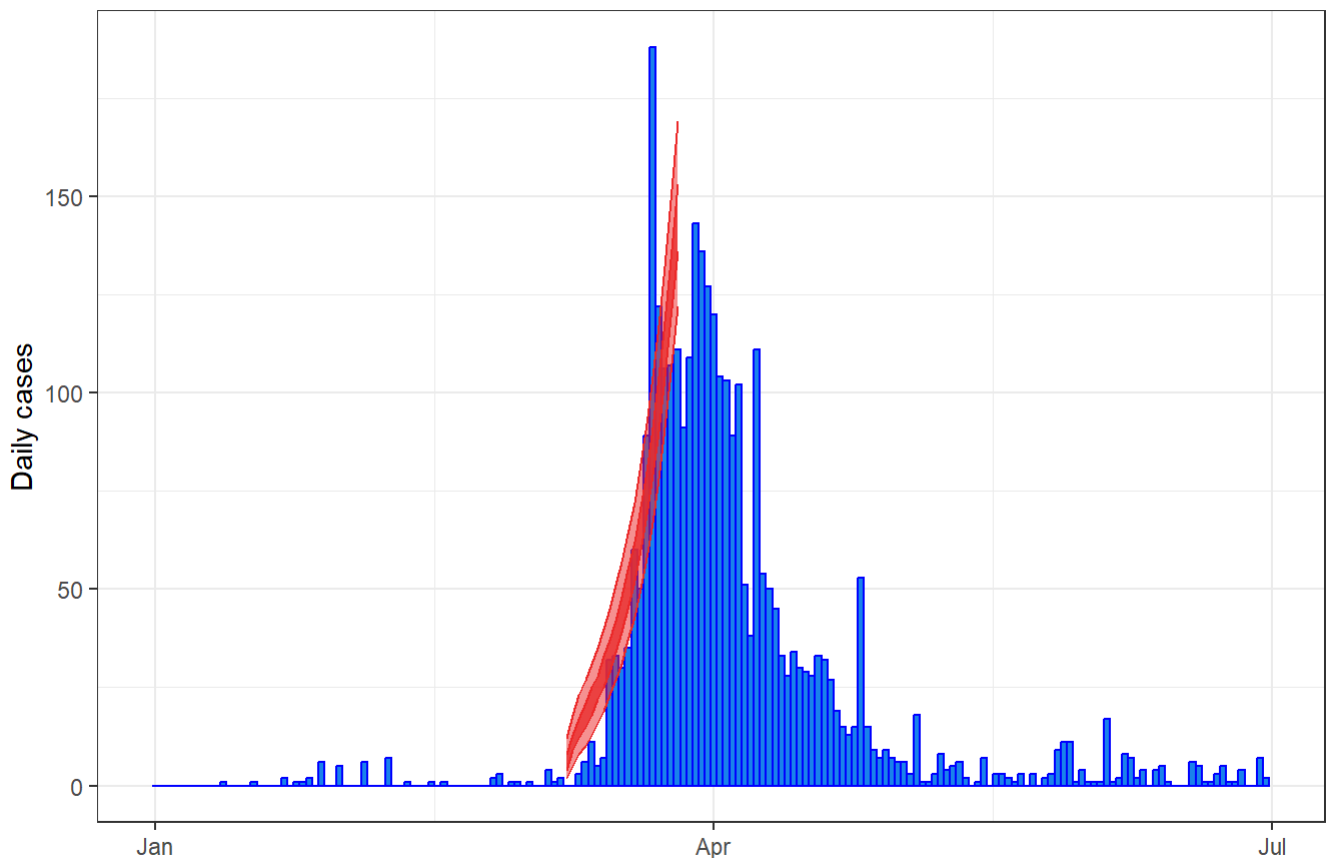
We are now going to put some confidence intervals around the data. This is commonly done but has some problems that relate to model assumptions. For example, if we use the Poisson distribution for estimating parameters, it assumes the spread of the data has a variance equal to the mean. The Negative Binomial distribution allows for greater data variance, but we have one extra parameter to estimate - the variance of the negative binomial distribution itself. Below is some code to examine the confidence intervals around your data with the caveats given above.

Firstly, calculate the Poisson distribution centred on the optimised model solution. Find the inter-quartile range and 95% confidence interval of this distribution.

```
# Calculate the observational confidence intervals
optimal_solution <- optimal_solution %>%
  mutate(
    lower50 = qpois(p = 0.25, lambda = Incidence), # 50% confidence interval (i.e., 25 - 7
    upper50 = qpois(p = 0.75, lambda = Incidence),
    lower95 = qpois(p = 0.025, lambda = Incidence), # 95% confidence interval (i.e., 2.5 -
    upper95 = qpois(p = 0.975, lambda = Incidence)
  )

# Plot confidence intervals as ribbons around the central estimates
ggplot(first_wave) +
  geom_col(aes(x = Date, y = Cases), width = 1.0, fill = "dodgerblue2", colour = "blue") +
  geom_ribbon(data = optimal_solution[-1, ], aes(x = Date, ymin = lower50, ymax = upper50),
  geom_ribbon(data = optimal_solution[-1, ], aes(x = Date, ymin = lower95, ymax = upper95),
  ylab("Daily cases") +
  xlab("") +
  ggtitle("Fit to unmitigated period (with observational uncertainty)") +
  theme_bw()
```

Fit to unmitigated period (with observational uncertainty)



More important than confidence intervals around the data are confidence intervals around the estimated parameters. We found an optimum value for the estimated parameters using the R inbuilt optim function, but this function can do much more.

In addition to the point estimates for each of the parameters, we can also use the results of the optimization search to calculate confidence intervals for them too. The idea is to use the curvature of the likelihood surface to determine the range of parameter values that exceed a particular likelihood score (pointer likelihood surfaces should lead to more smaller confidence intervals). The curvature values are stored in what is known as the Hessian matrix:

```
optim_NM$hessian
```

	R0	Initial_infectious
R0	669.88206	26.213285
Initial_infectious	26.21329	1.080023

The covariance matrix is the inverse of this matrix, and be calculated using:

```
# Covariance matrix
covar_matrix <- solve(optim_NM$hessian)
covar_matrix
```

	R0	Initial_infectious
R0	0.02971051	-0.7211053
Initial_infectious	-0.72110532	18.4278895

Using the covariance matrix, we can next calculate the standard errors for each parameter by taking the square root of the diagonal elements of the covariance matrix, and then adding and subtracting these to the central estimates to obtain the 95% confidence intervals:

```
# Now use the covariance matrix to generate standard errors for each parameter
std_errors <- sqrt(diag(covar_matrix))
estimates_transformed <- data.frame(
  estimate = optim_NM$par,
  std_error = std_errors
) %>%
  mutate(
    lower95CI = estimate - 1.96 * std_error,
    upper95CI = estimate + 1.96 * std_error
  )
estimates_transformed
```

	estimate	std_error	lower95CI	upper95CI
R0	3.970841	0.1723674	3.633001	4.308681
Initial_infectious	30.352094	4.2927718	21.938261	38.765927

```
# Back-transform estimates and confidence intervals
estimates <- exp(estimates_transformed[, -2])
estimates
```

	estimate	lower95CI	upper95CI
R0	5.302912e+01	3.782617e+01	7.434239e+01
Initial_infectious	1.519662e+13	3.370279e+09	6.852169e+16

Contributors

- Eamon Conway, Walter and Eliza Hall Institute
- Lisa White, Nuffield Department of Medicine, Oxford University
- Michael Meehan and Emma McBryde, Australian Institute of Tropical Health & Medicine, James Cook University