Modelling epidemics with R

Ewen Gallic Michel Lubrano Pierre Michel

5/31/23

Table of contents

Preface

As Michel likes to emphasize, our offices are located in a triangle within the Aix Marseille School of Economics (AMSE) department of Aix Marseille Université, so we couldn't be closer. However, it was not until the lockdown brought about by the COVID-19 pandemic that we actually started working together. The COVID-19 pandemic, despite all the disasters it has brought upon a lot of people, has had at least one positive externality for us –it has prompted remote discussions. We have spent a lot of time exploring the use of entirely new methods for each of us. Through this ebook, we aim to share the knowledge we have gained during this period. There may be errors in the codes provided, so if you spot any, please don't hesitate to point them out to us.

Outline

The first part of this ebook relates to the SIR model. In Chapter 1, the SIR model is presented. In Chapter 2, R codes to simulate an epidemics with the SIR model, with and without lockdown are provided. Chapter 3 shows the dynamics of the epidemics with animated graphs. The second part of this ebook is devoted to statistical models. Chapter 4 presents the Covid-19 data from Oxford University. Chapter 5 shows how to estimate the reproduction number. Chapter 6 presents the phenomenological models and Chapter 7 shows how to use R to estimate those phenomenological models.

Part I

SIR Model

1 Theoretical background

In order to fully understand the logic which is at work behind the general epidemic data and how lockdowns can be justified, it is useful to detail the simplest epidemiological model in use. Epidemiologists have the habit of classifying people in different compartments and to model the transition between the different compartments using differential equations. The SIR model of Kermack and McKendrick (1927) considers a finite and fixed population N which is divided into three exclusive groups summing to N:

- S: Susceptible,
- I: Infected,
- R: Recovered (or Removed).

These three letters giving its name to the model. The strength of the epidemic or the infection rate β determines the passage from group S to group I. The recovery rate γ determines the passage from group I to group R. When an infected person recovers, they become immune to the disease and cannot be reinfected. The system is completely described by three differential equations:

$$\begin{aligned}\frac{\mathrm{d}S}{\mathrm{d}t} &= -\beta I \times \frac{S}{N} \\ \frac{\mathrm{d}I}{\mathrm{d}t} &= -\beta I \times \frac{S}{N} - \gamma I \\ \frac{\mathrm{d}R}{\mathrm{d}t} &= \gamma I\end{aligned} \tag{1.1}$$

The parameter γ is a biological parameter. It measures the rate of recovery when being infected. It is equal to the inverse of the number of days needed to recover, $T_r = 1/\gamma$. With the COVID-19 pandemic, the average number of days to recover in most non-severe cases is between 7 to 14 days (see, *e.g.*, Park et al. (2020)). In Moll (2020), $\gamma = 1/7$ and in H. Wang et al. (2020), $\gamma = 1/18$.

The second parameter, β is related to the contagiousness of the disease. It takes into account the probability of contracting the disease when a susceptible person comes into contact with an infected one. $T_C = 1/beta$ can be thought as the typical time between contacts. The contact rate β is thus fundamentally a social parameter, because it depends on the contact habits (shaking hands or not for instance) as well as the hygiene habits of the population. It can vary a lot between countries and is the main object of inference (see, *e.g.*, Toda (2020)).

1.1 Reproduction Numbers

Since

$$\frac{\mathrm{d}S}{\mathrm{d}t} + \frac{\mathrm{d}I}{\mathrm{d}t} + \frac{\mathrm{d}R}{\mathrm{d}t} = 0,$$

and that by integration we find $S + I + R = N$, N can be seen as an arbitrary integration constant. Consequently, S, I, and R are usually considered to be proportions that add up to 1 with:

$$S + I + R = 1 \tag{1.2}$$

leading to a simpler presentation of the model:

$$\begin{aligned} \frac{\mathrm{d}S}{\mathrm{d}t} &= -\beta I \times S \\ \frac{\mathrm{d}I}{\mathrm{d}t} &= -\beta I \times S - \gamma I \\ \frac{\mathrm{d}R}{\mathrm{d}t} &= \gamma I \end{aligned} \tag{1.3}$$

The basic reproduction number $\mathcal{R}_0$, *i.e.*, the average number that an infected person manages to contaminate during the period of contagion is given by $T_r/T_c = \beta/\gamma$. This number is fixed at the beginning of the epidemic and is its main characteristics. For COVID-19, the first values taken in the model of Imperial College were between 2 and 2.6, later updated to an interval between 2.4 and 3.3 for the UK. In European countries, values as high as between 3 to 4.7 were found as reported in Adam (2020).

Because the epidemic evolves over time and finally stops, it is necessary to introduce a complementary notion, the effective reproduction number defined as:

$$\mathcal{R}_t^e = \frac{\beta}{\gamma} \times S_t = \mathcal{R}_0 \times S_t. \tag{1.4}$$

This effective reproduction number decreases with the number of susceptibles S_t.

- If $\beta > \gamma$ so that $\mathcal{R}_0 > 1$, then the epidemic grows exponentially.
- If $\beta < \gamma$ so that $\mathcal{R}_0 < 1$, then the epidemic dies out exponentially.

The major goal of a health policy is to obtain a $\mathcal{R}_0$ lower than 1.0, using a lockdown policy that will lead to a decrease in the value of β.

The model assumes that when a person has been infected, they recover (or die), but can never be re-infected. Because of the conservation identity Equation 1.2, the number of susceptible decreases while the number of recovered increases. But if in the long run I tends to 0, the number of susceptible does not decreases to zero, because of herd immunity. Herd immunity is reached when a sufficient proportion of individuals have been infected and have become immune to the virus. This proportion of immune people depends on the contagiousness of the disease and is equal to:

$$R^\star = 1 - 1/\mathcal{R}_0.$$

To this proportion corresponds the equilibrium proportion of infected people:

$$S^\star = 1/\mathcal{R}_0.$$

This proportion is reached at the peak of the epidemic and is usually lower than the limiting value S_∞ when $t \to \infty$. So the model is overshooting by a non-negligible percentage as will be detailed below. With a plausible value of $\mathcal{R}_0 = 2.5$ for the COVID-19, the herd immunity threshold is $S^\star = 0.4$, meaning that herd immunity is reached when 60% of the population has recovered or is protected by a vaccine.

The probability of dying is a constant proportion π of the infected, completing thus the model by a fourth equation:

$$\frac{\mathrm{d}D}{\mathrm{d}t} = \pi\gamma I, \tag{1.5}$$

which simply means that the proportion of deaths is a fraction of R with $D = \pi R$. This variable has no action on the dynamics of the model, but its prediction is of course of prime importance. As a matter of fact, most of the controversies reported in the literature (see, for instance Adam (2020)) concern the predicted number of deaths. The number of deaths at the end of the epidemic is computed as:

$$D = (1 - S_\infty)\pi \times N \times S_0 \tag{1.6}$$

1.2 Phase diagram

The dynamics of the model is best described using phase diagrams as advocated in Moll (2020). Phase diagrams plot S against I, assuming $S + I < 1$. After some algebraic manipulations, we can find the number of Infected as a function of the number of Susceptible, the $\mathcal{R}_0$ and the initial conditions. We get:

$$I_t = 1 - R_0 - S_t + \frac{1}{\mathcal{R}_0} log(S_t/S_0), \tag{1.7}$$

which is convenient for analysing some properties of the model. Typical initial conditions are:

$$\begin{aligned} S_0 &= 1 - I_0, \\ I_0 &\approx 0, \\ R_0 &= 0. \end{aligned}$$

where I_0 can be set for instance to $1/N$. With these elements in mind, a phase diagram can be drawn. For given initial conditions and a given grid of S_t , the corresponding proportion of infected persons is obtained.

1.3 Introducing a Lockdown

A lock-down is introduced in the SIR model by considering a time variable β_t . If ℓ_t is the strength of the lock-down and β_0 the value of β in the absence of lock-down, then: $\beta_t = \beta_0 \times (1 - \ell_t)$,

so that a lock-down is a very efficient way of decreasing the value of β_t . It implies that:

$$\mathcal{R}_t = (1 - \ell_t)\mathcal{R}_0 S_t,$$

which means that with a very strict lock-down the epidemic ceases to spread out. But that does not mean that the epidemic will cease, once the lock-down is removed.

With a very strict lock-down the epidemic ceases to expand at an exponential rate. But that does not mean that the epidemic will stop immediately. However, a lock-down is applied over a limited period, so we have to be able to provide a graph where time is the horizontal axis. So we have to find a numerical way to find the trajectory of the model in its three variables, and a simple phase diagram is no longer sufficient. For given values of the parameters, the trajectory of a SIR model can be found by discretizing the system with $\Delta_t < 1$ and use the Euler's method to solve the system:

$$\begin{aligned} S_i &= S_{i-1} - \beta_0(1 - \ell_i)S_{i-1}I_{i-1}\Delta_t, \\ I_i &= I_{i-1} + (\beta_0(1 - \ell_i)S_{i-1}I_{i-1} - \gamma I_{i-1})\Delta_t, \\ R_i &= I_{i-1} + \gamma I_{i-1}\Delta_t. \end{aligned} \tag{1.8}$$

When iterating this system, $1/\Delta_t$ iterations are needed to cover one period when the parameters are calibrated on a daily basis.

2 Simulations with R

In this chapter, we use R to simulate a SIR model, with and without lockdown.

2.1 Without lockdown

We can simulate a SIR without lockdown first. Let us adapt the The Matlab codes codes provided by Benjamin Moll.

We will need some functions to manipulate data from {tidyverse}.

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.2     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

Let us set the reproduction number $\mathcal{R}_0 = 2.5$:

```r
reprod_number <- 2.5
Tinf <- 7
beta <- reprod_number / Tinf
gamma <- 1 / Tinf
```

The choice of the number of periods can be made here:

```r
implicit <- 0
T <- 400    # Length of simulation
```

```r
dt <- 0.1    # Time step
Nt <- T / dt + 1
time <- seq(0, T, length.out = Nt)
```

A matrix with 3 rows can be initiated. Each row corresponds to S, I, and R, resp.

```r
mu <- matrix(rep(0, 3 * (Nt + 1)), nrow = 3)
S <- I <- R <- N <- F <- Re <- rep(0, Nt)
```

The transition matrices at each time step (we initialize an empty list):

```r
A_t <- vector(mode = "list", length = Nt)
```

Setting the initial conditions for S, I et R:

```r
# March 1st 2020 so that March 31st there are appx. 150,000 cases
# (50% more than measured)
I0 <- 0.35 * 10^(-4)
S0 <- 1 - I0
R0 <- 0
```

Let the first column of `mu` contain the initial conditions for S_0, I_0, and R_0

```r
mu[,1] <- matrix(c(S0, I0, R0), nrow = 3)
```

Now, a loop over the different periods can be made. Benjamin Moll uses the matrix form of the SIR model here.

```r
for (n in 1:(Nt - 1)) {
  S[n] <- mu[1, n]
  I[n] <- mu[2, n]
  R[n] <- mu[3, n]
  Re[n] <- reprod_number * S[n]
  A_t[[n]] <- matrix(
    c(
      -beta * I[n], beta * I[n], 0,
      0, -gamma, gamma,
      0, 0, 0
    ),
    byrow = TRUE,
    ncol = 3)
  if (implicit == 0) {
    mu[, n+1] <- dt * (t(A_t[[n]]) %*% mu[, n]) + mu[, n]
  } else {
    mu[n + 1, ] <- (I(3) - dt * (t(A_t[[n]])) %/% mu[, n])
  }
}

mu <- mu[, 1:Nt]
```

Recall that the rows of matrix `mu` contain the values for S, I, and R, respectively. The columns correspond to the values at all the periods.

```
S <- mu[1, ]
I <- mu[2, ]
R <- mu[3, ]

N <- S + I + R
D <- 100 - N

df_lockdown <- tibble(S = S, I = I, R = R, t = time, type = "no-lockdown")
df_lockdown
```

```
# A tibble: 4,001 x 5
       S         I           R     t type
   <dbl>     <dbl>       <dbl> <dbl> <chr>
 1  1.00 0.000035  0             0   no-lockdown
 2  1.00 0.0000357 0.0000005     0.1 no-lockdown
 3  1.00 0.0000365 0.00000101    0.2 no-lockdown
 4  1.00 0.0000373 0.00000153    0.3 no-lockdown
 5  1.00 0.0000381 0.00000207    0.4 no-lockdown
 6  1.00 0.0000389 0.00000261    0.5 no-lockdown
 7  1.00 0.0000397 0.00000317    0.6 no-lockdown
 8  1.00 0.0000406 0.00000373    0.7 no-lockdown
 9  1.00 0.0000415 0.00000431    0.8 no-lockdown
10  1.00 0.0000424 0.00000491    0.9 no-lockdown
# i 3,991 more rows
```

2.2 With lockdown

Now, let us create a function that will run the SIR model with a lockdown. The code from the SIR model without lockdown is slightly modified, to introduce the lockdown and its severity.

The following function allows us to easily make a simulation depending on the start and end of the lockdown, as well as its severity:

```r
#' Simulate the SIR model with a lockdown
#'
#' @param lock_start start of the lockdown (number of days from the outbreak)
#' @param lock_end end of the lockdown (number of days from the outbreak)
#' @param lock_severity severity of the lockdown (in [0,1], 0 for no lockdown)
simulate_sir_lockdown <- function(lock_start,
                                  lock_end,
                                  lock_severity,
                                  type) {
  # Matrix with 3 rows, each corresponding to S, I and R, resp.
  mu <- matrix(rep(0, 3 * (Nt + 1)), nrow = 3)
  S <- I <- R <- N <- F <- Re <- rep(0, Nt)

  # Transition matrix
  A_t <- vector(mode="list", length = Nt)

  # First column: initial conditions
  mu[, 1] <- matrix(c(S0, I0, R0), nrow = 3)

  lockdown <- rep(0, Nt)

  for (n in 1:(Nt - 1)) {
    S[n] <- mu[1, n]
    I[n] <- mu[2, n]
    R[n] <- mu[3, n]

    if (time[n] >= lock_start & time[n] <= lock_end) {
      # Lockout
      lockdown[n] <- lock_severity
    }

    Re[n] <- reprod_number * (1 - lockdown[n]) * S[n]

    A_t[[n]] <- matrix(
      c(
        -beta * (1-lockdown[n]) * I[n], beta * (1 - lockdown[n]) * I[n], 0,
        0, -gamma, gamma,
        0, 0, 0
      ),
      byrow = TRUE,
      ncol = 3)

    if (implicit == 0) {
      mu[, n+1] <- dt * (t(A_t[[n]]) %*% mu[, n]) + mu[, n]
    } else {
      mu[n + 1, ] <- (I(3) - dt * (t(A_t[[n]])) %/% mu[, n])
    }
  }

  mu <- mu[, 1:Nt]

  S_tight <- mu[1, ]
  I_tight <- mu[2, ]
  R_tight <- mu[3, ]

  N_tight <- S_tight + I_tight + R_tight
  D_tight <- 100 - N_tight

  tibble(S = S_tight, I = I_tight, R = R_tight, t = time)
}
```

Now let us set the values for a tight lockdown:

```r
lock_start_tight <- 37
lock_end_tight <- lock_start_tight+30
lock_severity_tight <- .7
```

An let us use those values in the modified SIR model:

```r
df_lockdown_tight <-
  simulate_sir_lockdown(lock_start = lock_start_tight,
                        lock_end = lock_end_tight,
                        lock_severity = .7) |>
  mutate(type = "tight-lockdown")
```

2.2.0.1 Tight lockdown

Now let us set the values for a tight lockdown:

```
lock_start_tight <- 37
lock_end_tight <- lock_start_tight + 30
lock_severity_tight <- .7
```

An let us use those values in the modified SIR model:

```
df_lockdown_tight <- simulate_sir_lockdown(
  lock_start = lock_start_tight,
  lock_end = lock_end_tight,
  lock_severity = .7) |>
  mutate(type = "tight-lockdown")
```

2.2.0.2 Loose Lockdown

The following values for the lockdown lead to simulating a loose lockdown:

```
lock_start_loose <- 37
lock_end_loose <- lock_start_loose + 30
lock_severity_loose <- .4
```

These values can be used to make a new simulation of the SIR model:

```
df_lockdown_loose <- simulate_sir_lockdown(
  lock_start = lock_start_loose,
  lock_end = lock_end_loose,
  lock_severity = lock_severity_loose) |>
  mutate(type = "loose-lockdown")
```

2.2.0.3 Moderate Lockdown

A moderate lockdown can then be considered, picking the following values:

```
lock_start_mix <- 37
lock_end_mix <- lock_start_mix + 90
lock_severity_mix <- .425
```

And these values can be given to the simulation function:

```r
df_lockdown_mix <- simulate_sir_lockdown(
  lock_start = lock_start_mix,
  lock_end = lock_end_mix,
  lock_severity = lock_severity_mix) |>
  mutate(type = "mix-lockdown")
```

2.3 Graphs of the evolution

The datasets for each scenario need to be reshaped to be used in `ggplot2()`.

```r
#' Reshape the data from two scenarios
#'
#' @param df_scenario_1 data for first scenario
#' @param df_scenario_2 data for second scenario
#' @param name_scenario_1 name of the scenario in `df_scenario_1`
#' @param label_scenario_1 desired label for the name of the first scenario
#' @param name_scenario_2 name of the scenario in `df_scenario_2`
#' @param label_scenario_2 desired label for the name of the second scenario
reshape_data_graph <- function(df_scenario_1,
                               df_scenario_2,
                               name_scenario_1,
                               label_scenario_1,
                               name_scenario_2,
                               label_scenario_2) {
  df_scenario_1 |>
  bind_rows(df_scenario_2) |>
  pivot_longer(cols = c("S", "I", "R")) |>
  filter(t <= Nt - 1) |>
  mutate(name = factor(name, levels = c("S", "I", "R")),
         type = factor(
           type,
           levels = c(name_scenario_1, name_scenario_2),
           labels = c(label_scenario_1, label_scenario_2)))
}
```

We define a theme function to make the graphs pretty (at least for us!).

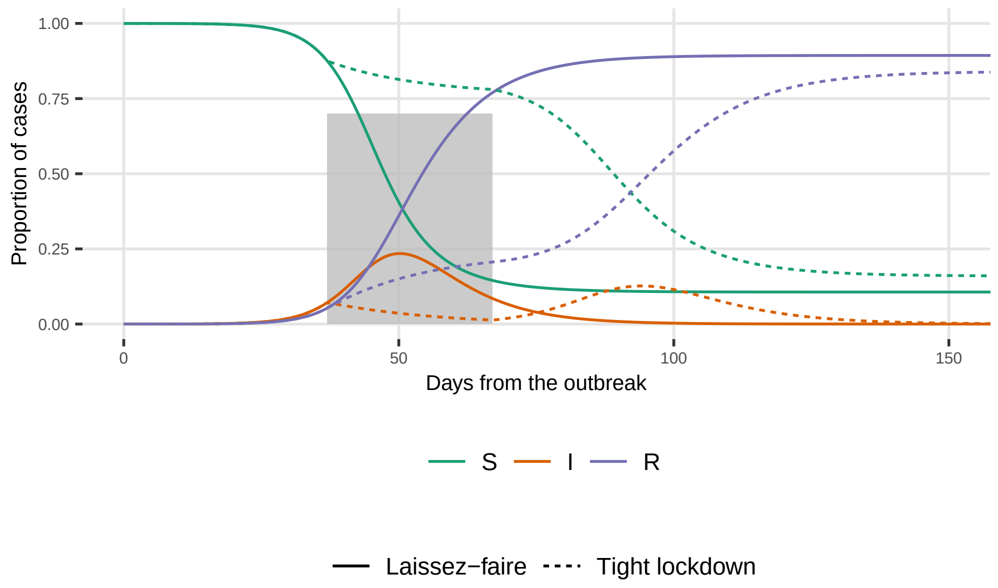
```r
library(grid)
theme_paper <- function(..., size_text = 8)
  theme(text = element_text(size = size_text),
        plot.background = element_rect(fill="transparent", color=NA),
        panel.background = element_rect(fill = "transparent", color=NA),
        panel.border = element_blank(),
        axis.text = element_text(),
        legend.text = element_text(size = rel(1.1)),
        legend.title = element_text(size = rel(1.1)),
        legend.background = element_rect(fill="transparent", color=NULL),
        legend.position = "bottom",
        legend.direction = "horizontal", legend.box = "vertical",
        legend.key = element_blank(),
        panel.spacing = unit(1, "lines"),
        panel.grid.major = element_line(colour = "grey90"),
        panel.grid.minor = element_blank(),
        plot.title = element_text(hjust = 0, size = rel(1.3), face = "bold"),
        plot.title.position = "plot",
        plot.margin = unit(c(1, 1, 1, 1), "lines"),
        strip.background = element_rect(fill=NA, colour = NA),
        strip.text = element_text(size = rel(1.1)))
```

And we define a function to plot the evolution of the two scenarios with respect to time.

```
#' Plots the evolution of two scenarios
#'
#' @param df_plot data with the two scenarios (obtained from
#' `reshape_data_graph()`)
#' @param lock_start start period of the lockdown
#' @param lock_end end period of the lockdown
#' @param lock_severity severity of the lockdown
plot_scenario <- function(df_plot,
                          lock_start,
                          lock_end,
                          lock_severity) {
  ggplot(data = df_plot) +
  geom_rect(
    data = tibble(x1 = lock_start,
                  x2 = lock_end,
                  y1 = 0, y2 = lock_severity),
    mapping = aes(xmin = x1, xmax = x2, ymin = y1, ymax = y2),
    fill = "grey", alpha = .8
  ) +
  geom_line(aes(x = t, y = value, linetype = type, colour = name)) +
  scale_linetype_discrete(NULL) +
  scale_colour_manual(
    NULL,
    values = c("S" = "#1b9e77", "I" = "#d95f02", "R" = "#7570b3")) +
  labs(x = "Days from the outbreak", y = "Proportion of cases") +
  theme_paper() +
  coord_cartesian(xlim = c(0, 150))
}
```

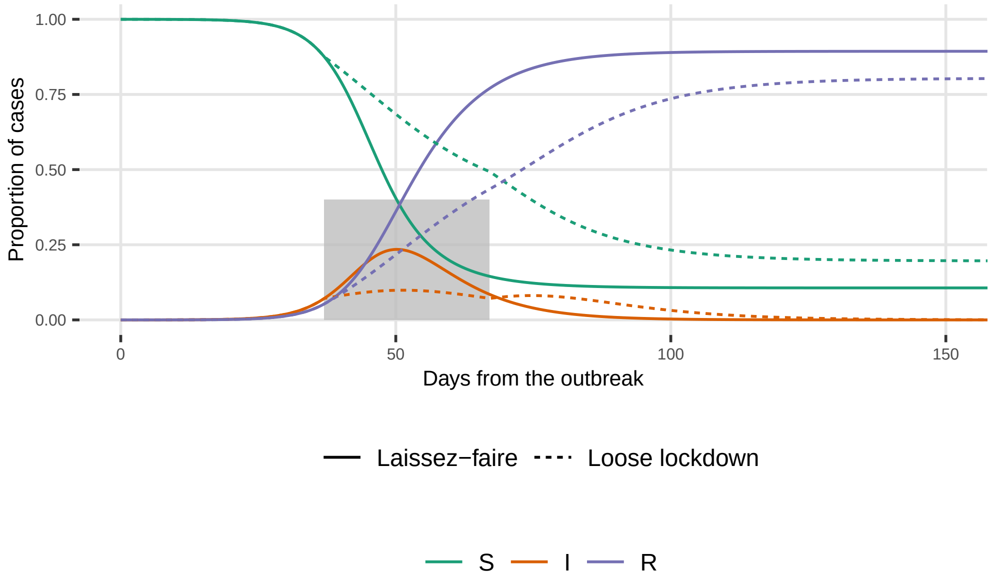
2.3.0.1 Tight lockdown

```
df_plot_lockdown_tight <- reshape_data_graph(
  df_scenario_1 = df_lockdown,
  df_scenario_2 = df_lockdown_tight,
  name_scenario_1 = "no-lockdown",
  label_scenario_1 = "Laissez-faire",
  name_scenario_2 = "tight-lockdown",
  label_scenario_2 = "Tight lockdown")
plot_scenario(
  df_plot = df_plot_lockdown_tight,
  lock_start = lock_start_tight,
  lock_end = lock_end_tight,
  lock_severity = lock_severity_tight)
```



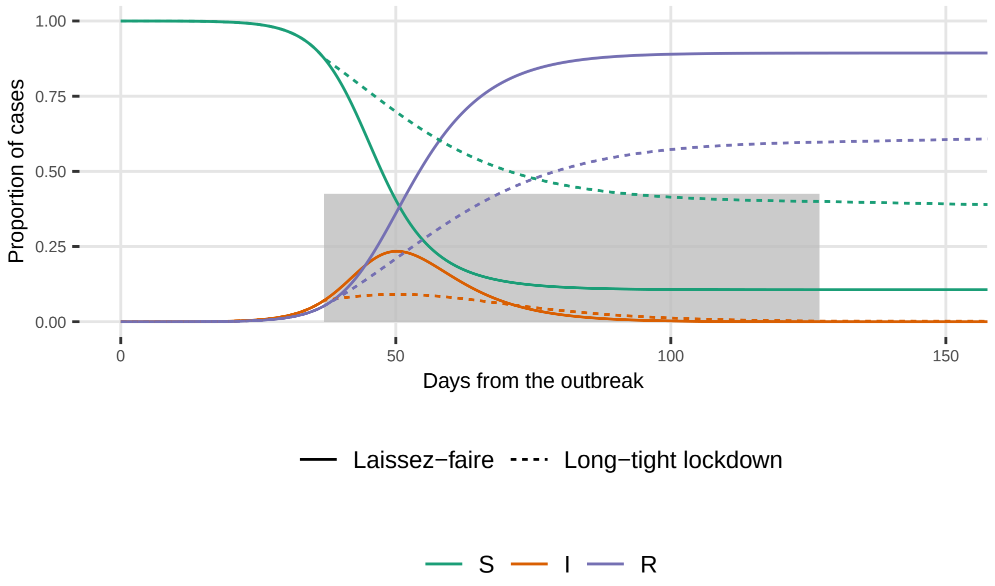
2.3.0.2 Loose lockdown

```r
df_plot_lockdown_loose <- reshape_data_graph(
  df_scenario_1 = df_lockdown,
  df_scenario_2 = df_lockdown_loose,
  name_scenario_1 = "no-lockdown",
  label_scenario_1 = "Laissez-faire",
  name_scenario_2 = "loose-lockdown",
  label_scenario_2 = "Loose lockdown")
plot_scenario(
  df_plot = df_plot_lockdown_loose,
  lock_start = lock_start_loose,
  lock_end = lock_end_loose,
  lock_severity = lock_severity_loose)
```



2.3.0.3 Moderate lockdown

```r
df_plot_lockdown_mix <- reshape_data_graph(
  df_scenario_1 = df_lockdown,
  df_scenario_2 = df_lockdown_mix,
  name_scenario_1 = "no-lockdown",
  label_scenario_1 = "Laissez-faire",
  name_scenario_2 = "mix-lockdown",
  label_scenario_2 = "Long-tight lockdown")
plot_scenario(
  df_plot = df_plot_lockdown_mix,
  lock_start = lock_start_mix,
  lock_end = lock_end_mix,
  lock_severity = lock_severity_mix)
```



2.4 Phase diagrams

The phase diagrams can also be plotted. Once again, the data need to be reshaped for use in `ggplot2`.

```
#' @param df_scenario_1 data for first scenario
#' @param df_scenario_2 data for second scenario
#' @param name_scenario_1 name of the scenario in `df_scenario_1`
#' @param label_scenario_1 desired label for the name of the first scenario
#' @param name_scenario_2 name of the scenario in `df_scenario_2`
#' @param label_scenario_2 desired label for the name of the second scenario
reshape_data_phase <- function(df_scenario_1,
                               df_scenario_2,
                               name_scenario_1,
                               label_scenario_1,
                               name_scenario_2,
                                   label_scenario_2) {
  df_scenario_1 |>
    filter(I > 0.001) |>
    group_by(S) |>
    slice(1) |>
    ungroup() |>
    bind_rows(
      df_scenario_2 |>
        filter(I > 0.001) |>
        group_by(S) |>
        slice(1) |>
        ungroup()
    ) |>
    mutate(
      type = factor(
        type,
        levels = c(name_scenario_1, name_scenario_2),
        labels = c(label_scenario_1, label_scenario_2))
    )
}
```

If we want to display some arrows to highlight the direction on the diagrams, we can create a function that will do so for a single scenario.

```
#' Gives the coordinates to draw a small arrow on the phase diagram of one
#' scenario
#'
#' @param df_plot table with the data of the diagram of one scenario ready for
#' use in ggplot2
#' @param val_S value for S
create_df_arrow <- function(df_plot,
                            val_S) {
  df_arrows <-
    df_plot |>
    arrange(desc(S)) |>
    group_by(type) |>
    filter(S < !!val_S) |>
    slice(1:2) |>
    select(S, I, type) |>
    ungroup() |>
    mutate(toto = rep(c("beg", "end"), 2))
  df_arrows |>
    select(-I) |>
    pivot_wider(names_from = toto, values_from = S, names_prefix = "S_") |>
    left_join(
      df_arrows |>
        select(-S) |>
        pivot_wider(names_from = toto, values_from = I, names_prefix = "I_"),
      by =  "type"
    )
}
```

Then, let us create a function that will plot the phase diagrams for two scenarios.

```
plot_phase_diagram_scenarios <- function(df_plot,
                                         x_arrows,
                                         reprod_number = 2.5
                                         ) {
  # Ending points of the epidemic
  df_dots <- df_plot |>
    group_by(type) |>
    filter(I == min(I))

  # Arrow
  df_arrows <-
    map_df(x_arrows, ~create_df_arrow(df_plot, val_S = .))

  # The coordinates of $S$ corresponding to herd immunity
  S_herd <- 1./reprod_number
  curve_scenario <- tibble(
    x1 = S_herd + .1,
    x2 = S_herd,
    y1 = .30,
    y2 = .27)
  label_curve_scenario <-
    tibble(
      x = S_herd + .15,
      y = .3,
      label = "Herd Immunity"
    )

  # The graph
  ggplot() +
    geom_line(
      data = df_plot,
      mapping = aes(x = S, y = I, linetype = type)
    ) +
    geom_vline(xintercept = S_herd, linetype = "dashed") +
    geom_curve(
      data = df_arrows,
      mapping = aes(x = S_beg, xend = S_end,
                    y = I_beg, yend = I_end),
      arrow = arrow(length = unit(0.04, "npc")),
      curvature = 0
    ) +
    geom_curve(
      data = curve_scenario,
      mapping = aes(x = x1, y = y1, xend = x2, yend = y2),
      arrow = arrow(length = unit(0.03, "npc")), curvature = -0.2)  +
    geom_label(
      data = label_curve_scenario,
      mapping = aes(x = x, y = y, label = label),
      size = rel(3)
    ) +
    geom_point(
      data = df_dots,
      mapping = aes(x = S, y = I),
      colour = "red", size = 3
      ) +
    scale_linetype_discrete(NULL) +
    scale_x_continuous(breaks = seq(0, 1, by = .1)) +
    labs(x = "Susceptible", y = "Infected") +
    theme_paper()

}
```

2.4.0.1 Tight lockdown

```
df_plot_tight <-
  reshape_data_phase(
  df_scenario_1     = df_lockdown,
  df_scenario_2     = df_lockdown_tight,
  name_scenario_1  = "no-lockdown",
  label_scenario_1 = "Laissez-faire",
  name_scenario_2  = "tight-lockdown",
  label_scenario_2 = "Tight lockdown")

plot_phase_diagram_scenarios(
  df_plot = df_plot_tight,
  x_arrows = c(.7, .5, .3, .2),
  reprod_number = 2.5) +
  coord_cartesian(ylim = c(0, .35))
```

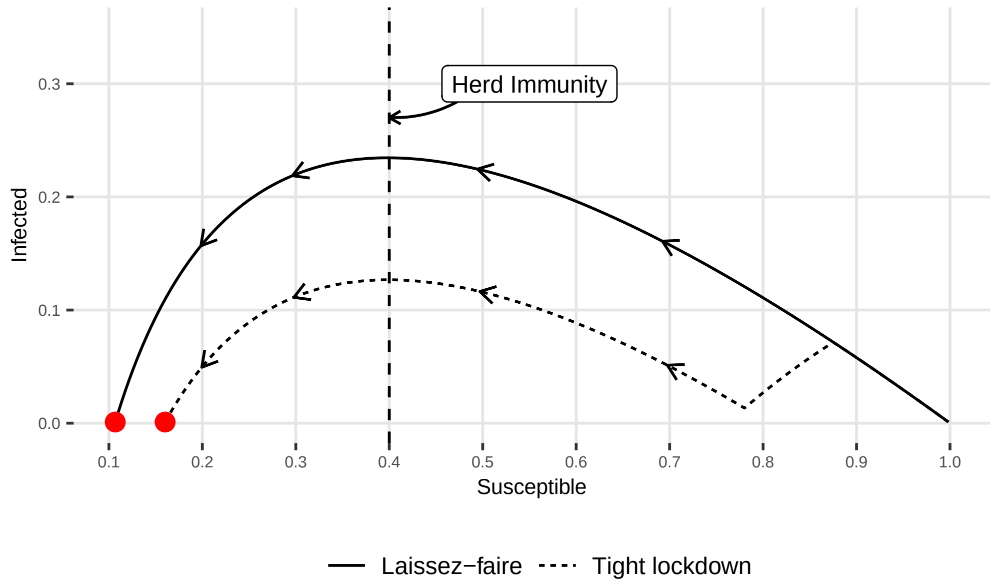


Figure 2.1: Tight Lockdown: Phase Diagram

2.4.0.2 Loose lockdown

```
df_plot_loose <-
  reshape_data_phase(
  df_scenario_1    = df_lockdown,
  df_scenario_2    = df_lockdown_loose,
  name_scenario_1  = "no-lockdown",
  label_scenario_1 = "Laissez-faire",
  name_scenario_2  = "loose-lockdown",
  label_scenario_2 = "Loose lockdown")
```

```
plot_phase_diagram_scenarios(
  df_plot = df_plot_loose,
  x_arrows = c(.7, .55, .3),
  reprod_number = 2.5
) +
  coord_cartesian(ylim = c(0, .35))
```

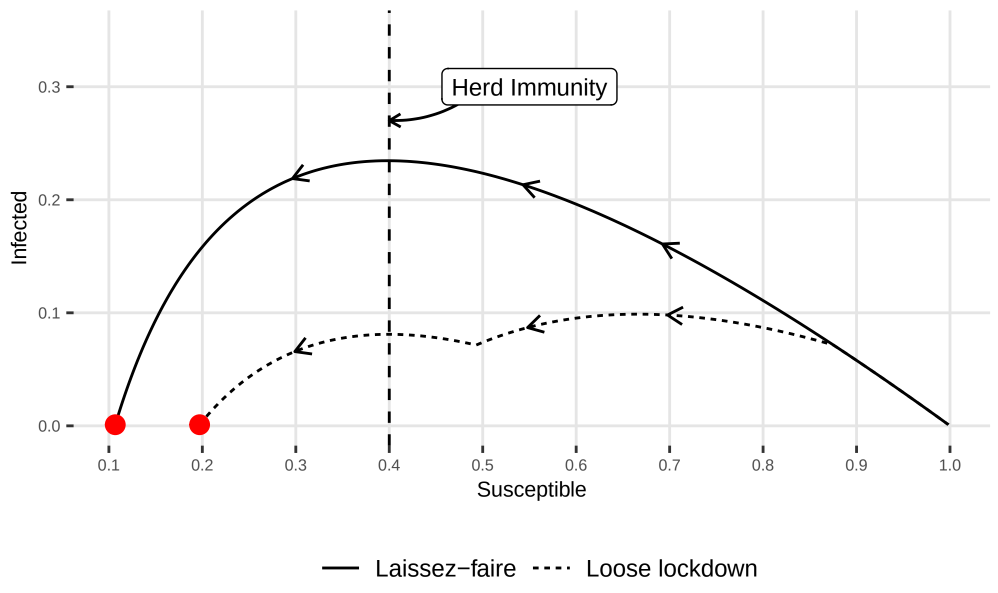


Figure 2.2: Loose Lockdown: Phase Diagram

2.4.0.3 Moderate lockdown

```
df_plot_mix <-
  reshape_data_phase(
  df_scenario_1     = df_lockdown,
  df_scenario_2     = df_lockdown_mix,
  name_scenario_1   = "no-lockdown",
  label_scenario_1 = "Laissez-faire",
  name_scenario_2   = "mix-lockdown",
  label_scenario_2 = "Long-tight lockdown")

plot_phase_diagram_scenarios(
  df_plot = df_plot_mix,
  x_arrows = c(.7, .5),
  reprod_number = 2.5) +
  coord_cartesian(ylim = c(0, .35))
```

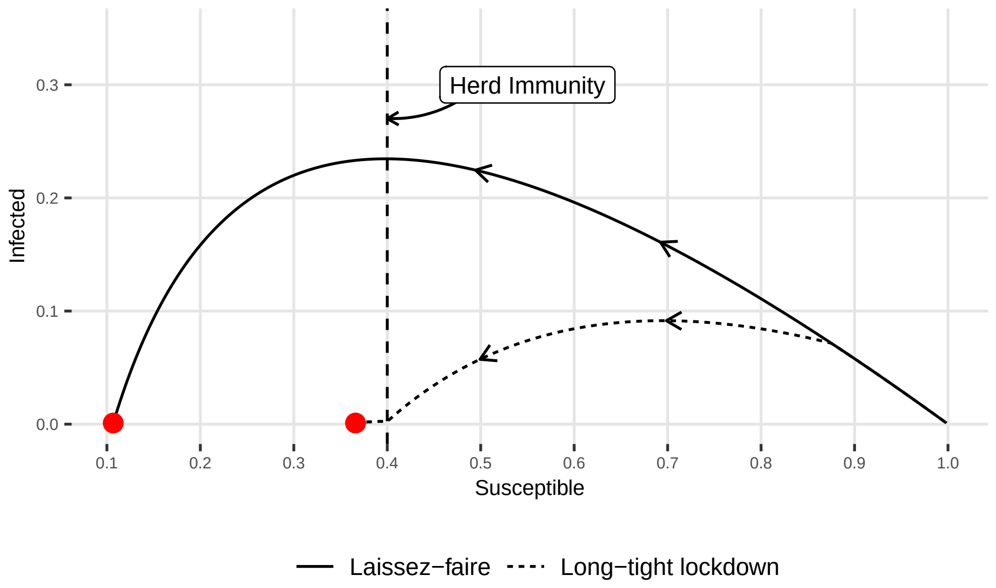


Figure 2.3: Long-tight Lockdown: Phase Diagram

3 Animations with R

In this chapter, we use R to make an animated version of the simulation of the epidemics using the SIR model.

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.2     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
library(raster)
```

```
Loading required package: sp

Attaching package: 'raster'

The following object is masked from 'package:dplyr':

    select
```

We will consider that individuals live on a square with sides of 5.

```r
bounds <- c(-5,5)
```

Each time unit, individuals can take a step. The size of the step is defined with the following variable:

```r
step_size <- abs(max(bounds)-min(bounds))/30
```

Let us set some key parameters of the epidemics:

- The probability that a S gets infected while in contact with an I is set here to .8
- To be infected, we consider a radius of .5 around an I.
- At the beginning of the simulation, the number of infected is set to .02.
- Once an individual has been infected, it takes 7 periods before it turns to R.

```r
proba_infect <- 0.8
radius_infec <- 0.5
I_0 <- 0.02
time_recovery <- 7
```

Let us consider $N = 150$ individuals, and a time span of 100 periods.

```r
n <- 150
nsteps <- 100
```

The position of the individuals will be stored in the following object:

```r
positions <- vector(mode = "list", length = nsteps)
```

The initial positions are randomly drawn from a Uniform distribution $\mathcal{U}(-5, 5)$.

```r
set.seed(123)
x <- runif(n = n, min = min(bounds), max = max(bounds))
y <- runif(n = n, min = min(bounds), max = max(bounds))
```

At each moment, let us store the status of the infected in an object called `status`. Here are the random status at the beginning. On average, if we replicate the analysis multiple times, the proportion of infected at the beginning of the epidemic should be equal to 0.02.

```r
status <- sample(
  c("S", "I"),
  replace = TRUE,
  size = round(n),
  prob = c(1-I_0, I_0)
)
table(status)
```

```
status
```

```
  I   S 
  6 144 
```

```r
prop.table(table(status))
```

```
status
   I    S 
0.04 0.96 
```

The number of infected at the beginning:

```r
nb_infected_start <- sum(status == "I")
nb_infected_start
```

```
[1] 6
```

Let us store the current state of our simulated world in a tibble:

```r
df_current <- tibble(x = x, y = y, step = 1, status = status)
```

At each time, the state of the world will be stored in the ith element of `positions`.

```r
positions[[1]] <- df_current
```

We need to identify the individuals with respect to their status:

```r
id_I <- which(df_current$status == "I")
id_S <- which(df_current$status == "S")
id_R <- NULL
```

The time since infection is set to 0 for all individuals…

```r
time_since_infection <- rep(0, n)
```

… except for those that are infected at the first period.For those, the time to infection is set to 1:

```r
time_since_infection[id_I] <- 1
```

Then we can begin the loop over the periods.

```r
for(i in 2:nsteps) {
  # Movements, in both directions
  deplacement_x <- runif(n = n, min = -step_size, max = step_size)
  deplacement_y <- runif(n = n, min = -step_size, max = step_size)

  x <- x+deplacement_x
  y <- y+deplacement_y

  x[x>max(x)] <- max(bounds)
  x[x<min(x)] <- min(bounds)

  y[y>max(y)] <- max(bounds)
  y[y<min(y)] <- min(bounds)

  # Recovery
  id_new_recovered <- which(time_since_infection>time_recovery)
  if (length(id_new_recovered) > 0) {
    id_R <- c(id_R, id_new_recovered)
    id_I <- id_I[!id_I %in% id_new_recovered]
    status[id_new_recovered] <- "R"
    time_since_infection[id_new_recovered] <- 0
  }

  # New infections
  dist_matrix <- pointDistance(
    cbind(x[id_I], y[id_I]),
    cbind(x[id_S], y[id_S]),
    lonlat=FALSE
  )

  # Identifying close individuals
  close_points <- which(dist_matrix < radius_infec, arr.ind = TRUE)

  if (length(close_points) > 0) {
    ids_potential_new_infected <- id_S[close_points[,"col"]]
    are_infected <- sample(
      c(TRUE, FALSE),
      size = nrow(close_points),
      prob = c(proba_infect, 1 - proba_infect),
      replace = TRUE
    )
    ids_new_infected <- ids_potential_new_infected[are_infected]

    time_since_infection[time_since_infection != 0] <-
      time_since_infection[time_since_infection != 0] + 1

    time_since_infection[ids_new_infected] <- 1

    id_I <- sort(c(id_I, ids_new_infected))
    status[ids_new_infected] <- "I"
    id_S <- id_S[!id_S %in% ids_new_infected]
  }

  # Storing the current state of the world
  positions[[i]] <- tibble(x = x, y = y, step = i, status = status) |>
    mutate(status = factor(status, levels = c("S", "I", "R")))

  # Plotting the situation
  p <-
    ggplot(
      data = positions[[i]],
      mapping = (aes(x = x, y = y, colour = status))) +
    ggforce::geom_circle(
      data = positions[[i]] |>
        filter(status == "I") |>
        mutate(r = radius_infec),
      mapping = aes(
        x0 = x, y0 = y, r = r
      ),
      fill = "#d95f02",
      colour = NA,
      alpha = .1,
      inherit.aes = FALSE
    ) +
    geom_point(size = 1) +
    labs(x = "", y = NULL,
         title = str_c("No. Infected: ", length(id_I))) +
    coord_equal(xlim = bounds, ylim = bounds) +
    scale_colour_manual("Status", values = c(
      "S" = "#1b9e77",
      "I" = "#d95f02",
      "R" = "#7570b3"
    ),
    guide = "none") +
    theme(
      plot.title.position = "plot",
      panel.border = element_rect(linetype = "solid", fill=NA),
      panel.grid = element_blank(),
      axis.ticks = element_blank(),
      axis.text = element_blank()
    )

  if (length(close_points) > 0) {
    if (length(ids_new_infected) > 0) {
      p <- p +
        geom_point(
          data = positions[[i]] |>
            slice(ids_new_infected),
          mapping = aes(x = x, y = y),
          size = 2, colour = "black"
        ) +
        geom_point(
          data = positions[[i]] |>
            slice(ids_new_infected),
          mapping = aes(x = x, y = y),
          size = 1.5
        )
    }

  }

  df_plot <-
    map_df(
      positions[1:i],
      ~group_by(., status) |> count(),
      .id = "time"
    ) |>
    ungroup() |>
    bind_rows(
      tibble(
        time = "0",
        status = c("S", "I", "R"),
        n = c(n - nb_infected_start, nb_infected_start, 0)
      )
    ) |>
    complete(status, time) |>
    mutate(
      n = replace_na(n, 0),
      time = as.numeric(time),
      status = factor(status, levels = c("S", "I", "R"))
    )

  p_counts <-
    ggplot(
      data = df_plot,
      mapping = aes(x = time, y = n, colour = status)) +
    geom_line() +
    scale_colour_manual(
      "Status", values = c(
        "S" = "#1b9e77",
        "I" = "#d95f02",
        "R" = "#7570b3"
      )
    ) +
    labs(x = "Time", y = NULL, title = "Counts") +
    theme(plot.title.position = "plot")


  p_both <- cowplot::plot_grid(p, p_counts)

  file_name <- str_c(
    "figs/simul_",
    str_pad(i, width = 3, side = "left", pad = 0),
    ".png"
  )
  cowplot::save_plot(
    filename = file_name,
    p_both, ncol = 2,
    base_asp = 1.1,
    base_width = 5,
    base_height = 5,
    dpi=72
  )
}
```

The png files can be merged in a gif file with a system command:

```
system("convert figs/*.png figs/animation-sir.gif")
```

The resulting gif can be seen in Figure 3.1.

```
if (knitr::is_html_output()) {
  knitr::include_graphics("figs/animation-sir.gif")
} else if (knitr::is_latex_output()) {
  knitr::include_graphics("figs/simul_100.png")
}
```

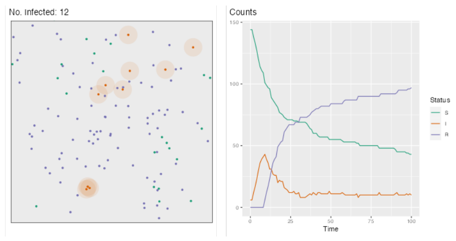


Figure 3.1: SIR simulation over 100 periods, with $I_0 \approx 0.02$

Part II

Statistical Models

4 Covid-19 Data

This chapter provides codes that can be used to work with the data from Oxford University: Oxford Covid-19 Government Response Tracker (OxCGRT).

Here, we will focus on 5 large and 5 small European countries, in terms of inhabitants: - “Large countries”: United Kingdom, Spain, Italy, Germany, France, - “Small countries”: Sweden, Belgium, Netherlands, Ireland, Denmark.

4.1 Load data

First of all, let us load some packages.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.2     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```
library(lubridate)
library(knitr)
library(kableExtra)
```

```
Attaching package: 'kableExtra'

The following object is masked from 'package:dplyr':
    group_rows
```

```r
library(RColorBrewer)
library(growthmodels)
library(minpack.lm)
library(scales)
```

```

Attaching package: 'scales'

The following object is masked from 'package:purrr':

    discard

The following object is masked from 'package:readr':

    col_factor
```

```r
library(nlstools)
```

```

'nlstools' has been loaded.

IMPORTANT NOTICE: Most nonlinear regression models and data set examples
related to predictive microbiolgy have been moved to the package 'nlsMicrobio'
```

```r
library(ggpubr)
library(gridExtra)
```

```

Attaching package: 'gridExtra'

The following object is masked from 'package:dplyr':

    combine
```

Let us also define a theme for the graphical outputs, as in Chapter 1

```r
library(grid)
theme_paper <- function(..., size_text = 8)
  theme(text = element_text(size = size_text),
        plot.background = element_rect(fill="transparent", color=NA),
        panel.background = element_rect(fill = "transparent", color=NA),
        panel.border = element_blank(),
        axis.text = element_text(),
        legend.text = element_text(size = rel(1.1)),
        legend.title = element_text(size = rel(1.1)),
        legend.background = element_rect(fill="transparent", color=NULL),
        legend.position = "bottom",
        legend.direction = "horizontal", legend.box = "vertical",
        legend.key = element_blank(),
        panel.spacing = unit(1, "lines"),
        panel.grid.major = element_line(colour = "grey90"),
        panel.grid.minor = element_blank(),
        plot.title = element_text(hjust = 0, size = rel(1.3), face = "bold"),
        plot.title.position = "plot",
        plot.margin = unit(c(1, 1, 1, 1), "lines"),
        strip.background = element_rect(fill=NA, colour = NA),
        strip.text = element_text(size = rel(1.1)))
```

Let us modify the locale settings. This depends on the OS. For Windows users:

```r
Sys.setlocale("LC_ALL", "English_United States")
```

For Unix users:

```r
# Only for Unix users
Sys.setlocale("LC_ALL", "en_US.UTF-8")
```

```
[1] "en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8"
```

4.1.1 Confirmed and Deaths data

As mentioned at the beginning of the notebook, we rely on Oxford Covid-19 Government Response Tracker (OxCGRT) data.

Let us define the vector of country names:

```r
names_countries <- c("United Kingdom", "Spain", "Italy", "Germany", "France",
                     "Sweden", "Belgium", "Netherlands", "Ireland", "Denmark")
names_countries_large <- c("United Kingdom", "Spain", "Italy", "Germany", "France")
names_countries_small <- c("Sweden", "Belgium", "Netherlands", "Ireland", "Denmark")
```

The raw data can be downloaded as follows:

```r
df_oxford <-
  read.csv("https://raw.githubusercontent.com/OxCGRT/covid-policy-tracker/master/data/OxCG
```

Then we save those:

```r
dir.create("data")
save(df_oxford, file = "data/df_oxford.rda")
```

Let us load the saved data (data saved on May 29, 2023):

The dimensions are the following:

```r
dim(df_oxford)
```

```
[1] 202819     61
```

Let us focus first on the number of confirmed cases:

```r
confirmed_df <-
  df_oxford |>
  select(
    country = CountryName,
    country_code = CountryCode,
    date = Date,
    value = ConfirmedCases,
    stringency_index = StringencyIndex_Average) |>
  filter(country %in% names_countries) |>
  as_tibble() |>
  mutate(
    date = ymd(date),
    days_since_2020_01_22 =
      lubridate::interval(
        lubridate::ymd("2020-01-22"), date) / lubridate::ddays(1)
  )
```

We can do the same for the number of deaths:

```r
deaths_df <-
  df_oxford |>
  select(country = CountryName,
         country_code = CountryCode,
         date = Date,
         value = ConfirmedDeaths,
         stringency_index = StringencyIndex_Average) |>
  filter(country %in% names_countries) |>
  as_tibble() |>
  mutate(
    date = ymd(date),
    days_since_2020_01_22 =
      lubridate::interval(
        lubridate::ymd("2020-01-22"), date) / lubridate::ddays(1)
  )
```

4.1.1.1 Confirmed extract

Here is an extract from the last 3 rows of the confirmed cases:

```r
confirmed_df |>
  group_by(country) |>
  slice_tail(n = 3) |>
  arrange(country, desc(date)) |>
  kable()
```

The number of observation per country:

```r
confirmed_df |> group_by(country) |> count()
```

```
# A tibble: 10 x 2
# Groups:   country [10]
   country         n
   <chr>       <int>
 1 Belgium      1096
 2 Denmark      1096
 3 France       1096
 4 Germany      1096
 5 Ireland      1096
```

Table 4.1: Last 3 rows of confirmed cases, by country

country	country_code	date	value	stringency_index	days_since_2020_01_22
Belgium	BEL	2022-12-31	4668248	11.11	1074
Belgium	BEL	2022-12-30	4668248	11.11	1073
Belgium	BEL	2022-12-29	4668248	11.11	1072
Denmark	DNK	2022-12-31	3385836	11.11	1074
Denmark	DNK	2022-12-30	3385836	11.11	1073
Denmark	DNK	2022-12-29	3385836	11.11	1072
France	FRA	2022-12-31	38266999	11.11	1074
France	FRA	2022-12-30	38266999	11.11	1073
France	FRA	2022-12-29	38243191	11.11	1072
Germany	DEU	2022-12-31	37369866	11.11	1074
Germany	DEU	2022-12-30	37369865	11.11	1073
Germany	DEU	2022-12-29	37345969	11.11	1072
Ireland	IRL	2022-12-31	1687668	5.56	1074
Ireland	IRL	2022-12-30	1687668	5.56	1073
Ireland	IRL	2022-12-29	1687668	5.56	1072
Italy	ITA	2022-12-31	25143705	21.99	1074
Italy	ITA	2022-12-30	25143705	21.99	1073
Italy	ITA	2022-12-29	25021606	21.99	1072
Netherlands	NLD	2022-12-31	8569095	11.11	1074
Netherlands	NLD	2022-12-30	8569095	11.11	1073
Netherlands	NLD	2022-12-29	8565716	11.11	1072
Spain	ESP	2022-12-31	13684258	5.56	1074
Spain	ESP	2022-12-30	13684258	5.56	1073
Spain	ESP	2022-12-29	13670037	5.56	1072
Sweden	SWE	2022-12-31	2674862	11.11	1074
Sweden	SWE	2022-12-30	2674862	11.11	1073
Sweden	SWE	2022-12-29	2674862	11.11	1072
United Kingdom	GBR	2022-12-31	24135080	5.56	1074
United Kingdom	GBR	2022-12-30	24135080	5.56	1073
United Kingdom	GBR	2022-12-29	24135080	5.56	1072

```
 6 Italy           1096
 7 Netherlands     1096
 8 Spain           1096
 9 Sweden          1096
10 United Kingdom  1096
```

4.1.1.2 Deaths extract

Here is an extract from the last 3 rows of the number of deaths, for each country:

```
deaths_df |>
  group_by(country) |>
  slice_tail(n = 3) |>
  arrange(country, desc(date)) |>
  kable()
```

The number of observation per country:

```
deaths_df |> group_by(country) |> count()
```

```
# A tibble: 10 x 2
# Groups:   country [10]
   country            n
   <chr>          <int>
 1 Belgium         1096
 2 Denmark         1096
 3 France          1096
 4 Germany         1096
 5 Ireland         1096
 6 Italy           1096
 7 Netherlands     1096
 8 Spain           1096
 9 Sweden          1096
10 United Kingdom  1096
```

Let us keep in a table the correspondence between the country name and its ISO 3166-1 alpha-3 code:

```
country_codes <-
  confirmed_df |>
  select(country, country_code) |>
```

Table 4.2: Last 3 rows of deaths, by country

country	country_code	date	value	stringency_index	days_since_2020_01_22
Belgium	BEL	2022-12-31	33228	11.11	1074
Belgium	BEL	2022-12-30	33228	11.11	1073
Belgium	BEL	2022-12-29	33228	11.11	1072
Denmark	DNK	2022-12-31	7758	11.11	1074
Denmark	DNK	2022-12-30	7758	11.11	1073
Denmark	DNK	2022-12-29	7758	11.11	1072
France	FRA	2022-12-31	158385	11.11	1074
France	FRA	2022-12-30	158385	11.11	1073
France	FRA	2022-12-29	158270	11.11	1072
Germany	DEU	2022-12-31	161465	11.11	1074
Germany	DEU	2022-12-30	161465	11.11	1073
Germany	DEU	2022-12-29	161321	11.11	1072
Ireland	IRL	2022-12-31	8293	5.56	1074
Ireland	IRL	2022-12-30	8293	5.56	1073
Ireland	IRL	2022-12-29	8293	5.56	1072
Italy	ITA	2022-12-31	184642	21.99	1074
Italy	ITA	2022-12-30	184642	21.99	1073
Italy	ITA	2022-12-29	183936	21.99	1072
Netherlands	NLD	2022-12-31	22990	11.11	1074
Netherlands	NLD	2022-12-30	22990	11.11	1073
Netherlands	NLD	2022-12-29	22971	11.11	1072
Spain	ESP	2022-12-31	117095	5.56	1074
Spain	ESP	2022-12-30	117095	5.56	1073
Spain	ESP	2022-12-29	116899	5.56	1072
Sweden	SWE	2022-12-31	21827	11.11	1074
Sweden	SWE	2022-12-30	21827	11.11	1073
Sweden	SWE	2022-12-29	21827	11.11	1072
United Kingdom	GBR	2022-12-31	216306	5.56	1074
United Kingdom	GBR	2022-12-30	216155	5.56	1073
United Kingdom	GBR	2022-12-29	216005	5.56	1072

```
  unique()
country_codes
```

```
# A tibble: 10 x 2
   country        country_code
   <chr>          <chr>
 1 Belgium        BEL
 2 Germany        DEU
 3 Denmark        DNK
 4 Spain          ESP
 5 France         FRA
 6 United Kingdom GBR
 7 Ireland        IRL
 8 Italy          ITA
 9 Netherlands    NLD
10 Sweden         SWE
```

4.1.2 Population

Let us define rough values for the population of each country:

```
population <-
  tribble(
    ~country, ~pop,
    "France", 70e6,
    "Germany", 83e6,
    "Italy", 61e6,
    "Spain", 47e6,
    "United Kingdom", 67e6,
    "Belgium", 12e6,
    "Denmark", 6e6,
    "Ireland", 5e6,
    "Netherlands", 18e6,
    "Sweden", 11e6
  )
```

4.1.3 Filtering data

In this notebook, let us use data up to September 30th, 2020. Let us keep only those data and extend the date up to November 3rd, 2020 (to assess the goodness of fit of the models with unseen data).

```r
end_date_sample <- lubridate::ymd("2020-09-30")
end_date_data <- lubridate::ymd("2020-10-31")
confirmed_df <- confirmed_df |> filter(date <= end_date_data)
deaths_df <- deaths_df |> filter(date <= end_date_data)
```

4.1.4 Defining start and end dates for a "first wave"

We create a table that gives some dates for each country, adopting the following convention:

- `start_first_wave`: start of the first wave, defined as the first date when the cumulative number of cases is greater than 1
- `start_high_stringency`: date at which the stringency index reaches its maximum value during the first 100 days of the sample
- `start_reduce_restrict`: moment at which the restrictions of the first wave starts to lower
- `start_date_sample_second_wave`: 60 days after the relaxation of restrictions (60 days after after `start_reduce_restrict`)
- `length_high_stringency`: number of days between `start_high_stringency` and start_reduce_restrict'.

```r
# Start of the outbreak
start_first_wave <-
  confirmed_df |>
  group_by(country) |>
  arrange(date) |>
  filter(value > 0) |>
  slice(1) |>
  select(country, start_first_wave = date)
```

The start of period with highest severity index among the first 100 days:

```r
start_high_stringency <-
  confirmed_df |>
  group_by(country) |>
  slice(1:100) |>
  arrange(desc(stringency_index), date) |>
  slice(1) |>
  select(country, start_high_stringency = date)
```

The moment at which the restrictions of the first wave starts to lower:

```
start_reduce_restrict <-
  confirmed_df |>
  group_by(country) |>
  arrange(date) |>
  left_join(start_high_stringency, by = "country") |>
  filter(date >= start_high_stringency) |>
  mutate(tmp = dplyr::lag(stringency_index)) |>
  mutate(same_strin = stringency_index == tmp) |>
  mutate(same_strin = ifelse(row_number()==1, TRUE, same_strin)) |>
  filter(same_strin == FALSE) |>
  slice(1) |>
  select(country, start_reduce_restrict = date)
```

The assumed start of the second wave:

```
start_date_sample_second_wave <-
  start_reduce_restrict |>
  mutate(
    start_date_sample_second_wave = start_reduce_restrict +
      lubridate::ddays(60)
  ) |>
  select(country, start_date_sample_second_wave)
```

Then, we can put all these dates into a single table:

```
stringency_dates <-
  start_first_wave |>
  left_join(start_high_stringency, by = "country") |>
  left_join(start_reduce_restrict, by = "country") |>
  left_join(start_date_sample_second_wave, by = "country")
```

4.2 Individual Data

Li et al. (2020) estimated the distribution of the incubation period by fitting a lognormal distribution on exposure histories, leading to an estimated mean of 5.2 days, a 0.95 confidence interval of [4.1,7.0] and the 95th percentile of 12.5 days.

This corresponds to a lognormal density with parameters $\mu = 1.43$ and $\sigma = 0.67$:

```
func <- function(x){
  # For finding gamma parameters of the serial interval distribution Li etal 2020
  a = x[1]
  s = x[2]
  (7.5-a*s)^2+(3.4-sqrt(a)*s)^2
}
x = c(4,2)
optim(x,func)
```

```
$par
[1] 4.866009 1.541308

$value
[1] 9.522571e-10

$counts
function gradient
      67       NA

$convergence
[1] 0

$message
NULL
```

The mean serial interval is defined as the time between the onset of symptoms in a primary case and the onset of symptoms in secondary cases. Li et al. (2020) fitted a gamma distribution to data from cluster investigations. They found a mean time of 7.5 days (SD = 3.4) with a 95% confidence interval of [5.3,19].

The corresponding parameters for the Gamma distribution can be obtained as follows:

```
mean_si <- 7.5
std_si <- 3.4

shape <- (mean_si / std_si)^2
scale <- mean_si / shape
str_c("Shape = ", shape, ", Scale = ", scale)
```

```
[1] "Shape = 4.8659169550173, Scale = 1.54133333333333"
```

The quantile of order .999 for such parameters is equal to:

```
h <- ceiling(qgamma(p = .99, shape = shape, scale = scale))
h
```

```
[1] 18
```

4.3 Descriptive statistics

Let us assign a colour to each country within each group (large or small countries)

```
colours_lugami <- rep(
  c("#1F78B4", "#33A02C", "#E31A1C", "#FF7F00", "#6A3D9A"), 2)
colours_lugami_names <- colours_lugami
names(colours_lugami_names) <- names_countries
colour_table <-
  tibble(
    colour = colours_lugami_names,
    country = names(colours_lugami_names)) |>
  left_join(country_codes, by = c("country"))
colour_table
```

```
# A tibble: 10 x 3
   colour  country        country_code
   <chr>   <chr>          <chr>
 1 #1F78B4 United Kingdom GBR
 2 #33A02C Spain          ESP
 3 #E31A1C Italy          ITA
 4 #FF7F00 Germany        DEU
 5 #6A3D9A France         FRA
 6 #1F78B4 Sweden         SWE
 7 #33A02C Belgium        BEL
 8 #E31A1C Netherlands    NLD
 9 #FF7F00 Ireland        IRL
10 #6A3D9A Denmark        DNK
```

Keep also a track of that in a vector (useful for graphs with {ggplot2}):

```
colour_countries <- colour_table$colour
names(colour_countries) <- colour_table$country_code
colour_countries
```

```
      GBR       ESP       ITA       DEU       FRA       SWE       BEL       NLD 
"#1F78B4" "#33A02C" "#E31A1C" "#FF7F00" "#6A3D9A" "#1F78B4" "#33A02C" "#E31A1C" 
      IRL       DNK 
"#FF7F00" "#6A3D9A" 
```

Let us create two figures that shows the evolution of the cumulative number of cases through time and the cumulative number of deaths through time, respectively.

To that end, we need to reshape the data. First, we need a table in which each row gives the value to be plotted for a given date, a given country and a given type of variable (confirmed cases or recovered).

```
df_plot_evolution_numbers <-
  confirmed_df |>
  mutate(type = "Cases") |>
  bind_rows(
    deaths_df |>
      mutate(type = "Deaths")
  ) |>
  mutate(type = factor(type, levels = c("Cases", "Deaths", "Recovered")))
df_plot_evolution_numbers
```

```
# A tibble: 6,100 x 7
   country country_code date       value stringency_index days_since_2020_01_22
   <chr>   <chr>        <date>     <int>            <dbl>                 <dbl>
 1 Belgium BEL          2020-01-01    NA                0                   -21
 2 Belgium BEL          2020-01-02    NA                0                   -20
 3 Belgium BEL          2020-01-03    NA                0                   -19
 4 Belgium BEL          2020-01-04    NA                0                   -18
 5 Belgium BEL          2020-01-05    NA                0                   -17
 6 Belgium BEL          2020-01-06    NA                0                   -16
 7 Belgium BEL          2020-01-07    NA                0                   -15
 8 Belgium BEL          2020-01-08    NA                0                   -14
 9 Belgium BEL          2020-01-09    NA                0                   -13
10 Belgium BEL          2020-01-10    NA                0                   -12
# i 6,090 more rows
# i 1 more variable: type <fct>
```

From this table, we can only keep cases and deaths cumulative values, filter the observation to keep only those after January 21, 2020. Let us also add for each line, whether the country it corresponds to is small or large (we will create two panels in the graphs). We can also add the country code.

```r
df_plot_evolution_numbers_dates <-
  df_plot_evolution_numbers |>
  filter(type %in% c("Cases", "Deaths")) |>
  filter(date >= ymd("2020-01-21")) |>
  mutate(
    country_type = ifelse(country %in% names_countries_large, yes = "L", "S"),
    country_type = factor(
      country_type,
      levels = c("L", "S"),
      labels = c("Larger countries", "Smaller countries")
    )
  ) |>
  left_join(country_codes, by = c("country", "country_code"))
df_plot_evolution_numbers_dates
```

```
# A tibble: 5,700 x 8
   country country_code date       value stringency_index days_since_2020_01_22
   <chr>   <chr>        <date>     <int>            <dbl>                 <dbl>
 1 Belgium BEL          2020-01-21    NA             0                       -1
 2 Belgium BEL          2020-01-22     0             0                        0
 3 Belgium BEL          2020-01-23     0             0                        1
 4 Belgium BEL          2020-01-24     0             0                        2
 5 Belgium BEL          2020-01-25     0             0                        3
 6 Belgium BEL          2020-01-26     0             0                        4
 7 Belgium BEL          2020-01-27     0             0                        5
 8 Belgium BEL          2020-01-28     0             5.56                     6
 9 Belgium BEL          2020-01-29     0             5.56                     7
10 Belgium BEL          2020-01-30     0             5.56                     8
# i 5,690 more rows
# i 2 more variables: type <fct>, country_type <fct>
```

We want the labels of the legends for the countries to be ordered in a specific way. To that end, let us create two variables that specify this order:

```r
order_countries_L <- c("GBR", "ESP", "ITA", "DEU", "FRA")
order_countries_S <- c("SWE", "BEL", "NLD", "IRL", "DNK")
```

4.3.1 Confirmed cases

Let us focus on the confirmed cases here. We can create two tables: one for the large countries, and another one for small ones:

```r
df_plot_evolution_numbers_dates_L <-
  df_plot_evolution_numbers_dates |>
  filter(country_type == "Larger countries") |>
  filter(type == "Cases")
```

And for small countries:

```r
df_plot_evolution_numbers_dates_S <-
  df_plot_evolution_numbers_dates |>
  filter(country_type == "Smaller countries") |>
  filter(type == "Cases")
```

As we would like the graphs to display the day at which the stringency index reaches its max value for the first time, we need to extract the cumulative number of cases at the corresponding dates.

```r
df_lockdown_plot <-
  stringency_dates |>
  rename(date = start_first_wave) |>
  left_join(
    df_plot_evolution_numbers_dates |>
      filter(type == "Cases"),
    by = c("country", "date")
  )
df_lockdown_plot
```

```
# A tibble: 10 x 11
# Groups:   country [10]
   country        date       start_high_stringency start_reduce_restrict
   <chr>          <date>     <date>                <date>
 1 Belgium        2020-02-04 2020-03-20            2020-05-05
 2 Denmark        2020-02-27 2020-03-18            2020-04-15
 3 France         2020-01-24 2020-03-17            2020-05-11
 4 Germany        2020-01-27 2020-03-22            2020-05-03
 5 Ireland        2020-02-29 2020-04-06            2020-05-18
 6 Italy          2020-01-31 2020-03-20            2020-04-10
 7 Netherlands    2020-02-27 2020-03-23            2020-05-11
 8 Spain          2020-02-01 2020-03-30           2020-05-04
 9 Sweden         2020-02-01 2020-04-01           2020-06-13
10 United Kingdom 2020-01-31 2020-03-24           2020-05-11
# i 7 more variables: start_date_sample_second_wave <date>, country_code <chr>,
#   value <int>, stringency_index <dbl>, days_since_2020_01_22 <dbl>,
#   type <fct>, country_type <fct>
```

The two plots can be created:

```r
plot_evolution_numbers_dates_L <-
  ggplot(
    data = df_plot_evolution_numbers_dates_L |>
      mutate(
        country_code = factor(country_code, levels = order_countries_L)
      ),
    mapping = aes(x = date, y = value, colour = country_code)
  ) +
  geom_line(linewidth = 1.1) +
  geom_point(
    data = df_lockdown_plot |>
      filter(
        country %in% unique(df_plot_evolution_numbers_dates_L$country)
      ),
    colour = "black", size = 4
  ) +
  geom_point(
    data = df_lockdown_plot |>
      filter(country %in% unique(df_plot_evolution_numbers_dates_L$country)),
    mapping = aes(colour = country_code), size = 3, show.legend = F
  ) +
  scale_shape_discrete("Lockdown") +
  scale_fill_manual(NULL, values = colour_countries, guide = "none") +
  scale_colour_manual(NULL, values = colour_countries) +
  labs(x = NULL, y = NULL) +
  scale_y_continuous(labels = comma) +
  scale_x_date(
    breaks = ymd(pretty_dates(df_plot_evolution_numbers_dates$date, n = 5)),
    date_labels = "%b %d %Y"
  ) +
  theme_paper() +
  guides(colour = guide_legend(nrow = 2, byrow = TRUE))
```

and for small countries:

```r
plot_evolution_numbers_dates_S <-
  ggplot(
    data = df_plot_evolution_numbers_dates_S |>
           mutate(
             country_code = factor(country_code, levels = order_countries_S)
             ),
    mapping = aes(x = date, y = value, colour = country_code)
    ) +
  geom_line(linewidth = 1.1) +
  geom_point(
    data = df_lockdown_plot |>
      filter(country %in% unique(df_plot_evolution_numbers_dates_S$country)),
    colour = "black", size = 4
  ) +
  geom_point(
    data = df_lockdown_plot |>
      filter(country %in% unique(df_plot_evolution_numbers_dates_S$country)),
    mapping = aes(colour = country_code), size = 3, show.legend = F
  ) +
  scale_shape_discrete("Lock-down") +
  scale_fill_manual(NULL, values = colour_countries, guide = "none") +
  scale_colour_manual(NULL, values = colour_countries) +
  labs(x = NULL, y = NULL) +
  scale_y_continuous(labels = comma) +
  scale_x_date(
    breaks = ymd(pretty_dates(df_plot_evolution_numbers_dates$date, n = 5)),
    date_labels = "%b %d %Y") +
  theme_paper() +
  guides(colour = guide_legend(nrow = 2, byrow = TRUE))
```

These two plots can be put together on a single one:

```r
p <-
  arrangeGrob(
    # Row 1
    plot_evolution_numbers_dates_L +
      labs(y = NULL, title = "(a) Confirmed case, large countries"),
    plot_evolution_numbers_dates_S +
      labs(y = NULL, title = "(b) Confirmed cases, small countries."),
    nrow = 1
  ) |>
  as_ggplot()
```

```
Warning: Removed 5 rows containing missing values (`geom_line()`).
Removed 5 rows containing missing values (`geom_line()`).
```

```r
p
```

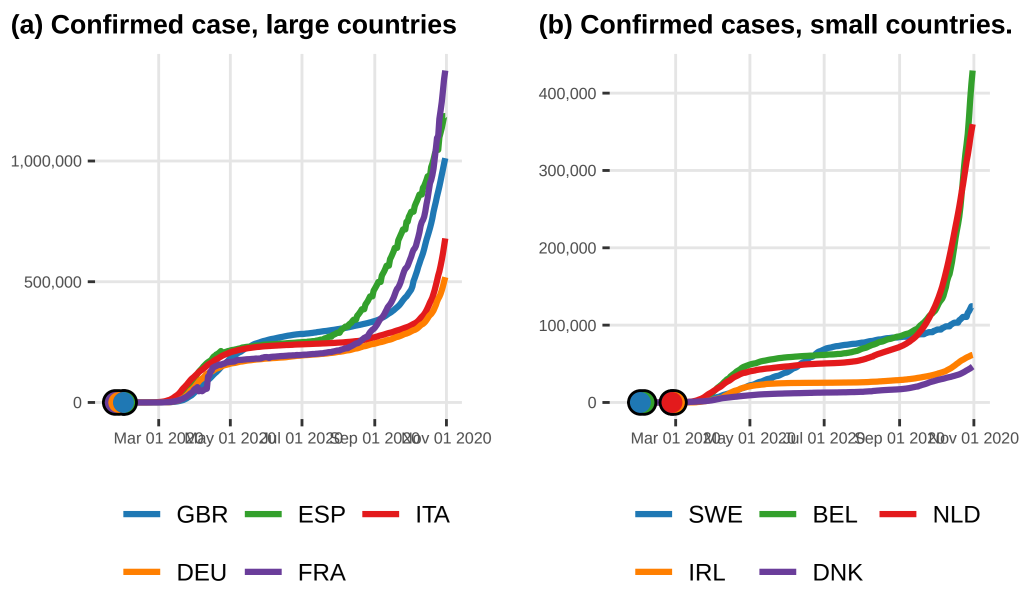


Figure 4.1: Confirmed cases between the beginning of the Covid-19 epidemics and November 2020.

4.3.2 Deaths

Let us focus on the number of deaths here. We can create two tables: one for the large countries, and another one for small ones:

```r
df_plot_evolution_numbers_dates_L <-
  df_plot_evolution_numbers_dates |>
  filter(country_type == "Larger countries") |>
  filter(type == "Deaths")
```

And for small countries:

```r
df_plot_evolution_numbers_dates_S <-
  df_plot_evolution_numbers_dates |>
  filter(country_type == "Smaller countries") |>
  filter(type == "Deaths")
```

As we would like the graphs to display the day at which the stringency index becomes greater or equal to 60 for the first time, we need to extract the cumulative number of cases at the corresponding dates.

```r
df_lockdown_plot <-
  stringency_dates |>
  rename(date = start_first_wave) |>
  left_join(
    df_plot_evolution_numbers_dates |>
      filter(type == "Deaths"),
    by = c("country", "date")
  )
df_lockdown_plot
```

```
# A tibble: 10 x 11
# Groups:   country [10]
   country        date       start_high_stringency start_reduce_restrict
   <chr>          <date>     <date>                <date>
 1 Belgium        2020-02-04 2020-03-20            2020-05-05
 2 Denmark        2020-02-27 2020-03-18            2020-04-15
 3 France         2020-01-24 2020-03-17            2020-05-11
 4 Germany        2020-01-27 2020-03-22            2020-05-03
 5 Ireland        2020-02-29 2020-04-06            2020-05-18
 6 Italy          2020-01-31 2020-03-20            2020-04-10
 7 Netherlands    2020-02-27 2020-03-23            2020-05-11
 8 Spain          2020-02-01 2020-03-30            2020-05-04
 9 Sweden         2020-02-01 2020-04-01            2020-06-13
10 United Kingdom 2020-01-31 2020-03-24            2020-05-11
# i 7 more variables: start_date_sample_second_wave <date>, country_code <chr>,
#   value <int>, stringency_index <dbl>, days_since_2020_01_22 <dbl>,
#   type <fct>, country_type <fct>
```

The two plots can be created:

```r
plot_evolution_numbers_dates_L <-
  ggplot(
    data = df_plot_evolution_numbers_dates_L |>
      mutate(
        country_code = factor(country_code, levels = order_countries_L)
      ),
    mapping = aes(x = date, y = value, colour = country_code)
  ) +
  geom_line(linewidth = 1.1) +
  geom_point(
    data = df_lockdown_plot |>
      filter(country %in% unique(df_plot_evolution_numbers_dates_L$country)),
    colour = "black", size = 4
  ) +
  geom_point(
    data = df_lockdown_plot |>
      filter(country %in% unique(df_plot_evolution_numbers_dates_L$country)),
    mapping = aes(colour = country_code), size = 3, show.legend = F) +
  labs(x = NULL, y = NULL) +
  scale_colour_manual(NULL, values = colour_countries) +
  scale_y_continuous(labels = comma) +
  scale_x_date(
    breaks = ymd(pretty_dates(df_plot_evolution_numbers_dates$date, n = 5)),
    date_labels = "%b %d %Y") +
  theme_paper() +
  guides(colour = guide_legend(nrow = 2, byrow = TRUE))
```

and for small countries:

```r
plot_evolution_numbers_dates_S <-
  ggplot(
    data = df_plot_evolution_numbers_dates_S |>
      mutate(country_code = factor(country_code, levels = order_countries_S)),
    mapping = aes(x = date, y = value, colour = country_code)
  ) +
  geom_line(linewidth = 1.1) +
  geom_point(
    data = df_lockdown_plot |>
      filter(country %in% unique(df_plot_evolution_numbers_dates_S$country)),
    colour = "black", size = 4) +
  geom_point(
    data = df_lockdown_plot |>
      filter(country %in% unique(df_plot_evolution_numbers_dates_S$country)),
    mapping = aes(colour = country_code), size = 3, show.legend = F
  ) +
  scale_shape_discrete("Lock-down") +
  scale_fill_manual(NULL, values = colour_countries, guide = "none") +
  scale_colour_manual(NULL, values = colour_countries) +
  labs(x = NULL, y = NULL) +
  scale_y_continuous(labels = comma) +
  scale_x_date(
    breaks = ymd(pretty_dates(df_plot_evolution_numbers_dates$date, n = 5)),
    date_labels = "%b %d %Y"
  ) +
  theme_paper() +
  guides(colour = guide_legend(nrow = 2, byrow = TRUE))
```

These two plots can be put together on a single one:

```
p <-
  arrangeGrob(
    # Row 1
    plot_evolution_numbers_dates_L +
      labs(y = NULL, title = "(a) Confirmed deaths, large countries"),
    plot_evolution_numbers_dates_S +
      labs(y = NULL, title = "(b) Confirmed deaths, small countries."),
    nrow = 1
    ) |>
  as_ggplot()
```

```
Warning: Removed 5 rows containing missing values (`geom_line()`).
Removed 5 rows containing missing values (`geom_line()`).
```

```
p
```

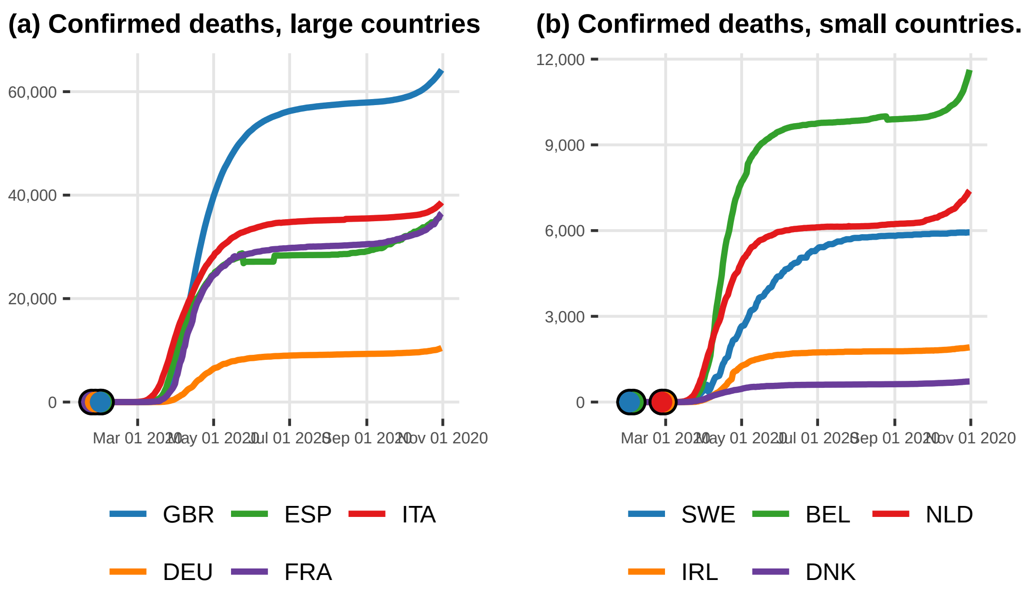


Figure 4.2: Confirmed deaths between the beginning of the Covid-19 epidemics and November 2020.

4.3.3 Stringency Index

Let us create a similar plot for the stringency index.

```r
df_plot_stringency_index <-
  confirmed_df |>
  select(country, country_code, date, stringency_index) |>
  mutate(
    country_type = ifelse(
      country %in% c("France", "Germany", "Italy",
                     "Spain", "United Kingdom"),
      yes = "L", "S")
  ) %>%
  mutate(
    country_type = factor(
      country_type, levels = c("L", "S"),
      labels = c("Larger countries", "Smaller countries")
    )
  )
```

Then we can create the plot for large countries:

```r
plot_stringency_index_L <-
  ggplot(
    data = df_plot_stringency_index |>
      filter(country_type == "Larger countries") |>
      mutate(country_code = fct_relevel(country_code, order_countries_L)),
    mapping =  aes(x = date, y = stringency_index, colour = country_code)
  ) +
  geom_line(linewidth = 1.1) +
  geom_hline(yintercept = 70, linetype = "dotted") +
  labs(x = NULL, y = NULL) +
  scale_colour_manual(NULL, values = colour_countries) +
  scale_y_continuous(breaks = seq(0, 100, by = 20)) +
  scale_x_date(
    breaks = ymd(pretty_dates(df_plot_stringency_index$date, n = 7)),
    date_labels = "%b %d %Y"
  ) +
  theme_paper() +
  guides(colour = guide_legend(nrow = 2, byrow = TRUE))
```

And for small countries:

```r
plot_stringency_index_S <-
  ggplot(
    data = df_plot_stringency_index |>
      filter(country_type == "Smaller countries") |>
      mutate(country_code = fct_relevel(country_code, order_countries_S)),
    mapping= aes(x = date, y = stringency_index, colour = country_code)
  ) +
  geom_line(linewidth = 1.1) +
  geom_hline(yintercept = 70, linetype = "dotted") +
  labs(x = NULL, y = NULL) +
  scale_colour_manual(NULL, values = colour_countries) +
  scale_y_continuous(breaks = seq(0, 100, by = 20)) +
  scale_x_date(
    breaks = ymd(pretty_dates(df_plot_stringency_index$date, n = 7)),
    date_labels = "%b %d %Y") +
  theme_paper() +
  guides(colour = guide_legend(nrow = 2, byrow = TRUE))
```

Lastly, we can plot these two graphs on a single figure:

```
p <-
  arrangeGrob(
    # Row 1
    plot_stringency_index_L +
      labs(y = NULL, title = "(a) Severity index, large countries"),
    plot_stringency_index_S +
      labs(y = NULL, title = "(b) Severity index, small countries"),
    nrow = 1
  ) |>
  as_ggplot()
p
```

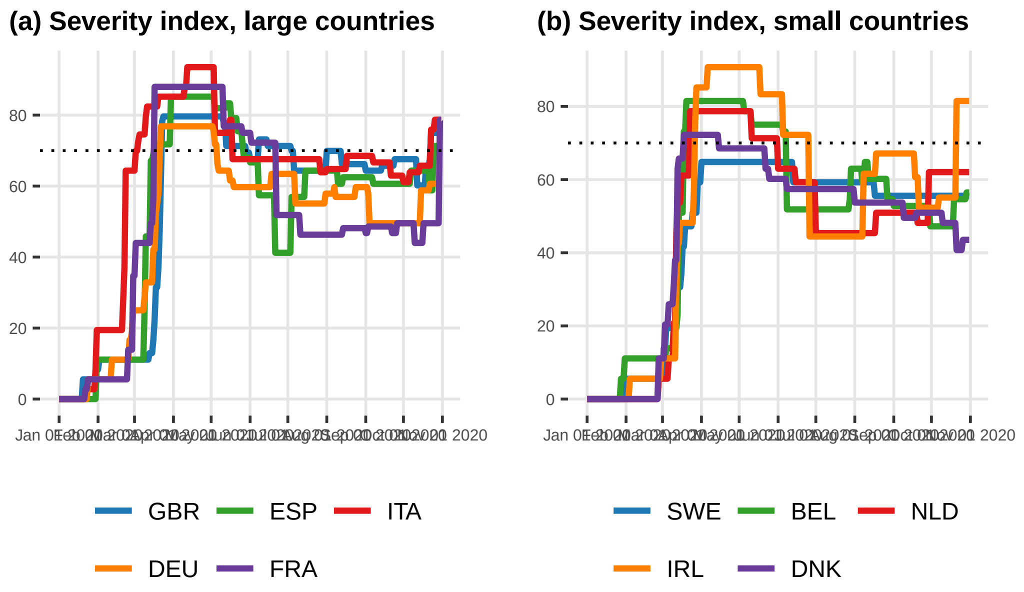


Figure 4.3: Severity index between the beginning of the Covid-19 epidemics and November 2020.

4.3.4 Speed of reaction to the epidemic outbreak

The observation of a first case was the sign that the epidemic had reached the country. What was the delay between this first case and a significant reaction identified when the index was greater than 20?

We can first extract the date on which the severity index reaches the value of 20 for the first time as follows:

```r
start_stringency_20 <-
  confirmed_df |>
  filter(stringency_index >= 20) |>
  group_by(country) |>
  slice(1) |>
  select(country, date_stringency_20 = date, cases = value) |>
  ungroup()
```

Then, we can get the date of the first case for each country:

```r
start_first_case <-
  df_plot_evolution_numbers_dates |>
  filter(type == "Cases") |>
  group_by(country) |>
  filter(value > 0) |>
  arrange(date) |>
  slice(1) |>
  select(country, first_case = date) |>
  ungroup()
```

```r
start_stringency_20 |>
  left_join(start_first_case) |>
  mutate(interval = lubridate::interval(first_case, date_stringency_20),
         delay = interval / lubridate::ddays(1)) |>
  mutate(country = fct_relevel(country, names_countries)) |>
  arrange(country) |>
  select(country, first_case, delay, cases) |>
  kableExtra::kable()
```

```
Joining with `by = join_by(country)`
```

4.3.5 Confinement and deconfinement policies

Let us check how the different countries proceeded with deconfinement.

The date at which the stringency index reached its maximum value within the first 100 days since the end of January:

```r
start_max_stringency <-
  confirmed_df |>
```

Table 4.3: Speed of reaction to the epidemic outbreak

country	first_case	delay	cases
United Kingdom	2020-01-31	46	4452
Spain	2020-02-01	37	1073
Italy	2020-01-31	21	20
Germany	2020-01-27	33	66
France	2020-01-24	36	100
Sweden	2020-02-01	40	771
Belgium	2020-02-04	38	559
Netherlands	2020-02-27	12	503
Ireland	2020-02-29	12	43
Denmark	2020-02-27	5	6

```
  group_by(country) |>
  slice(1:100) |>
  arrange(desc(stringency_index), date) |>
  slice(1) |>
  select(country, start = date)
start_max_stringency
```

```
# A tibble: 10 x 2
# Groups:   country [10]
   country        start
   <chr>          <date>
 1 Belgium        2020-03-20
 2 Denmark        2020-03-18
 3 France         2020-03-17
 4 Germany        2020-03-22
 5 Ireland        2020-04-06
 6 Italy          2020-03-20
 7 Netherlands    2020-03-23
 8 Spain          2020-03-30
 9 Sweden         2020-04-01
10 United Kingdom 2020-03-24
```

We can easily obtain the date on which the index begins to fall from its maximum value, corresponding to a relaxation of policy measures (*i.e.*, end of lockdown):

```
policies <-
  confirmed_df |>
  select(country, date, stringency_index) |>
  left_join(start_max_stringency, by = c("country")) |>
  group_by(country) |>
  arrange(date) |>
  filter(date >= start) |>
  mutate(tmp = dplyr::lag(stringency_index)) |>
  mutate(same_strin = stringency_index == tmp) |>
  mutate(same_strin = ifelse(row_number()==1, TRUE, same_strin)) |>
  filter(same_strin == FALSE) |>
  slice(1) |>
  mutate(length = lubridate::interval(start, date) / ddays(1)) |>
  select(country, start, end=date, length)
policies
```

```
# A tibble: 10 x 4
# Groups:   country [10]
   country        start      end        length
   <chr>          <date>     <date>      <dbl>
 1 Belgium        2020-03-20 2020-05-05     46
 2 Denmark        2020-03-18 2020-04-15     28
 3 France         2020-03-17 2020-05-11     55
 4 Germany        2020-03-22 2020-05-03     42
 5 Ireland        2020-04-06 2020-05-18     42
 6 Italy          2020-03-20 2020-04-10     21
 7 Netherlands    2020-03-23 2020-05-11     49
 8 Spain          2020-03-30 2020-05-04     35
 9 Sweden         2020-04-01 2020-06-13     73
10 United Kingdom 2020-03-24 2020-05-11     48
```

The average of the strigency index between the max value and the end of containment can be obtained as follows:

```
strength_containment <-
  confirmed_df |>
  select(country, date, stringency_index) |>
  left_join(policies, by = "country") |>
  filter(date >= start, date < end) |>
  group_by(country) |>
  summarise(strength = mean(stringency_index))
```

```r
strength_containment
```

```
# A tibble: 10 x 2
   country        strength
   <chr>             <dbl>
 1 Belgium            81.5
 2 Denmark            72.2
 3 France             88.0
 4 Germany            76.8
 5 Ireland            90.7
 6 Italy              85.2
 7 Netherlands        78.7
 8 Spain              85.2
 9 Sweden             64.8
10 United Kingdom     79.6
```

We can count how many smoothing and how many restrengthening actions were made till the end of our sample:

```r
policy_changes <-
  confirmed_df |>
  select(country, date, stringency_index) |>
  left_join(policies, by = "country") |>
  filter(date >=  end, date <= end_date_sample) |>
  group_by(country) |>
  mutate(tmp = dplyr::lag(stringency_index)) |>
  mutate(tmp = ifelse(row_number()==1, stringency_index, tmp)) |>
  mutate(same_strin = stringency_index == tmp) |>
  mutate(
    smoothing = ifelse(!same_strin & stringency_index < tmp, TRUE, FALSE),
    restrenghtening = ifelse(
      !same_strin & stringency_index > tmp, TRUE, FALSE)
  ) |>
  summarise(
    changes_smo = sum(smoothing),
    changes_res = sum(restrenghtening)
  )
policy_changes
```

```
# A tibble: 10 x 3
   country        changes_smo changes_res
   <chr>                  <int>        <int>
 1 Belgium                    8            3
 2 Denmark                    5            1
 3 France                     6            3
 4 Germany                    9            4
 5 Ireland                    4            2
 6 Italy                      5            4
 7 Netherlands                4            2
 8 Spain                      9            4
 9 Sweden                     1            0
10 United Kingdom             8            4
```

Lastly, we can gather all this information in a single table:

```
policies |>
  left_join(strength_containment, by = "country") |>
  left_join(policy_changes, by = "country") |>
  mutate(country = factor(country)) |>
  mutate(country = fct_relevel(country, names_countries)) |>
  arrange(country) |>
  mutate(
    start = format(start, "%B %d"),
    end = format(end, "%B %d")
  ) |>
  kableExtra::kable()
```

4.4 Saving the results

Let us save the following R objects for later use.

```
save(
  confirmed_df,
  deaths_df,
  population,
  h,
  stringency_dates,
  names_countries,
  names_countries_large,
  names_countries_small,
  order_countries_L,
```

Table 4.4: **?(caption)**

(a)

country	start	end	length	strength	changes_smo	changes_res
United Kingdom	March 24	May 11	48	79.63	8	4
Spain	March 30	May 04	35	85.19	9	4
Italy	March 20	April 10	21	85.19	5	4
Germany	March 22	May 03	42	76.85	9	4
France	March 17	May 11	55	87.96	6	3
Sweden	April 01	June 13	73	64.81	1	0
Belgium	March 20	May 05	46	81.48	8	3
Netherlands	March 23	May 11	49	78.70	4	2
Ireland	April 06	May 18	42	90.74	4	2
Denmark	March 18	April 15	28	72.22	5	1

Overview of the smoothing and restrenghthening actions during between the beginning of the Codiv-19 epidemic and November 2020.

```
    order_countries_S,
    colour_countries,
    colour_table,
    country_codes,
    theme_paper,
    file = "data/data_after_load.rda"
    )
```

5 Estimating the reproduction number

In this chapter, we provide some codes to estimate the reproduction number $\mathcal{R}_0$.

5.1 Load data

```
# FOR UNIX USERS
Sys.setlocale("LC_ALL", "en_US.UTF-8")
```

```
[1] "en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8"
```

```
# FOR WINDOWS USERS
# Sys.setlocale("LC_ALL", "English_United States")
```

Some packages that will be used:

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.2     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```
library(scales)
```

```
Attaching package: 'scales'

The following object is masked from 'package:purrr':

    discard

The following object is masked from 'package:readr':

    col_factor
```

```r
library(minpack.lm)
library(mvtnorm)
```

Let us load the data (results obtained from Chapter 4).

```r
load("data/data_after_load.rda")
```

5.2 Objective

In this section, we present the codes to estimate the reproduction number of the first and second waves using either an **exponential model** or a **generalized exponential model**. Based on those models, we estimate the reproduction number $\mathcal{R}_0$ for both waves.

We consider the following start and end dates for each sample, for each country:

Wave	Start	End
First	Date at which the number of cases exceeds 0	Seven days after the stringency index reaches its max value (during the first 100 days of the epidemics)
Second	Sixty days after the stringency index begins to decrease from its maximum value	September 30, 2020

For Sweden, the severity index does not reach 70. Here, we use the dates from Ireland for the definition of those for Sweden.

For the estimation of the reproduction number $\mathcal{R}_0$, we rely on the estimations made by Li et al. (2020). We propose to set the window size h so that it represents 0.99 of the probability in the Gamma distribution of the serial interval.

```r
# Individual data
# From Li et al. (2020):
mean_si <- 7.5
std_si <- 3.4

shape <- (mean_si / std_si)^2
scale <- mean_si / shape
h <- ceiling(qgamma(p = .99, shape = shape, scale = scale))
h
```

```
[1] 18
```

5.3 Some functions needed for the estimation

Let us create a function that, when provided with the name of the country, returns a table for that country that contains the following columns:

- `start_first_wave`: start of the first wave, defined as the first date when the cumulative number of cases is greater than 1
- `start_high_stringency`: date at which the stringency index reaches 70 for the first time within the first 100 days of the sample (for Sweden, as the index never reached 70, we use the time at which the stringency is at its maximum value for the first time within the same time interval)
- `start_reduce_restrict`: moment at which the restrictions of the first wave starts to lower
- `start_date_sample_second_wave`: 60 days after the relaxation of restrictions (60 days after after `start_reduce_restrict`)
- `length_high_stringency`: number of days between `start_high_stringency` and start_reduce_restrict`.

```r
#' Gives the dates of the different periods (first wave, start of containment, ...)
#' @param country_name name of the country
#' @param type if `"deaths"` returns the number of deaths, otherwise the number of cases
get_dates <- function(country_name) {
  df_country <- confirmed_df |>
    filter(country == !!country_name)
```

```r
# Start of the first wave
start_first_wave <-
  df_country |>
  arrange(date) |>
  filter(value > 0) |>
  slice(1) |>
  magrittr::extract2("date")

# Start of period with severity greater or equal than 70 index among the first 100 days
start_high_stringency <-
  df_country |>
  slice(1:100) |>
  filter(stringency_index >= 70) |>
  slice(1) |>
  magrittr::extract2("date")

# Max for Sweden
if(country_name == "Sweden"){
  start_high_stringency <-
    df_country |>
    slice(1:100) |>
    arrange(desc(stringency_index), date) |>
    slice(1) |>
    magrittr::extract2("date")
}

# Max stringency first 100 days
start_max_stringency <-
  df_country |>
  slice(1:100) |>
  arrange(desc(stringency_index), date) |>
  slice(1) |>
  magrittr::extract2("date")

# Moment at which the restrictions of the first wave starts to lower
start_reduce_restrict <-
  df_country |>
  arrange(date) |>
  filter(date >= start_max_stringency) |>
  mutate(tmp = dplyr::lag(stringency_index)) |>
  mutate(same_strin = stringency_index == tmp) |>
```

Table 5.2: Key dates for each countries

country	start_first_wave	start_high_stringency	start_reduce_restrict	start_date_sample_
United Kingdom	2020-01-31	2020-03-23	2020-05-11	2020-07-10
Spain	2020-02-01	2020-03-17	2020-05-04	2020-07-03
Italy	2020-01-31	2020-03-04	2020-04-10	2020-06-09
Germany	2020-01-27	2020-03-22	2020-05-03	2020-07-02
France	2020-01-24	2020-03-17	2020-05-11	2020-07-10
Sweden	2020-02-01	2020-04-01	2020-06-13	2020-08-12
Belgium	2020-02-04	2020-03-18	2020-05-05	2020-07-04
Netherlands	2020-02-27	2020-03-23	2020-05-11	2020-07-10
Ireland	2020-02-29	2020-03-27	2020-05-18	2020-07-17
Denmark	2020-02-27	2020-03-18	2020-04-15	2020-06-14

```r
    mutate(same_strin = ifelse(row_number()==1, TRUE, same_strin)) |>
    filter(same_strin == FALSE) |>
    slice(1) |>
    magrittr::extract2("date")

  start_date_sample_second_wave <- start_reduce_restrict + lubridate::ddays(60)


  # Length of high stringency period
  length_high_stringency <- lubridate::interval(
    start_high_stringency, start_reduce_restrict) / lubridate::ddays(1)

  tibble(
    country = country_name,
    start_first_wave = start_first_wave,
    start_high_stringency = start_high_stringency,
    start_reduce_restrict = start_reduce_restrict,
    start_date_sample_second_wave = start_date_sample_second_wave,
    length_high_stringency = length_high_stringency
  )
}# End of get_dates()
```

If we apply this function for each of the 10 countries of interest:

```r
map_df(names_countries, get_dates) |>
  kableExtra::kable()
```

Based on those dates, we can create a function that will prepare the dataset that will be used to estimate the exponential model, for each country, for the first wave (`wave="first"`) or for the second (`wave="second"`). This functions returns a list of two elements:

1. The dataset
2. The table which gives the dates obtained with the function `get_dates()`. We add two columns to that table : the start and end date of the sample.

```r
#' Extracts the cases data for a country
#' @param country_name name of the country
#' @param sample
get_cases_country <- function(country_name,
                              sample = c("first", "second")) {
  df_country <-
    confirmed_df |>
    filter(country == !!country_name)
  dates_country <- get_dates(country_name)

  # Maximum of the severity index
  max_severity <- max(df_country$stringency_index, na.rm=TRUE)
  dates_country$max_severity <- max_severity

  if (sample == "first") {
    df_country <-
      df_country |>
      # `out_of_sample_horizon` more days for out-of-sample pred
      filter(
        date >= dates_country$start_first_wave,
        date <= (dates_country$start_high_stringency +
                   lubridate::ddays(7) +
                   lubridate::ddays(out_of_sample_horizon))
      )
  } else {
    df_country <-
      df_country |>
      filter(date >= dates_country$start_date_sample_second_wave)

    # Let us remove the number of cases of the first date of this sample
    # to all observation (translation to 1)
    start_val_cases <- df_country$value[1]
    df_country <-
      df_country |>
      mutate(value = value - start_val_cases + 1)
  }


  # Moving Average for missing values (i.e., for Ireland)
  if (any(is.na(df_country$value))) {
    replacement_values <- round(
      zoo::rollapply(
        df_country$value,
        width=3,
        FUN=function(x) mean(x, na.rm=TRUE),
        by=1, by.column=TRUE, partial=TRUE, fill=NA, align="center"
      )
    )
    # Replace only missing values
    df_country <-
      df_country |>
      mutate(replacement_values = !!replacement_values) |>
      mutate(
        value = ifelse(
          is.na(value),
          yes = replacement_values,
          no = value
        )
      ) |>
      select(-replacement_values)
  }

  df_country <-
    df_country |>
    mutate(t = row_number() - 1) |>
    mutate(y = value)

  dates_country <-
    dates_country |>
    mutate(
      start_sample = first(df_country$date),
      end_sample_in = last(df_country$date) -
        lubridate::ddays(out_of_sample_horizon),
      end_sample_out = last(df_country$date)
    )
  list(df_country = df_country, dates_country = dates_country)
}
```

For example, for the UK:

```
out_of_sample_horizon <- 0 # This variable is explained later
cases_uk <- get_cases_country(
  country_name = "United Kingdom", sample = "first"
)
cases_uk$df_country
```

```
# A tibble: 60 x 8
   country   country_code date       value stringency_index days_since_2020_01_22
   <chr>     <chr>        <date>     <int>            <dbl>                 <dbl>
 1 United ~ GBR          2020-01-31     2             8.33                     9
 2 United ~ GBR          2020-02-01     2             8.33                    10
 3 United ~ GBR          2020-02-02     2            11.1                     11
 4 United ~ GBR          2020-02-03     8            11.1                     12
 5 United ~ GBR          2020-02-04     8            11.1                     13
 6 United ~ GBR          2020-02-05     9            11.1                     14
 7 United ~ GBR          2020-02-06     9            11.1                     15
 8 United ~ GBR          2020-02-07     9            11.1                     16
 9 United ~ GBR          2020-02-08    13            11.1                     17
10 United ~ GBR          2020-02-09    14            11.1                     18
# i 50 more rows
# i 2 more variables: t <dbl>, y <int>
```

```
cases_uk$dates_country
```

```
# A tibble: 1 x 10
  country        start_first_wave start_high_stringency start_reduce_restrict
  <chr>          <date>           <date>                <date>
1 United Kingdom 2020-01-31       2020-03-23            2020-05-11
# i 6 more variables: start_date_sample_second_wave <date>,
#   length_high_stringency <dbl>, max_severity <dbl>, start_sample <date>,
#   end_sample_in <date>, end_sample_out <date>
```

We need to write the prediction function for the exponential model and for the generalized exponential model. To compute the reproduction number $\mathcal{R}_0$, we also need to write down the first derivative of these functions, with respect to the time component (x in the functions).

For the exponential model:

```r
#' Exponential model function
#' @param theta vector of named parameters
#' @param x observation / training example
exponential_f <- function(theta, x) {
  c0 <- theta[["c0"]]
  r <- theta[["r"]]
  c0 * exp(r * x)
}
```

```r
#' Derivative of exponential for R0
#' @param theta vector of coefficients
#' @param x time values
derivative_exponential <- function(theta, x) {
  c0 <- theta[["c0"]]
  r <- theta[["r"]]
  c0 * r * exp(r * x)
}
```

For the generalized exponential model:

```r
#' General Exponential model function
#' @param theta vector of named parameters
#' @param x observation / training example
gen_exponential_f <- function(theta, x) {
  A <- theta[["A"]]
  r <- theta[["r"]]
  alpha <- theta[["alpha"]]
  ((1 - alpha) * r * x + A)^( 1 / (1 - alpha))
}
```

```r
#' Derivative of generalized exponential for R0
derivative_gen_exponential <- function(theta, x) {
  A <- theta[["A"]]
  r <- theta[["r"]]
  alpha <- theta[["alpha"]]
  # r * ( ( 1 - alpha ) * r * x + A )^( alpha / ( 1 - alpha ) )
  numb <- A-alpha*r*x + r*x
  expon <- alpha / ( 1 - alpha )
  r * sign(numb) * abs(numb)^expon
}
```

The effective reproduction number R_t is obtained, with the exponential model as follows (Cori et al. 2013):

$$R_t = \frac{I_t}{\sum_{s=1}^{t} I_{t-s}\omega(s)},$$

An estimation of the reproduction number $\mathcal{R}_0$, can be obtained by truncating this summation:

$$R_t = \frac{I_t}{\sum_{s=1}^{h} I_{t-s}\omega(s)}$$

```
#' R0 for exponential
#'
#' @param ti
#' @param h size of the window
#' @param theta estimated coefficients
#' @param shape @param scale shape and scale parameters of the Gamma distribution
R0_expo <- function(ti,
                    h,
                    theta,
                    shape,
                    scale) {
  s_R <- 0
  for (s in 1:h) {
    s_R <- s_R +
      derivative_exponential(theta, ti-s) *
      dgamma(s, shape = shape, scale = scale)
  }

  R0 <- derivative_exponential(theta, ti) / s_R
  R0
}
```

For the generalized exponential model:

$$R_t = \frac{((1-\alpha)\,r\,t + A)^{\alpha/(1-\alpha)}}{\sum_{s=1}^{h}((1-\alpha)\,r\,(t-s) + A)^{\alpha/(1-\alpha)}\omega(s)}.$$

```r
#' R0 for generalized exponential
#'
#' @param ti
#' @param h size of the window
#' @param theta estimated coefficients
#' @param shape @param scale shape and scale parameters of the Gamma distribution
R0_gen_expo <- function(ti,
                        h,
                        theta,
                        shape,
                        scale) {
  s_R <- 0
  for (s in 1:h) {
    s_R <- s_R +
      derivative_gen_exponential(theta, ti-s) *
      dgamma(s, shape = shape, scale = scale)
  }

  R0 <- derivative_gen_exponential(theta, ti) / s_R
  R0
}
```

Let us define a loss function. We will use that function to try to find the parameters of the model (either the exponential model or the generalized exponential model) which minimize it. Note that we use the `nls.lm()` function from {minpack.lm}; hence we only need to compute the residuals and not the residual sum of square.

```r
#' Loss function
#'
#' @param theta vector of named parameters of the model
#' @param fun prediction function of the model
#' @param y target variable
#' @param t time component (feature)
loss_function <- function(theta,
                          fun,
                          y,
                          t) {
  (y - fun(theta = theta, x = t))
}
```

Once the model are estimated, we can compute some goodness of fit criteria. Let us create a function that computes the AIC, the BIC and the RMSE for a specific model. The function expects three arguments: the prediction function of the model (`f`), the values for the parameters of the model (in a named vector – `theta`), and the observations (`data`).

```r
#' Compute some goodness of fit criteria
#'
#' @param f prediction function of the model
#' @param data data that contains the two columns `y` and `t`
#' @param theta estimated coefficients for the model (used in `f`)
get_criteria <- function(f,
                         data,
                         theta) {
  n <- nrow(data)
  k <- length(theta)
  w <- rep(1, n)

  errors <- loss_function(
    theta = theta,
    fun = f,
    y = data$y,
    t = data$t
  )

  mse <- sum(errors^2) / n
  rmse <- sqrt(mse)

  # Log-likelihood
  ll <- 0.5 *
    (sum(log(w)) - n *
       (log(2 * pi) + 1 - log(n) + log(sum(w * errors^2)))
    )
  aic <- 2 * (k + 1) - 2 * ll
  bic <- -2 * ll + log(n) * (k + 1)

  c(AIC = aic, BIC = bic, RMSE = rmse)
}
```

Lastly, to get a confidence interval for the estimated reproduction number $\mathcal{R}_0$, we create a function that performs simulations. From the estimated exponential model (or generalized exponential model), we compute the variance-covariance matrix and then randomly draw `n_repl` observations from a multivariate normal distribution. Based on these simulated numbers, we estimate the reproduction number using the `R0_expo()` or `R0_gen_expo()` function previously defined. We finally compute the average $\mathcal{R}_0$ and its standard deviation based on the `n_repl` simulations.

```
#' Compute variance-covariance matrix from nls.lm
#' Simulate a Normal and compute the corresponding $\mathcal{R}_0$
#'
#' @param out result of nls.lm estimation
#' @param n_repl numbr of desired replications (default to 1,000)
#' @param ti
#' @param h window length
#' @param model_name if `"Exponential"` then uses the Exponential model formula.
#'  Otherwise, the General Exponential one.
sim_ec <- function(out,
                   n_repl = 1000,
                   ti,
                   h,
                   model_name = c("Exponential", "Gen_Exponential")) {

  ibb    <- chol(out$hessian)
  ih     <- chol2inv(ibb)
  p      <- length(out$par)
  rdf    <- length(out$fvec) - p
  resvar <- out$deviance/rdf
  se     <- sqrt(diag(ih) * resvar)
  mean   <- out$par

  Sigv <- ih*resvar
  the  <- rmvnorm(n = n_repl, mean = unlist(mean), sigma = Sigv)

  Ro <- rep(0,n_repl)
  if (model_name == "Exponential") {
    for (i in 1:n_repl) {
      Ro[i] <- R0_expo(
        ti = ti,
        h = h,
        theta = the[i,],
        shape = shape,
        scale = scale
      )
    }
  }else{
    for (i in 1:n_repl) {
      Ro[i] <- R0_gen_expo(
        ti = ti,
        h = h,
        theta = the[i,],
        shape = shape,
        scale = scale
      )
    }
  }

  R0_mu <- mean(Ro)
  R0_sd <- sd(Ro)

  c(R0_mu = R0_mu, R0_sd = R0_sd)
}
```

5.4 Example with only one country

Let us estimate an exponential model first and then a generalized exponential model on the number of cases for one country, United Kingdom. Then we can create a wraping function to apply the codes to all countries.

```
country_name <- "United Kingdom"
```

5.4.1 Exponential model

```
model_function <- exponential_f
```

We need to get the data that contain the number of cases for the UK. The previously defined `get_cases_country()` function can be used:

```
cases_country <- get_cases_country(country_name, sample = "first")
df_country <- cases_country$df_country
dates_country <- cases_country$dates_country
cases_country
```

```
$df_country
# A tibble: 60 x 8
   country  country_code date       value stringency_index days_since_2020_01_22
   <chr>    <chr>        <date>     <int>            <dbl>                <dbl>
```

```
 1 United ~ GBR          2020-01-31     2            8.33                     9
 2 United ~ GBR          2020-02-01     2            8.33                    10
 3 United ~ GBR          2020-02-02     2           11.1                     11
 4 United ~ GBR          2020-02-03     8           11.1                     12
 5 United ~ GBR          2020-02-04     8           11.1                     13
 6 United ~ GBR          2020-02-05     9           11.1                     14
 7 United ~ GBR          2020-02-06     9           11.1                     15
 8 United ~ GBR          2020-02-07     9           11.1                     16
 9 United ~ GBR          2020-02-08    13           11.1                     17
10 United ~ GBR          2020-02-09    14           11.1                     18
# i 50 more rows
# i 2 more variables: t <dbl>, y <int>

$dates_country
# A tibble: 1 x 10
  country        start_first_wave start_high_stringency start_reduce_restrict
  <chr>          <date>           <date>                <date>
1 United Kingdom 2020-01-31       2020-03-23            2020-05-11
# i 6 more variables: start_date_sample_second_wave <date>,
#   length_high_stringency <dbl>, max_severity <dbl>, start_sample <date>,
#   end_sample_in <date>, end_sample_out <date>
```

```r
dates_country
```

```
# A tibble: 1 x 10
  country        start_first_wave start_high_stringency start_reduce_restrict
  <chr>          <date>           <date>                <date>
1 United Kingdom 2020-01-31       2020-03-23            2020-05-11
# i 6 more variables: start_date_sample_second_wave <date>,
#   length_high_stringency <dbl>, max_severity <dbl>, start_sample <date>,
#   end_sample_in <date>, end_sample_out <date>
```

Here are some starting values for the optimization algorithm:

```r
# The starting values
start <- list(
  c0 = 1,
  r  = .14
)
```

The function we want to minimize is the loss function, previously defined in `loss_function()`. It expects four arguments:

- `theta`: a vector of named coefficients
- `fun`: a prediction function (for the exponential model, we pass on the function `exponential_f()`)
- `y`: a vector of observed valued
- `t`: a vector containing the time component.

The `nls.lm()` function can then be used. We provide the starting values to the `par` argument. The `fn` argument is provided with the function to minimize. We also set the `maxiter` component of the `control` argument to `100` (maximum number of iterations). The argument `y`, `t` and `fun` are directly passed on to the arguments of the function given to the `fn` argument.

```
# The estimated coefficients
out <- nls.lm(
  par = start,
  fn = loss_function,
  y = df_country$y,
  t = df_country$t,
  fun = model_function,
  control = nls.lm.control(maxiter = 100),
  jac = NULL, lower = NULL, upper = NULL
)
```

The results can be summarized as follows:

```
summary(out)
```

```

Parameters:
   Estimate Std. Error t value Pr(>|t|)
c0 4.542031   0.587775   7.727 1.76e-10 ***
r  0.151973   0.002299  66.092  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 523.9 on 58 degrees of freedom
Number of iterations to termination: 9
Reason for termination: Relative error in the sum of squares is at most `ftol'.
```

The estimated coefficients can be extracted and saved in a tibble:

```
params <- tibble(
  model_type = "Exponential",
```

```r
    country = country_name,
    coef_estimate_name = names(coef(out)),
    coef_estimate = coef(out)
  )
  params
```

```
# A tibble: 2 x 4
  model_type  country        coef_estimate_name coef_estimate
  <chr>       <chr>          <chr>                      <dbl>
1 Exponential United Kingdom c0                         4.54
2 Exponential United Kingdom r                          0.152
```

The goodness of fit criterion can be computed using the `get_criteria()` function previously defined.

```r
crit <- get_criteria(
  f = model_function,
  data = df_country,
  theta = params$coef_estimate
)
crit
```

```
     AIC      BIC     RMSE
925.5892 931.8723 515.0712
```

And they can be stored in a tibble:

```r
criteria <-
  tibble(
    model_type = "Exponential",
    country = country_name,
    bind_rows(crit)
  )
criteria
```

```
# A tibble: 1 x 5
  model_type  country          AIC   BIC  RMSE
  <chr>       <chr>          <dbl> <dbl> <dbl>
1 Exponential United Kingdom  926.  932.  515.
```

The $\mathscr{R}_0$ can be estimated with the `sim_ec()` function:

```r
R0_i <- sim_ec(
  out = out,
  n_repl = 1000,
  ti = h,
  h = h,
  model_name = "Exponential"
)
R0_i
```

```
     R0_mu      R0_sd 
2.78973941 0.03994991 
```

They can also be saved in a tibble:

```r
R0_df <-
  tibble(
    model_type = "Exponential",
    country = country_name,
    bind_rows(R0_i)
  )
R0_df
```

```
# A tibble: 1 x 4
  model_type  country        R0_mu  R0_sd
  <chr>       <chr>          <dbl>  <dbl>
1 Exponential United Kingdom  2.79 0.0399
```

Then, we can plot the observed values and the estimated ones. First, let us get the estimated values, using the obtained parameters:

```r
fitted_val_expo_uk <-
  df_country |>
  mutate(index = row_number()-1) |>
  mutate(
    model_type    = "Exponential",
    fitted_value = model_function(theta = params$coef_estimate, x = index)
  )
fitted_val_expo_uk
```

```
# A tibble: 60 x 11
   country   country_code date       value stringency_index days_since_2020_01_22
   <chr>     <chr>        <date>     <int>            <dbl>                 <dbl>
 1 United ~ GBR          2020-01-31     2             8.33                     9
 2 United ~ GBR          2020-02-01     2             8.33                    10
 3 United ~ GBR          2020-02-02     2            11.1                     11
 4 United ~ GBR          2020-02-03     8            11.1                     12
 5 United ~ GBR          2020-02-04     8            11.1                     13
 6 United ~ GBR          2020-02-05     9            11.1                     14
 7 United ~ GBR          2020-02-06     9            11.1                     15
 8 United ~ GBR          2020-02-07     9            11.1                     16
 9 United ~ GBR          2020-02-08    13            11.1                     17
10 United ~ GBR          2020-02-09    14            11.1                     18
# i 50 more rows
# i 5 more variables: t <dbl>, y <int>, index <dbl>, model_type <chr>,
#   fitted_value <dbl>
```

```
ggplot(
  data = fitted_val_expo_uk |>
    select(date, value, fitted_value) |>
    pivot_longer(cols = c(value, fitted_value)) |>
    mutate(name = factor(name, levels = c("value", "fitted_value"))),
  mapping = aes(x = date, y = value, linetype = name)) +
  geom_line() +
  labs(x = NULL, y = "Cases") +
  scale_y_continuous(labels = comma) +
  scale_linetype_discrete(
    NULL,
    labels = c("value" = "Observed values",
               "fitted_value" = "Fitted values")) +
  theme(
    legend.position = "bottom",
    plot.title.position = "plot"
  )
```

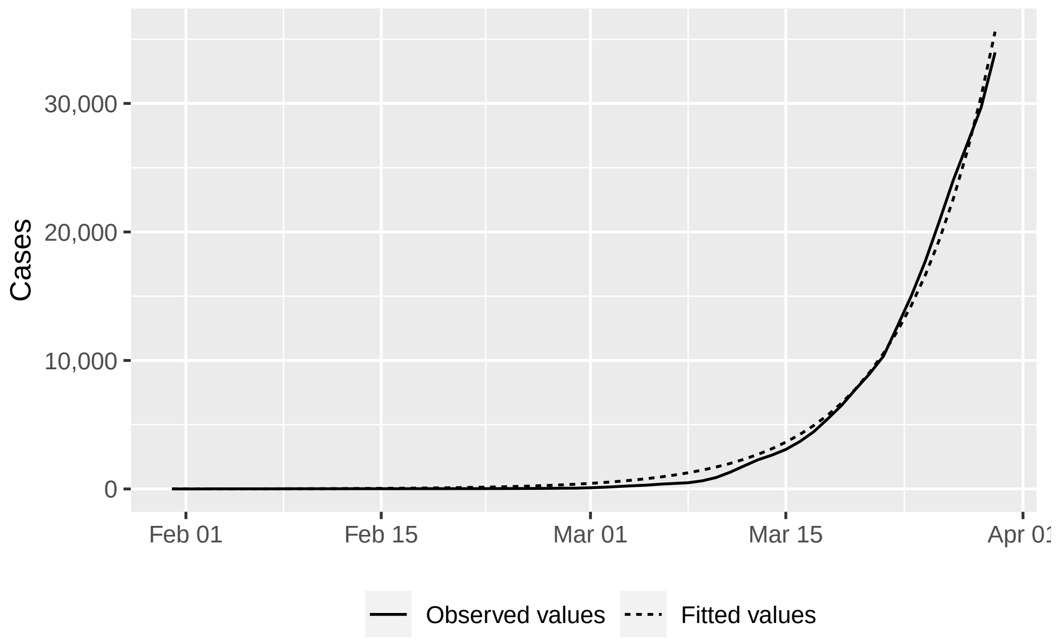


Figure 5.1: Number of cases, Exponential model

We can compute the predicted values up to a given horizon. Let us assume that we want to make predictions up to the 80th day.

```r
horizon_pred <- 80
obs <- df_country$y

type_obs <- rep("obs", length(obs))
if (length(obs) < horizon_pred) {
  obs <- c(obs, rep(NA, horizon_pred-length(obs)))
  type_obs <- c(
    type_obs,
    rep("out_of_sample", horizon_pred-length(type_obs))
  )
}

length(obs)
```

```
[1] 80
```

```r
table(type_obs)
```

```
type_obs
         obs out_of_sample
          60            20
```

Let us keep track on the corresponding dates.

```
dates <- df_country$date
if (length(dates) < horizon_pred) {
  dates <- dates[1] + lubridate::ddays(seq_len(horizon_pred) - 1)
}
tail(dates)
```

```
[1] "2020-04-14" "2020-04-15" "2020-04-16" "2020-04-17" "2020-04-18"
[6] "2020-04-19"
```

The predictions can be made, using the estimated parameters, and stored in a tibble.

```
fitted_val_expo_uk_80 <- tibble(
  country  = country_name,
  index    = seq_len(horizon_pred) - 1,
  value    = obs,
  type_obs = type_obs,
  date     = dates
) |>
  mutate(
    model_type   = "Exponential",
    fitted_value = model_function(theta = params$coef_estimate, x = index)
  )
fitted_val_expo_uk_80
```

```
# A tibble: 80 x 7
   country        index value type_obs date       model_type  fitted_value
   <chr>          <dbl> <int> <chr>    <date>     <chr>              <dbl>
 1 United Kingdom     0     2 obs      2020-01-31 Exponential         4.54
 2 United Kingdom     1     2 obs      2020-02-01 Exponential         5.29
 3 United Kingdom     2     2 obs      2020-02-02 Exponential         6.16
 4 United Kingdom     3     8 obs      2020-02-03 Exponential         7.17
 5 United Kingdom     4     8 obs      2020-02-04 Exponential         8.34
 6 United Kingdom     5     9 obs      2020-02-05 Exponential         9.71
 7 United Kingdom     6     9 obs      2020-02-06 Exponential        11.3
 8 United Kingdom     7     9 obs      2020-02-07 Exponential        13.2
 9 United Kingdom     8     13 obs       2020-02-08 Exponential        15.3
10 United Kingdom     9     14 obs       2020-02-09 Exponential        17.8
# i 70 more rows
```

We can plot those values:

```r
ggplot(
  data = fitted_val_expo_uk_80 |>
    pivot_longer(cols = c(value, fitted_value)),
  mapping = aes(x = date, y = value, linetype = name, colour = type_obs)) +
  geom_line() +
  labs(x = NULL, y = "Cases") +
  scale_y_continuous(labels = comma) +
  scale_linetype_discrete(
    NULL,
    labels = c("value" = "Observed values",
               "fitted_value" = "Fitted values")) +
  scale_colour_manual(
    NULL,
    values = c(
      "obs" = "#44AA99",
      "out_of_sample" = "#AA4499"
    ),
    labels = c(
      "obs" = "Observed",
      "out_of_sample" = "Out-of-sample"
    )
  ) +
  theme(
    legend.position = "bottom",
    plot.title.position = "plot"
  )
```

```
Warning: Removed 20 rows containing missing values (`geom_line()`).
```

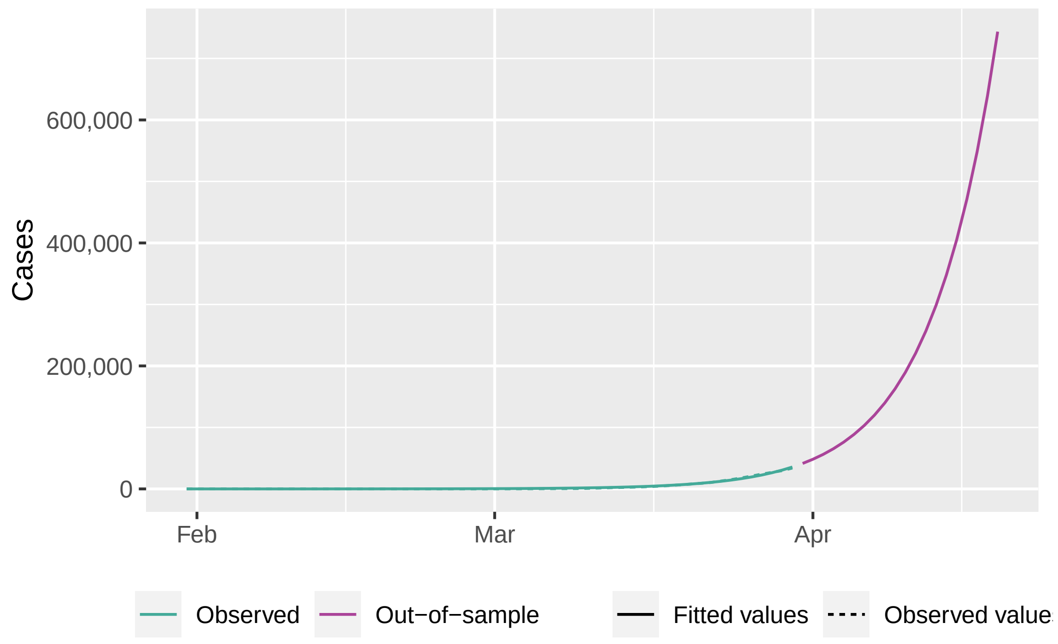


Figure 5.2: Number of cases, Exponential model, out-of-sample predictions

5.4.2 Generalized exponential model

Now let us turn to the generalized exponential model.

```
model_function <- gen_exponential_f
```

Again, we rely on the `nls.lm()` function from package {minpack.lm}.

Here are some starting values for the optimization algorithm:

```
# The starting values
start <- list(
  A = 1,
  r  = .14,
  alpha = .99
)
```

We need to change the prediction function that will be passed on to the `loss_function()` that will be minimized:

The `nls.lm()` function can then be used.

```
# The estimated coefficients
out <- nls.lm(
  par = start,
  fn = loss_function,
  y = df_country$y,
  t = df_country$t,
  fun = model_function,
  control = nls.lm.control(maxiter = 100),
  jac = NULL,
  lower = NULL,
  upper = NULL
)
```

The results can be summarized as follows:

```
summary(out)
```

```
Parameters:
       Estimate Std. Error t value Pr(>|t|)    
A     1.469e-07  9.854e-02    0.00        1    
r     5.011e-01  1.431e-02   35.01   <2e-16 ***
alpha 8.751e-01  2.981e-03  293.51   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 302.5 on 57 degrees of freedom
Number of iterations to termination: 38 
Reason for termination: Relative error between `par' and the solution is at most `ptol'. 
```

The estimated coefficients can be extracted and saved in a tibble:

```
params <- tibble(
  model_type = "Gen_Exponential",
  country = country_name,
  coef_estimate_name = names(coef(out)),
  coef_estimate = coef(out)
)
params
```

```
# A tibble: 3 x 4
```

```
  model_type      country        coef_estimate_name coef_estimate
  <chr>           <chr>          <chr>                      <dbl>
1 Gen_Exponential United Kingdom A                      0.000000147
2 Gen_Exponential United Kingdom r                      0.501
3 Gen_Exponential United Kingdom alpha                  0.875
```

The goodness of fit criterion can be computed using the `get_criteria()` function previously defined.

```
crit <- get_criteria(
  f = model_function,
  data = df_country,
  theta = params$coef_estimate
)
crit
```

```
     AIC      BIC     RMSE
860.6478 869.0251 294.8479
```

And they can be stored in a tibble:

```
criteria <- tibble(
  model_type = "Gen_Exponential",
  country = country_name,
  bind_rows(crit)
)
criteria
```

```
# A tibble: 1 x 5
  model_type      country          AIC   BIC  RMSE
  <chr>           <chr>          <dbl> <dbl> <dbl>
1 Gen_Exponential United Kingdom  861.  869.  295.
```

The $\mathcal{R}_0$ can be estimated with the `sim_ec()` function. But the generalized exponential does not provide good estimates for the reproduction number.

```
R0_i <- sim_ec(
  out = out,
  n_repl = 1000,
  ti = h,
  h = h,
  model_name = "Gen_Exponential"
)
R0_i
```

```
    R0_mu     R0_sd 
13.693533  2.029799 
```

They can also be saved in a tibble:

```
R0_df <- tibble(
  model_type = "Gen_Exponential",
  country = country_name,
  bind_rows(R0_i)
)
R0_df
```

```
# A tibble: 1 x 4
  model_type      country        R0_mu R0_sd
  <chr>           <chr>          <dbl> <dbl>
1 Gen_Exponential United Kingdom  13.7  2.03
```

We can compute the predicted values and store those in a tibble:

```
fitted_val_genexpo_uk <-
  df_country |>
  mutate(index = row_number()-1) |>
  mutate(
    model_type   = "Gen_Exponential",
    fitted_value = model_function(theta = params$coef_estimate, x = index)
  )
fitted_val_genexpo_uk
```

```
# A tibble: 60 x 11
   country   country_code date       value stringency_index days_since_2020_01_22
   <chr>     <chr>        <date>     <int>            <dbl>                 <dbl>
 1 United ~ GBR          2020-01-31     2             8.33                     9
 2 United ~ GBR          2020-02-01     2             8.33                    10
 3 United ~ GBR          2020-02-02     2            11.1                     11
 4 United ~ GBR          2020-02-03     8            11.1                        12
 5 United ~ GBR          2020-02-04     8            11.1                        13
 6 United ~ GBR          2020-02-05     9            11.1                        14
 7 United ~ GBR          2020-02-06     9            11.1                        15
 8 United ~ GBR          2020-02-07     9            11.1                        16
 9 United ~ GBR          2020-02-08    13            11.1                        17
10 United ~ GBR          2020-02-09    14            11.1                        18
# i 50 more rows
# i 5 more variables: t <dbl>, y <int>, index <dbl>, model_type <chr>,
#   fitted_value <dbl>
```

```
ggplot(
  data = fitted_val_genexpo_uk |>
    select(date, value, fitted_value) |>
    pivot_longer(cols = c(value, fitted_value)) |>
    mutate(name = factor(name, levels = c("value", "fitted_value"))),
  mapping = aes(x = date, y = value, linetype = name)) +
  geom_line() +
  labs(x = NULL, y = "Cases") +
  scale_y_continuous(labels = comma) +
  scale_linetype_discrete(
    NULL,
    labels = c("fitted_value" = "Fitted values",
               "value" = "Observed values")
  ) +
  theme(
    legend.position = "bottom",
    plot.title.position = "plot"
  )
```

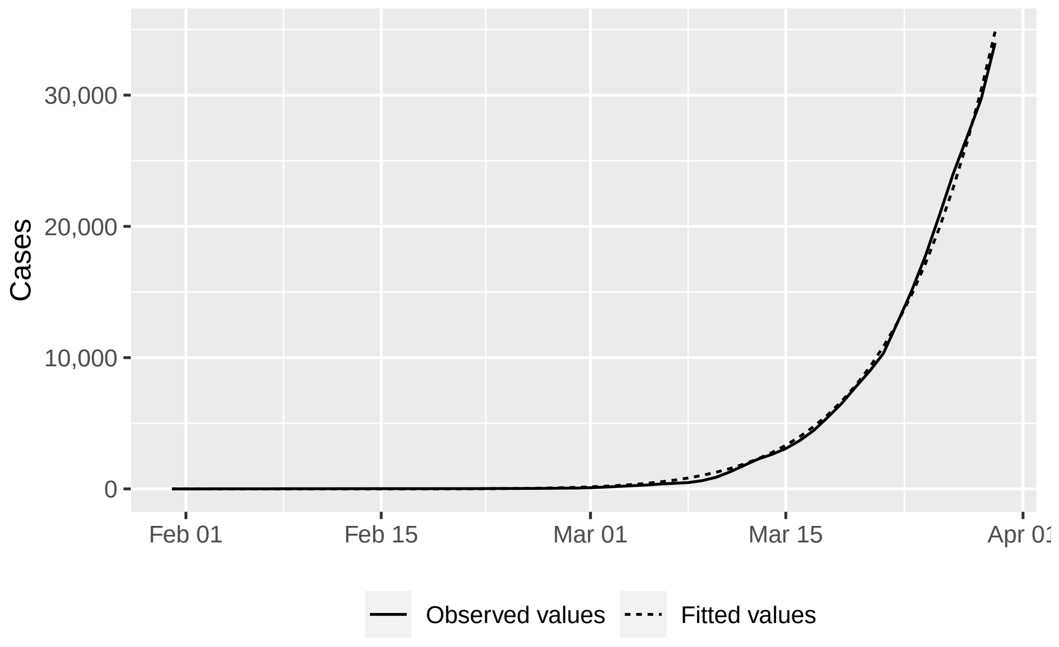


Figure 5.3: Number of cases, Generalized Exponential Model

We can compute the predicted values up to a given horizon. Let us assume that we want to make predictions up to the 80th day.

```r
horizon_pred <- 80
obs <- df_country$y

type_obs <- rep("obs", length(obs))
if (length(obs) < horizon_pred) {
  obs <- c(obs, rep(NA, horizon_pred-length(obs)))
  type_obs <- c(
    type_obs,
    rep("out_of_sample", horizon_pred-length(type_obs))
  )
}

length(obs)
```

```
[1] 80
```

```r
table(type_obs)
```

```
type_obs
          obs out_of_sample
           60            20
```

Let us keep track on the corresponding dates.

```r
dates <- df_country$date
if (length(dates) < horizon_pred) {
  dates <- dates[1] + lubridate::ddays(seq_len(horizon_pred) - 1)
}
tail(dates)
```

```
[1] "2020-04-14" "2020-04-15" "2020-04-16" "2020-04-17" "2020-04-18"
[6] "2020-04-19"
```

The predictions can be made, using the estimated parameters, and stored in a tibble.

```r
fitted_val_genexpo_uk_80 <- tibble(
  country  = country_name,
  index    = seq_len(horizon_pred) - 1,
  value    = obs,
  type_obs = type_obs,
  date     = dates
) |>
  mutate(
    model_type    = "Gen_Exponential",
    fitted_value = model_function(theta = params$coef_estimate, x = index)
  )
fitted_val_genexpo_uk_80
```

```
# A tibble: 80 x 7
   country        index value type_obs date       model_type      fitted_value
   <chr>          <dbl> <int> <chr>    <date>     <chr>                  <dbl>
 1 United Kingdom     0     2 obs      2020-01-31 Gen_Exponential     1.97e-55
 2 United Kingdom     1     2 obs      2020-02-01 Gen_Exponential     2.32e-10
 3 United Kingdom     2     2 obs      2020-02-02 Gen_Exponential     5.95e- 8
 4 United Kingdom     3     8 obs      2020-02-03 Gen_Exponential     1.53e- 6
 5 United Kingdom     4     8 obs      2020-02-04 Gen_Exponential     1.53e- 5
 6 United Kingdom     5     9 obs      2020-02-05 Gen_Exponential     9.13e- 5
 7 United Kingdom     6     9 obs      2020-02-06 Gen_Exponential     3.93e- 4
 8 United Kingdom     7     9 obs      2020-02-07 Gen_Exponential     1.35e- 3
 9 United Kingdom     8     13 obs       2020-02-08 Gen_Exponential     3.93e- 3
10 United Kingdom     9     14 obs       2020-02-09 Gen_Exponential     1.01e- 2
# i 70 more rows
```

We can plot those values:

```r
ggplot(
  data = fitted_val_genexpo_uk_80 |>
    pivot_longer(cols = c(value, fitted_value)),
  mapping = aes(x = date, y = value, linetype = name, colour = type_obs)) +
  geom_line() +
  labs(x = NULL, y = "Cases") +
  scale_y_continuous(labels = comma) +
  scale_linetype_discrete(
    NULL,
    labels = c("value" = "Observed values",
               "fitted_value" = "Fitted values")) +
  scale_colour_manual(
    NULL,
    values = c(
      "obs" = "#44AA99",
      "out_of_sample" = "#AA4499"
    ),
    labels = c(
      "obs" = "Observed",
      "out_of_sample" = "Out-of-sample"
    )
  ) +
  theme(
    legend.position = "bottom",
    plot.title.position = "plot"
  )
```

```
Warning: Removed 20 rows containing missing values (`geom_line()`).
```

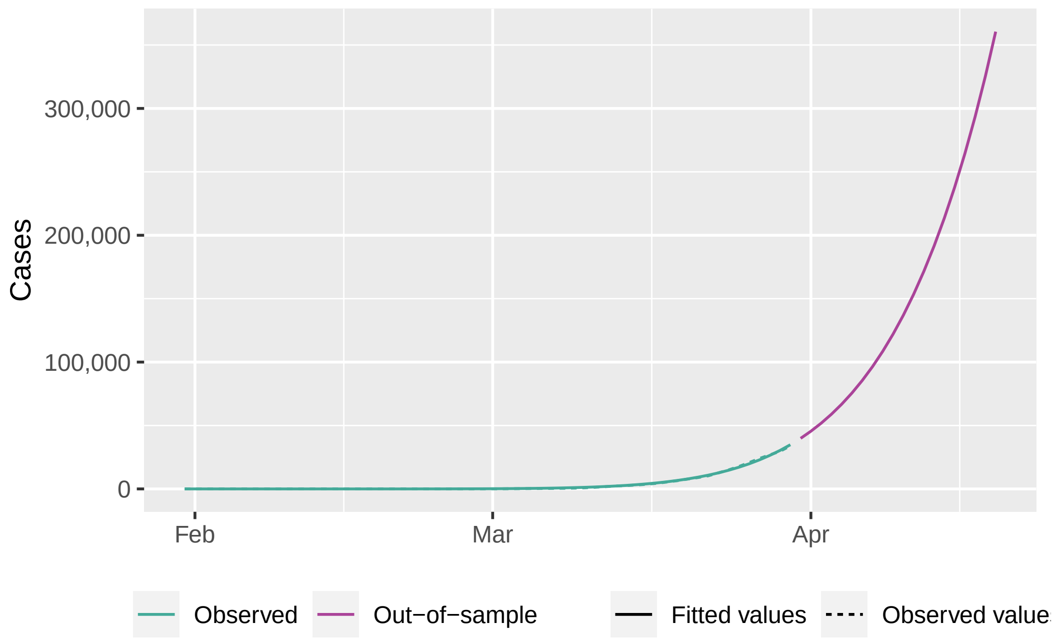


Figure 5.4: Number of cases, Generalized Exponential model, out-of-sample predictions

6 Background on Phenomenological Models

This chapter provides a bit of background about phenomenological models that can be used to model an epidemic.

6.1 Simple phenomenological models

We will present three models to estimate the number of cases and the number of deaths: the logistic model, the Gompertz model, and the Richards model.

6.1.1 A Generic Equation

X.-S. Wang, Wu, and Yang (2012) develop the similarity between the SIR model and the Richards population model Richards (1959) which lead them to consider the following growth equation for confirmed cases noted $C(t)$:

$$\frac{d\,C}{d\,t} = rC^{\alpha}\left[1 - \left(\frac{C}{K}\right)^{\delta}\right], \tag{6.1}$$

with r, α, δ and K being positive real numbers with the further restriction $0 \leq \alpha \leq 1$.

A general solution to this equation has the form:

$$C(t) = F(r, \alpha, \delta, K, t),$$

with the property that $\lim_{t\to\infty} C(t) = K$. If $C(t)$ corresponds to the total number of cases, the number of new cases is found by computing the derivative of $C(t)$ with respect to t and noted $c(t)$. The relative speed of the process is defined as $c(t)/C(t)$ and is constant over time only when $\alpha = 1$, $\delta = 0$. The doubling time is constant over time only under those restrictions.

A crucial question is to characterize the speed at which the process will reach its peak and when. Tsoularis and Wallace (2002) have shown that the value of the peak is given by the value of C at the inflexion point of the curve:

$$C_{inf} = K\left(1 + \frac{\delta}{\alpha}\right)^{-1/\delta}. \tag{6.2}$$

The corresponding time, that we shall note τ is obtained by inverting $C(t)$. For some models, when an analytical expression for $C(t)$ is available, τ can be directly included in the parameterization. This point is of particular importance because it corresponds to the period when the epidemic starts to regress, or equivalently when the effective reproduction number R_t starts to be below 1.

6.1.2 Logistic Model

The logistic model initiated by Verhulst (1845) provides the most simple solution to the Equation 6.1 and corresponds to $\alpha = \delta = 1$:

$$C(t) = \frac{K}{1 + \exp\left(-r(t-\tau)\right)}. \tag{6.3}$$

It has been recently applied to study the evolution of an epidemic Ma (2020). We have introduced a parameterization where τ directly represents the inflection point with that $C(\tau) = K/2$. So the peak is at the mid of the epidemic which reaches its cumulated maximum K, r representing the growth rate. The first order derivative of this function provides an estimate of the number of cases at each point in time:

$$\frac{\partial C(t)}{\partial t} = \frac{Kr\exp\left(-r(t-\tau)\right)}{\left[1+\exp(-r(t-\tau)\right]^2}.$$

The relative speed is thus:

$$\frac{c(t)}{C(t)} = r\frac{e^{-r(t-\tau)}}{1+e^{-r(t-\tau)}}.$$

This model might appear as a nice solution to represents the three phases of an epidemic, but we shall discover below that its symmetry tends to be in contradiction with epidemic data. So we have to explore alternative solutions.

In R, we define the logistic function as follows:

```r
#' Logistic function
#'
#' @param theta vector of named parameters
#' @param x time
logistic_f <-  function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]
  k / ( 1+exp( -r*( x - tau ) ) )
}
```

```r
#' First derivative of the logistic function
#'
#' @param theta vector of named parameters
#' @param x time
logistic_f_first_d <- function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]

  (k * r * exp( -r * (x - tau) )) / (1+ exp( -r * (x - tau) ))^2
}

#' Second derivative of the logistic function
#'
#' @param theta vector of named parameters
#' @param x time
logistic_f_second_d <- function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]

  k*((2*r^2 * exp(-2*r*(x - tau)))/
       (exp(-r*(x - tau)) + 1)^3 -
       (r^2*exp(-r*(x - tau)))/(exp(-r*(x - tau)) + 1)^2)
}
```

Now, let us make some graph to give an idea of how the dynamic changes with the parameters r and tau.

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.2     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
library(scales)
```

```
Attaching package: 'scales'

The following object is masked from 'package:purrr':

    discard

The following object is masked from 'package:readr':

    col_factor
```

```r
g_legend<-function(a.gplot){
  tmp <- ggplot_gtable(ggplot_build(a.gplot))
  leg <- which(sapply(tmp$grobs, function(x) x$name) == "guide-box")
  legend <- tmp$grobs[[leg]]
  return(legend)}
```

The different values for r and tau (k is set to 5000).

```r
situations <-
  expand_grid(tau = c(25,35), r = c(.15, .25, .35)) |>
  mutate(k = 5000)
```

Let us apply, for a set of parameters, at different time horizon (from 0 to 60 by steps of .1) the logistic function, its first and second derivatives with respect to time. To do so, we define a "simulation" function:

```r
#' Simulation of the logistic model for some parameters
#'
#' @param i row number of situations
simu <- function(i) {
  current_params <- situations |> slice(i)
  current_params <- c(as_vector(current_params)) |> as.list()

  n     <- 60
  step  <- .01
  sim   <-  logistic_f(
    theta = current_params,
    x = seq(0,n, by = step)
  )
  simD  <-  logistic_f_first_d(
    theta = current_params,
    x = seq(0,n, by = step)
  )
  simD2 <-  logistic_f_second_d(
    theta = current_params,
    x = seq(0,n, by = step)
  )

  tibble(
    t = seq(0, n, by = step),
    k = current_params[["k"]],
    tau = current_params[["tau"]],
    r = current_params[["r"]],
    sim = sim
  ) |>
    mutate(
      simD = simD,
      simD2 = simD2
    )
}
```

Let us do so for all the different sets of parameters:

```r
simu_res <- map_df(1:nrow(situations), simu)
```

We would like to show the thresholds on the graph:

```r
threshold_times <-
  simu_res |>
  group_by(k, tau, r) |>
  summarise(threshold_time = unique(tau)) |>
  ungroup() |>
  mutate(
    r = str_c("$\\r = ", r, "$"),
    tau = str_c("$\\tau = ", tau, "$")
  )
```

```
`summarise()` has grouped output by 'k', 'tau'. You can override using the
`.groups` argument.
```

```r
threshold_times
```

```
# A tibble: 6 x 4
      k tau             r               threshold_time
  <dbl> <chr>           <chr>                    <dbl>
1  5000 "$\\tau = 25$" "$\\r = 0.15$"             25
2  5000 "$\\tau = 25$" "$\\r = 0.25$"             25
3  5000 "$\\tau = 25$" "$\\r = 0.35$"             25
4  5000 "$\\tau = 35$" "$\\r = 0.15$"             35
5  5000 "$\\tau = 35$" "$\\r = 0.25$"             35
6  5000 "$\\tau = 35$" "$\\r = 0.35$"             35
```

Now, we are ready to create the plots:

```r
p_simu_logis <-
  ggplot(
    data =  simu_res |>
      mutate(
        r = str_c("$\\r = ", r, "$"),
        tau = str_c("$\\tau = ", tau, "$")
      ),
    mapping = aes(x = t)
  ) +
  geom_line(mapping = aes(y = sim, linetype = "sim")) +
  geom_line(mapping = aes(y = simD*10, linetype = "simD")) +
  geom_line(mapping = aes(y = simD2*10, linetype = "simD2")) +
  geom_vline(
    data = threshold_times,
    mapping = aes(xintercept = threshold_time),
    colour = "red", linetype = "dotted") +
  facet_grid(
    r~tau,
    labeller = as_labeller(
      latex2exp::TeX,
      default = label_parsed
    )
  ) +
  labs(x = "Time", y = latex2exp::TeX("$F(t)$")) +
  scale_y_continuous(
    labels = scales::comma,
    sec.axis = sec_axis(
      ~./10,
      name = latex2exp::TeX(
        "$\\partial F(t) / \\partial t$, $\\partial^2 F(t) / \\partial t^2$"
      ),
      labels = comma)) +
  scale_linetype_manual(
    NULL,
    values = c("sim" = "solid", "simD" = "dashed", "simD2" = "dotdash"),
    labels = c(
      "sim" = latex2exp::TeX("$F(t)$"),
      "simD" = latex2exp::TeX("$\\partial F(t) / \\partial t$"),
      "simD2" = latex2exp::TeX("$\\partial^2 F(t) / \\partial t^2$"))) +
  theme(axis.ticks.y = element_blank())

p_simu_logis +
  theme_minimal() +
  theme(legend.position = "bottom")
```

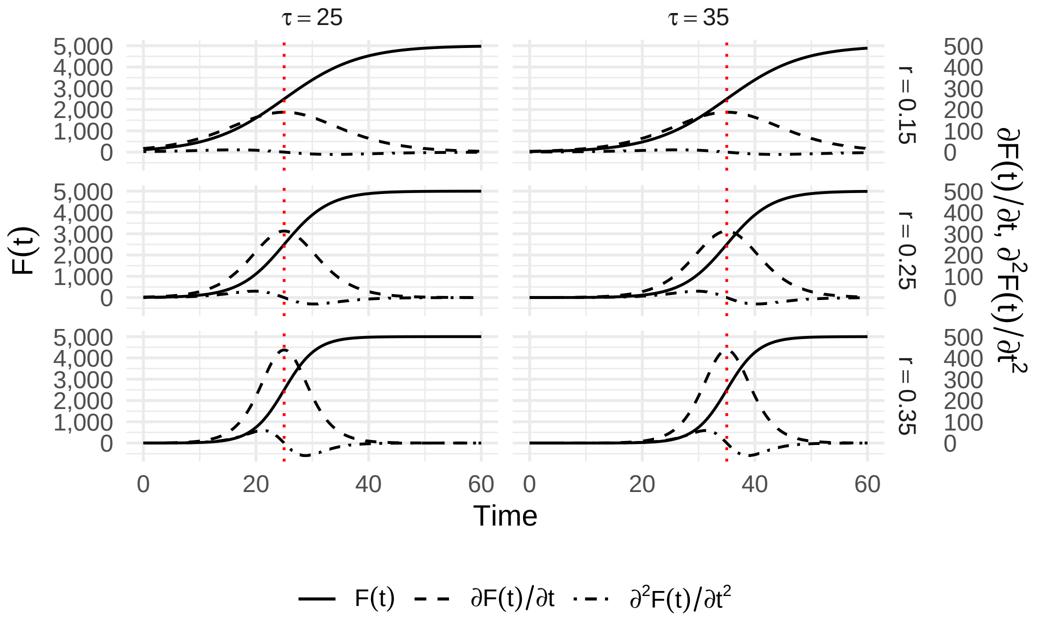


Figure 6.1: Illustration of the logistic model for different values for τ and r, with $k = 5,000$

6.1.2.1 Key moments

Let us focus on the key moment of the epidemics. Let us illustrate them using one of the scenarios.

```r
data_sim <- simu(1)
```

The starting point:

```r
df_starting <-
  data_sim %>%
  filter(sim >= 1) %>%
  slice(1)
```

The acceleration point

```r
df_acceleration <-
  data_sim %>%
  filter(simD2 == max(simD2)) %>%
  slice(1)
```

The peak:

```r
df_peak <-
  data_sim %>%
  filter(simD == max(simD)) %>%
  slice(1)
```

The deceleration point:

```r
df_deceleration <-
  data_sim %>%
  filter(simD2 == min(simD2)) %>%
  slice(1)
```

The return point:

```r
df_return <-
  data_sim %>%
  filter(sim > k - 1) %>%
  slice(1)
```

Let us reshape the data:

```r
df_plot_key_moments <-
  data_sim %>%
  gather(key, value, sim, simD, simD2)

df_plot_key_moments_points <-
  df_acceleration %>% mutate(label = "$t_A$") %>%
  bind_rows(df_peak %>% mutate(label = "$t_P$")) %>%
  bind_rows(df_deceleration %>% mutate(label = "$t_D$"))
```

And the plot:

```r
p_key_moments <-
  ggplot() +
  geom_hline(yintercept = 0, col = "grey") +
  geom_line(
    data = df_plot_key_moments,
    mapping = aes(x = t, y = value, linetype = key)
  ) +
  geom_segment(
    data = df_plot_key_moments_points,
    mapping = aes(x = t, xend = t, y = 0, yend = sim),
    linetype = "dotted", colour = "red"
  ) +
  geom_point(
    data = df_plot_key_moments_points,
    mapping = aes(x = t, y = sim), size = 2
  ) +
  geom_point(
    data = df_plot_key_moments_points,
    mapping = aes(x = t, y = sim),
    size = 1, colour = "white"
    ) +
  geom_text(
    data = df_plot_key_moments_points,
    mapping = aes(x = t-2, y = sim + .05*k, label = label)
  ) +
  labs(x = "Time", y = latex2exp::TeX("$F(t)$")) +
  scale_linetype_manual(
    NULL,
    values = c("sim" = "solid", "simD" = "dashed", "simD2" = "dotdash"),
    labels = c(
      "sim" = latex2exp::TeX("$F(t)$"),
      "simD" = latex2exp::TeX("$\\partial F(t) / \\partial t$"),
      "simD2" = latex2exp::TeX("$\\partial^2 F(t) / \\partial t^2$")
    )
  ) +
  theme(axis.text = element_blank(), axis.ticks = element_blank())

p_key_moments
```

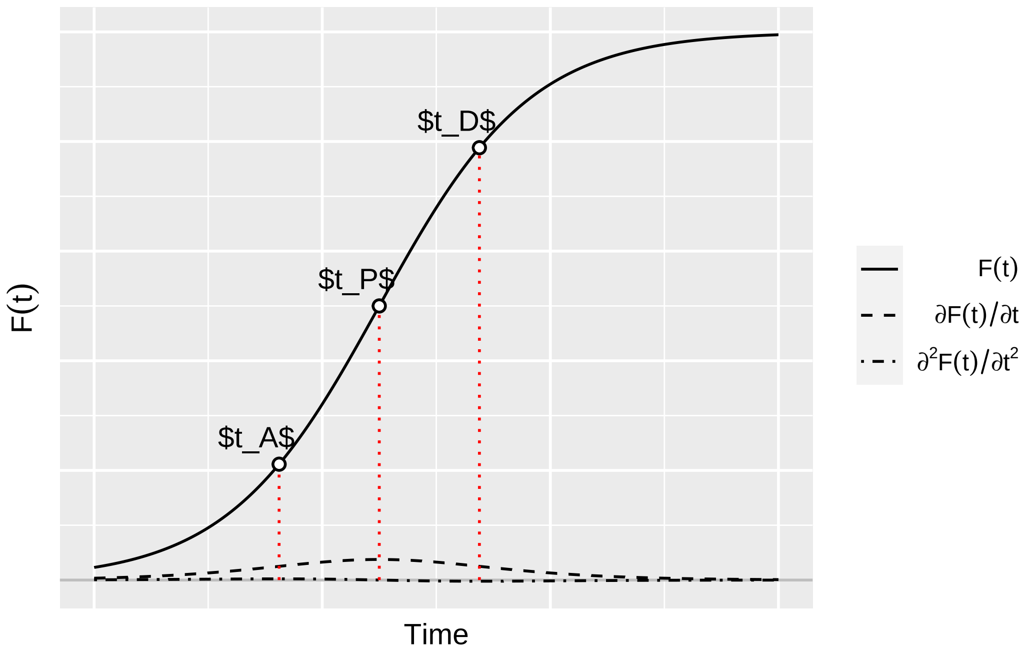


Figure 6.2: Key moments of the epidemics, using a logistim model.

6.1.3 Gompertz Model

The Gompertz model (Gompertz 1825) has three parameters, but corresponds to alternative restrictions with $\alpha = 1$ but this time $\delta = 0$. The solution to Equation 6.1 can be written as follows Tjørve and Tjørve (2017):

$$C(t) = K \exp\left[-\exp\left(-r(t-\tau)\right)\right]. \tag{6.4}$$

In Equation 6.4, the parameters are interpreted in the same way as those of the logistic model. The main difference between the logistic and Gompertz models is that the latter is not symmetric around the inflection point, which now appears earlier as $C(\tau) = K/e$. The first order derivative of this function provides an estimate of the number of cases at each point in time:

$$\frac{\partial C(t)}{\partial t} = Kr \exp\left[r(\tau - t) - \exp\left(r(\tau - t)\right)\right].$$

So that the relative speed of the epidemic is given by:

$$\frac{c(t)}{C(t)} = re^{-r(t-\tau)},$$

with of course a maximum speed at the peak.

In R, this translates to:

```r
#' Gompertz function with three parameters
#'
#' @param theta vector of named parameters
#' @param x time
gompertz_f <- function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]
  k*exp( -exp( -r * (x - tau) ) )
}
#' First order derivative of Gompertz wrt x
#'
#' @param theta vector of named parameters
#' @param x time
gompertz_f_first_d <- function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]

  k * (x - tau) * exp(r * (tau - x) - exp(r * (tau - x)))
}

#' Second order derivative of Gompertz
#'
#' @param theta vector of named parameters
#' @param x time
gompertz_f_second_d <- function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]
  -k * r^2 * exp(-r * (x - tau)) *
    exp(-exp(-r * (x - tau))) +
    k * r^2 * (exp(-r * (x - tau)))^2 * exp(-exp(-r * (x - tau)))
}
```

Let us make some graph to give an idea of how the dynamic changes with the parameters r and tau.

```r
k <- 5000
situations <-
  expand_grid(tau = c(25,35), r = c(.15, .25, .35)) |>
  mutate(k = k)

#' Simulation of the logistic model for some parameters
#'
#' @param i row number of situations
simu_gomp <- function(i) {
  current_params <- situations |> slice(i)
  current_params <- c(as_vector(current_params)) |> as.list()

  n <- 60
  step <- .01
  sim <-  gompertz_f(theta = current_params, x = seq(1,n, by = step))
  simD <-  gompertz_f_first_d(theta = current_params, x = seq(1,n, by = step))
  simD2 <-  gompertz_f_second_d(theta = current_params, x = seq(1,n, by = step))

  tibble(
    t = seq(1, n, by = step),
    k = current_params[["k"]],
    tau = current_params[["tau"]],
    r = current_params[["r"]],
    sim = sim) |>
    mutate(simD = simD,
           simD2 = simD2)

}
```

The values of $C(t)$, its first and second derivatives with respect to time can be computed for the different scenarios:

```r
simu_gomp_res <- map_df(1:nrow(situations), simu_gomp)
```

The threshold for each scenario:

```r
threshold_times <-
  simu_gomp_res |>
  group_by(k, tau, r) |>
  summarise(threshold_time = unique(tau)) |>
  ungroup() |>
  mutate(
    r = str_c("$\\r = ", r, "$"),
    tau = str_c("$\\tau = ", tau, "$")
  )
```

```
`summarise()` has grouped output by 'k', 'tau'. You can override using the
`.groups` argument.
```

And we can plot the results:

```r
p_simu_gomp <-
  ggplot(
  data = simu_gomp_res |>
    mutate(
      r = str_c("$\\r = ", r, "$"),
      tau = str_c("$\\tau = ", tau, "$")
    ),
  mapping = aes(x = t)) +
  geom_line(mapping = aes(y = sim, linetype = "sim")) +
  geom_line(mapping = aes(y = simD, linetype = "simD")) +
  geom_line(mapping = aes(y = simD2, linetype = "simD2")) +
  facet_grid(
    r~tau,
    labeller = as_labeller(
      latex2exp::TeX,
      default = label_parsed
    )
  ) +
  geom_vline(
    data = threshold_times,
    mapping = aes(xintercept = threshold_time),
    colour = "red", linetype = "dotted") +
  labs(x = "Time", y = latex2exp::TeX("$F(t)$")) +
  scale_linetype_manual(
    NULL,
    values = c("sim" = "solid", "simD" = "dashed", "simD2" = "dotdash"),
    labels = c("sim" = latex2exp::TeX("$F(t)$"),
               "simD" = latex2exp::TeX("$\\partial F(t) / \\partial t$"),
               "simD2" = latex2exp::TeX("$\\partial^2 F(t) / \\partial t^2$")
    )
  ) +
  theme(axis.ticks.y = element_blank())

p_simu_gomp +
  theme_minimal() +
  theme(legend.position = "bottom")
```

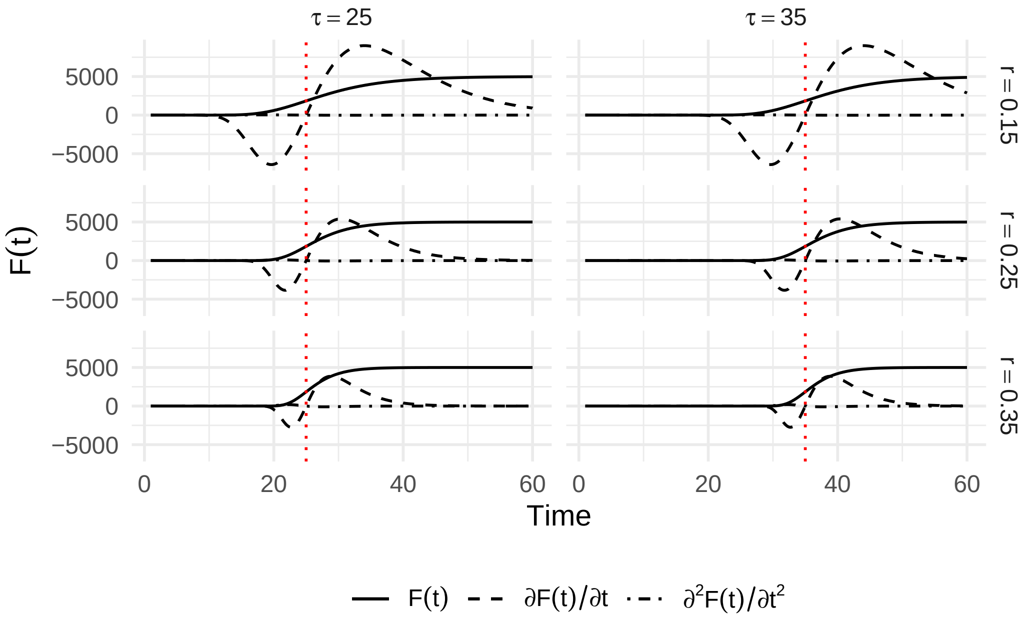


Figure 6.3: Illustration of Gompertz model for different values for τ and r, with $k = 5,000$

6.1.4 Richards Model

The question "what happens with $0 < \delta < 1$" finds an answer with Richards (1959) and the Richards model which is widely used in nonlinear regression analysis. The solution to Equation 6.1 provided in Lee, Lei, and Mallick (2020) corresponds to:

$$C(t) = K \left[1 + \delta \exp\left(-r(t-\tau)\right)\right]^{-1/\delta},$$

where parameters K and τ and are interpreted in the same way as those of the logistic and Gompertz models, while the growth rate is now r/δ. The parameter $\delta > 0$ is used to control the value of the curve at the inflection point $C(\tau) = K/(1+\delta)^{1/\delta}$, and thus the asymmetry around $t = \tau$. This model considers one more parameter than the Gompertz model to model this asymmetry. The first order derivative of this function will provide an estimate of the number of cases at each point in time:

$$\frac{\partial C(t)}{\partial t} = \frac{Kr\left[\delta \exp\left(r(\tau - t)\right) + 1\right]^{-1/\delta}}{\delta + \exp\left(r(t-\tau)\right)},$$

leading to a relative speed of:

$$\frac{c(t)}{C(t)} = r\frac{e^{-r(t-\tau)}}{1+\delta e^{-r(t-\tau)}}.$$

This model is also a generalization of the two previous ones as can be seen for instance when comparing the different relative speeds.

In R, this translates to the following functions:

```r
#' Richards function with four parameters
#'
#' @param theta vector of named parameters
#' @param x time
richards_f <- function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]
  delta <- theta[["delta"]]

  k / (1 + delta * exp(-r * delta * (x - tau)))^(1 / delta)
}


#' First order derivative of Richards function wrt time (x)
#'
#' @param theta vector of named parameters
#' @param x time
richards_f_first_d <- function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]
  delta <- theta[["delta"]]

  delta * k * r * exp(delta * (-r) * (x - tau)) *
    (delta * exp(delta * (-r) * (x - tau)) + 1)^(-1 / delta - 1)
}
```

```r
#' Second order derivative of Richards function wrt time (x)
#'
#' @param theta vector of named parameters
#' @param x time
richards_f_second_d <- function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]
  delta <- theta[["delta"]]

  k * ((-1 / delta - 1) *
         delta^3 * r^2 * (-exp(-2 * delta * r * (x - tau))) *
         (delta * exp(delta * (-r) * (x - tau)) + 1)^(-1 / delta - 2) -
         delta^2 * r^2 * exp(delta * (-r) * (x - tau)) *
         (delta * exp(delta * (-r) * (x - tau)) + 1)^(-1 / delta - 1))
}
```

Again, let us provide different scenarios to plot the curves for $C(t)$, its first and second derivatives with respect to time. We will make the parameters δ and r vary. We set k to $5,000$ and τ to 25.

```r
situations <-
  expand_grid(delta = c(.5, 1.5), r = c(.15, .25, .35)) |>
  mutate(k = 5000, tau = 25)
```

We create a function to get the values of $C(t)$ and its derivatives depending on the scenario.

```r
#' Simulation of Richards model for some parameters
#'
#' @param i row number of situations
simu_richards <- function(i) {
  current_params <- situations |> slice(i)
  current_params <- c(as_vector(current_params)) |> as.list()

  n <- 60
  step <- .01
  sim <-  richards_f(theta = current_params, x = seq(1,n, by = step))
  simD <-  richards_f_first_d(theta = current_params, x = seq(1,n, by = step))
  simD2 <-  richards_f_second_d(theta = current_params, x = seq(1,n, by = step))
```

```
    tibble(
      t = seq(1, n, by = step),
      k = current_params[["k"]],
      tau = current_params[["tau"]],
      r = current_params[["r"]],
      delta = current_params[["delta"]],
      sim = sim
      ) |>
      mutate(simD = simD,
             simD2 = simD2)
  }
```

This function is then applied to the different scenarios:

```
simu_richards_res <- map_df(1:nrow(situations), simu_richards)
```

The thresholds:

```
threshold_times <-
  simu_richards_res |>
  group_by(k, tau, r, delta) |>
  summarise(threshold_time = unique(tau)) |>
  ungroup() |>
  mutate(
    r = str_c("$r = ", r, "$"),
    delta = str_c("$\\delta = ", delta, "$")
  )
```

```
`summarise()` has grouped output by 'k', 'tau', 'r'. You can override using the
`.groups` argument.
```

```
p_simu_richards <-
  ggplot(
    data = simu_richards_res |>
      mutate(
        r = str_c("$r = ", r, "$"),
        delta = str_c("$\\delta = ", delta, "$")
      ),
    mapping = aes(x = t)
  ) +
  geom_line(mapping = aes(y = sim, linetype = "sim")) +
  geom_line(mapping = aes(y = simD * 10, linetype = "simD")) +
  geom_line(mapping = aes(y = simD2 * 10, linetype = "simD2")) +
  facet_grid(
    r ~ delta,
    labeller = as_labeller(
      latex2exp::TeX,
      default = label_parsed
    )
  ) +
  geom_vline(
    data = threshold_times,
    mapping = aes(xintercept = threshold_time),
    colour = "red", linetype = "dotted") +
  labs(x = "Time", y = latex2exp::TeX("$F(t)$")) +
  scale_y_continuous(
    labels = comma,
    breaks = seq(0, 5000, by = 1000),
    sec.axis = sec_axis(
      ~./10,
      name = latex2exp::TeX(
        "$\\partial F(t)/\\partial t$, $\\partial^2 F(t)/\\partial t^2$"
      ),
      labels = comma)
  ) +
  scale_linetype_manual(
    NULL,
    values = c("sim" = "solid", "simD" = "dashed", "simD2" = "dotdash"),
    labels = c("sim" = latex2exp::TeX("$F(t)$"),
               "simD" = latex2exp::TeX("$\\partial F(t) / \\partial t$"),
               "simD2" = latex2exp::TeX("$\\partial^2 F(t) / \\partial t^2$"))
  ) +
  theme(axis.ticks.y = element_blank())

p_simu_richards +
  theme_minimal() +
  theme(legend.position = "bottom")
```

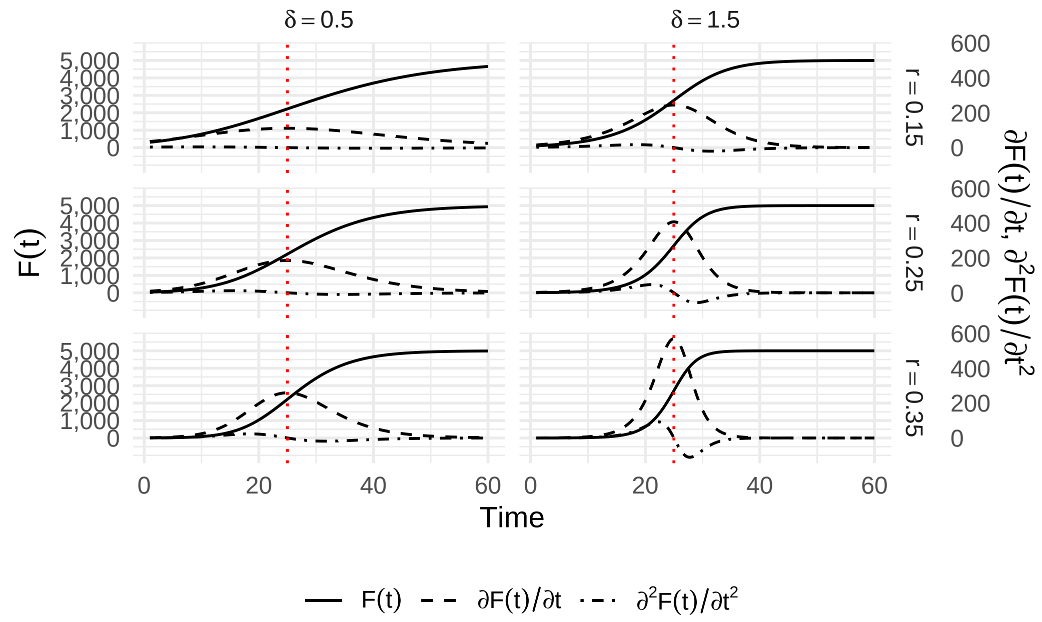


Figure 6.4: Illustration of Richards' model for different values for δ and r, with $k = 5,000$ and $\tau = 25$

6.2 Double sigmoid functions to account for a second wave

One possibility to account for two distinct phases is to use a double sigmoid. This is done by adding or multiplying two sigmoid functions, as follows:

$$C(t) = C_1^{(m_1)}(t) + C_2^{(m_2)}(t),$$

where $C_1^{(m_1)}(t)$ is the first sigmoid function and $C_2^{(m_2)}(t)$ is the second one. The types m_1 and m_2 of sigmoid can be, for example, a logistic curve (see, *e.g.*, Lipovetsky (2010) or Bock et al. (1973)), a Gompertz curve Thissen et al. (1976), or a Richards curve Oswald et al. (2012).

We limit ourselves here to the presentation of a double-Gompertz curve:

$$\begin{aligned} C(t) = &K_1 \exp\left[-\exp\left(-r_1(t-\tau_1)\right)\right] \\ &+ (K_2 - K_1)\exp\left[-\exp\left(-r_2(t-\tau_2)\right)\right], \end{aligned}$$

where K_1 and K_2 are the intermediate and final plateau of saturation, respectively. The parameters τ_1 and τ_2 define the inflection points of each phase when the intermediate plateau is long enough while the growth of the process is determined by r_1 and r_2, for the first and second sigmoid. However there is no analytical formula to determine the value and position of the two peaks. We have to evaluate numerically the extremum of the second order derivative of $C(t)$. The first order derivative of $C(t)$ provides an estimate of the number of cases at each point of time:

$$\begin{aligned}\frac{\partial C(t)}{\partial t} =& K_1 r_1 \exp(-r_1(t-\tau_1)) \exp(-\exp(-r1(t-\tau 1))) \\ & + (K_2 - K_1) r_2 \exp(-r_2(t-\tau_2)) \exp(-\exp(-r2(t-\tau 2))).\end{aligned}$$

The formula giving the relative speed $c(t)/C(t)$ does not lead to any simplification and will not be detailed.

6.2.1 Double Logistic

The double logistic function, its first and second derivatives with respect to time can be coded as follows:

```
# Double logistic
double_logistic_f <-  function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]

  (K1/(1 + exp(-r1 * (x - tau1)))) + ((K2 - K1)/(1 + exp(-r2 * (x - tau2))))
}
# Double logistic in two parts
double_logistic_f_2 <- function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]

  f_1 <- K1 / (1 + exp(-r1 * (x - tau1)))
  f_2 <- (K2 - K1)/(1 + exp(-r2 * (x - tau2)))
  tibble(x = x, f_1 = f_1, f_2 = f_2)
}

# First derivative
double_logistic_f_first_d <-  function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]

  (r1 * K1 * exp(-r1 * (x - tau1))) / ((exp(-r1 * (x-tau1)) + 1)^2) +
    (r2 * (K2-K1) * exp(-r2 * (x - tau2))) / ((exp(-r2 * (x-tau2)) + 1)^2)
}
# Second derivative
double_logistic_f_second_d <-  function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]

  K1 *  ((2 * r1^2 * exp(-2 * r1 * (x - tau1)))/(exp(-r1 * (x - tau1)) + 1)^3 -
           ( r1^2 * exp(- r1 * (x - tau1)))/(exp(-r1 * (x - tau1)) + 1)^2) +

    (K2-K1) *
    ((2 * r2^2 * exp(-2 * r2 * (x - tau2)))/(exp(-r2 * (x - tau2)) + 1)^3 -
       ( r2^2 * exp(- r2 * (x - tau2)))/(exp(-r2 * (x - tau2)) + 1)^2)

}
```

And now, let us make some illustrations by varying r_1 and r_2.

```
situations <-
  expand_grid(r1 = c(.2, .5), r2 = c(.2, .5)) |>
  mutate(
    K1   = 2500,
    K2   = 5000,
    tau1 = 20,
    tau2 = 40
  )
```

The function that simulates $C(t)$ and its first and second derivatives with respect to time for a single scenario:

```
#' Simulation of the logistic model for some parameters
#'
#' @param i row number of situations
simu <- function(i) {
  current_params <- situations |> slice(i)
  current_params <- c(as_vector(current_params)) |> as.list()

  n <- 60
  step <- .01
  sim <-  double_logistic_f(theta = current_params, x = seq(1,n, by = step))
  simD <-  double_logistic_f_first_d(
    theta = current_params, x = seq(1,n, by = step))
  simD2 <-  double_logistic_f_second_d(
    theta = current_params, x = seq(1,n, by = step))

  tibble(
    t = seq(1, n, by = step),
    K1 = current_params[["K1"]],
    r1 = current_params[["r1"]],
    tau1 = current_params[["tau1"]],
    K2 = current_params[["K2"]],
    r2 = current_params[["r2"]],
    tau2 = current_params[["tau2"]],
    sim = sim) |>
    mutate(simD = simD,
           simD2 = simD2)

}
```

The simulated values for all the scenarios:

```
simu_res <- map_df(1:nrow(situations), simu)
```

The thresholds:

```
threshold_times <-
  simu_res |>
  group_by(K1, tau1, r1, K2, tau2, r2) |>
  summarise(
    threshold_time_1 = unique(tau1),
    threshold_time_2 = unique(tau2)
  ) |>
```

```
    ungroup() |>
    mutate(
      r1 = str_c("$r_1 = ", r1, "$"),
      r2 = str_c("$r_2 = ", r2, "$")
    )
```

```
`summarise()` has grouped output by 'K1', 'tau1', 'r1', 'K2', 'tau2'. You can
override using the `.groups` argument.
```

```
p_simu_double_logis <-
  ggplot(
    data = simu_res |>
      mutate(
        r1 = str_c("$r_1 = ", r1, "$"),
        r2 = str_c("$r_2 = ", r2, "$")
      ),
    mapping = aes(x = t)
  ) +
  geom_line(aes(y = sim, linetype = "sim")) +
  geom_line(aes(y = simD * 10, linetype = "simD")) +
  geom_line(aes(y = simD2 * 10, linetype = "simD2")) +
  geom_vline(
    data = threshold_times,
    mapping = aes(xintercept = threshold_time_1),
    colour = "red", linetype = "dotted") +
  geom_vline(
    data = threshold_times,
    mapping = aes(xintercept = threshold_time_2),
    colour = "red", linetype = "dotted") +
  facet_grid(
    r2 ~ r1,
    labeller = as_labeller(
      latex2exp::TeX,
      default = label_parsed
    )
  ) +
  labs(x = "Time", y = "$F(t)$") +
  scale_y_continuous(
    labels = comma,
    sec.axis = sec_axis(
      ~./10,
```

```r
      name = latex2exp::TeX(
        "$\\partial F(t) / \\partial t$, $\\partial^2 F(t) / \\partial t^2$"
      ),
      labels = comma)
  ) +
  scale_linetype_manual(
    NULL,
    values = c("sim" = "solid", "simD" = "dashed", "simD2" = "dotdash"),
    labels = c("sim" = latex2exp::TeX("$F(t)$"),
               "simD" = latex2exp::TeX("$\\partial F(t) / \\partial t$"),
               "simD2" = latex2exp::TeX("$\\partial^2 F(t) / \\partial t^2$"))
               ) +
  theme(axis.ticks.y = element_blank())

p_simu_double_logis +
  theme_minimal() +
  theme(legend.position = "bottom")
```

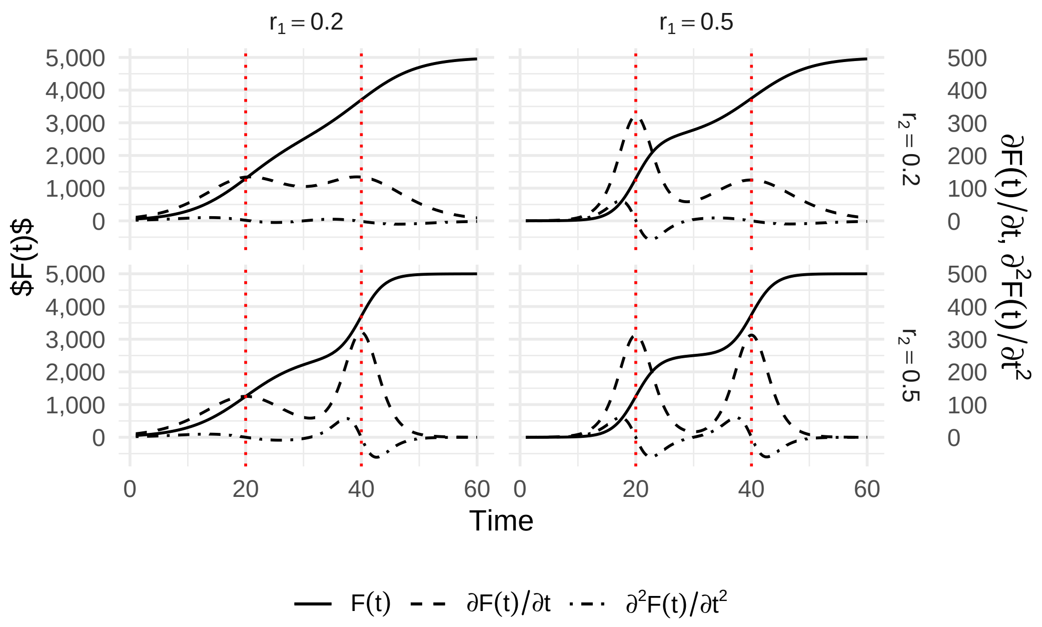


Figure 6.5: Illustration of a Double Logistic model for different values for r_1 and r_2, with $K_1 = 2,500$, $K_2 = 5,000$, $\tau_1 = 20$, and $\tau_2 = 40$

6.2.2 Double Gompertz Model

The double Gompertz function, its first and second derivatives with respect to time can be coded as follows:

```r
# Double-Gompertz function
#' @param theta vector of named parameters
#' @param x observation / training example
double_gompertz_f <- function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]

  f_1 <- K1 * exp(-exp(-r1 * (x - tau1)))
  f_2 <- (K2-K1) * exp(-exp(-r2 * (x - tau2)))

  f_1 + f_2
}

#' Double-Gompertz function in two parts
#'
double_gompertz_f_2 <- function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]

  f_1 <- K1 * exp(-exp(-r1 * (x - tau1)))
  f_2 <- (K2-K1) * exp(-exp(-r2 * (x-tau2)))
  tibble(x = x, f_1 = f_1, f_2 = f_2)
}

#' First derivative
#'
double_gompertz_f_first_d <- function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]

  f_1_d <- r1 * K1 * exp(-r1 * (x - tau1) - exp(- r1 * (x - tau1)))
  f_2_d <- r2 * (K2-K1) * exp(-r2 * (x - tau2) - exp(- r2 * (x - tau2)))

  f_1_d + f_2_d
}

#' Second derivative
double_gompertz_f_second_d <- function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]

  -K1 * r1^2 * exp(-r1*(x - tau1)) * exp(-exp(-r1*(x - tau1))) +
    K1 * r1^2 * (exp( -r1*(x - tau1)))^2 * exp(-exp(-r1*(x - tau1))) -
    (K2 - K1) * r2^2 * exp( -r2 * (x - tau2)) * exp( -exp(-r2*(x - tau2))) +
    (K2 - K1) * r2^2 * (exp( -r2 * (x - tau2)))^2 * exp(-exp(-r2*(x-tau2)))
}
```

Let us make some illustrations by varying r_1 and r_2.

```
situations <-
  expand_grid(r1 = c(.2, .5), r2 = c(.2, .5)) |>
  mutate(
    K1     = 2500,
    K2     = 5000,
    tau1   = 20,
    tau2   = 40
  )
```

The function that simulates $C(t)$ and its first and second derivatives with respect to time for a single scenario:

```r
#' Simulation of the logistic model for some parameters
#'
#' @param i row number of situations
simu <- function(i) {
  current_params <- situations |> slice(i)
  current_params <- c(as_vector(current_params)) |> as.list()

  n <- 60
  step <- .01
  sim <-  double_gompertz_f(
    theta = current_params, x = seq(1,n, by = step))
  simD <-  double_gompertz_f_first_d(
    theta = current_params, x = seq(1,n, by = step))
  simD2 <-  double_gompertz_f_second_d(
    theta = current_params, x = seq(1,n, by = step))

  tibble(
    t = seq(1, n, by = step),
    K1 = current_params[["K1"]],
    r1 = current_params[["r1"]],
    tau1 = current_params[["tau1"]],
    K2 = current_params[["K2"]],
    r2 = current_params["r2"],
    tau2 = current_params[["tau2"]],
    sim = sim) |>
    mutate(simD = simD,
           simD2 = simD2)

}
```

Let us apply this function to all scenarios:

```r
simu_res <- map_df(1:nrow(situations), simu)
```

The thresholds:

```r
threshold_times <-
  simu_res |>
  group_by(K1, tau1, r1, K2, tau2, r2) |>
  summarise(threshold_time_1 = unique(tau1),
            threshold_time_2 = unique(tau2)) |>
  ungroup() |>
```

```
    mutate(
      r1 = str_c("$\\r_1 = ", r1, "$"),
      r2 = str_c("$\\r_2 = ", r2, "$")
    )
```

```
`summarise()` has grouped output by 'K1', 'tau1', 'r1', 'K2', 'tau2'. You can
override using the `.groups` argument.
```

And the the plots:

```
p_simu_double_gompertz <-
  ggplot(
    data =  simu_res |>
      mutate(r1 = str_c("$\\r_1 = ", r1, "$"),
             r2 = str_c("$\\r_2 = ", r2, "$")
      ),
    mapping = aes(x = t)) +
  geom_line(aes(y = sim, linetype = "sim")) +
  geom_line(aes(y = simD*10, linetype = "simD")) +
  geom_line(aes(y = simD2*10, linetype = "simD2")) +
  geom_vline(
    data = threshold_times,
    mapping = aes(xintercept = threshold_time_1),
    colour = "red", linetype = "dotted") +
  geom_vline(
    data = threshold_times,
    mapping = aes(xintercept = threshold_time_2),
    colour = "red", linetype = "dotted") +
  facet_grid(
    r2 ~ r1,
    labeller = as_labeller(
      latex2exp::TeX,
      default = label_parsed
    )
  ) +
  labs(x = "Time", y = latex2exp::TeX("$F(t)$")) +
  scale_y_continuous(
    labels = comma,
    sec.axis = sec_axis(
      ~./10,
      name = latex2exp::TeX(
        "$\\partial F(t) / \\partial t$, $\\partial^2 F(t) / \\partial t^2$"
      ),
      labels = comma)) +
  scale_linetype_manual(
    NULL,
    values = c("sim" = "solid", "simD" = "dashed", "simD2" = "dotdash"),
    labels = c("sim" = latex2exp::TeX("$F(t)$"),
               "simD" = latex2exp::TeX("$\\partial F(t) / \\partial t$"),
               "simD2" = latex2exp::TeX("$\\partial^2 F(t) / \\partial t^2$"))
    ) +
  theme(axis.ticks.y = element_blank())

p_simu_double_gompertz +
  theme_minimal() +
  theme(legend.position = "bottom")
```

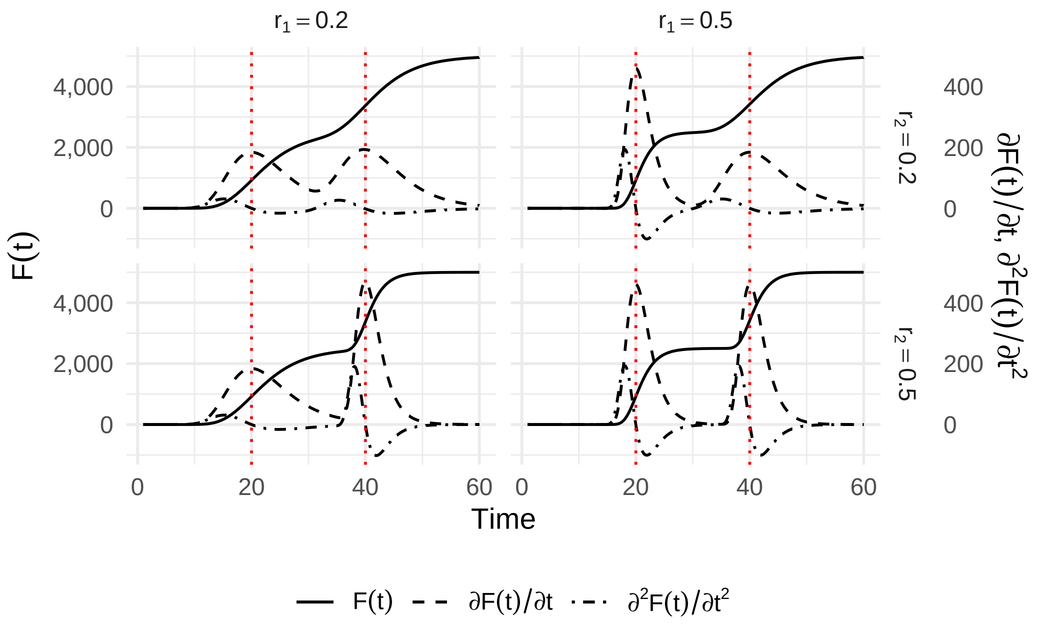


Figure 6.6: Illustration of a Double Gompertz model for different values for r_1 and r_2, with $K_1 = 2,500$, $K_2 = 5,000$, $\tau_1 = 20$, and $\tau_2 = 40$

6.2.3 Double Richards Model

The double Richards function, its first and second derivatives with respect to time can be coded as follows:

```r
#' Double Richards
#'
double_richards_f <- function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  delta1 <- theta[["delta1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]
  delta2 <- theta[["delta2"]]

  K1 * (1 + delta1 * exp(-r1 * (x - tau1)))^(-1 / delta1) +
    (K2 - K1) * (1 + delta2 * exp(-r2 * (x - tau2)))^(-1 / delta2)
}

#' Double Richards in two parts
#'
double_richards_f_2 <- function(theta, x) {

  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  delta1 <- theta[["delta1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]
  delta2 <- theta[["delta2"]]

  f_1 <- K1 * (1 + delta1 * exp(-r1 * (x - tau1)))^(-1 / delta1)
  f_2 <- (K2 - K1) * (1 + delta2 * exp(-r2 * (x - tau2)))^(-1 / delta2)

  tibble(x = x, f_1 = f_1, f_2 = f_2)
}

#' First derivative
#'
double_richards_f_first_d <- function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  delta1 <- theta[["delta1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]
  delta2 <- theta[["delta2"]]

  r1 * K1 * exp(-r1 * (x - tau1)) *
    (delta1 * exp(-r1 * (x - tau1)) + 1)^(-1/delta1 - 1) +
    r2 * (K2-K1) * exp(-r2 * (x - tau2)) *
    (delta2 * exp(-r2 * (x - tau2)) + 1)^(-1/delta2 - 1)
}

#' Second derivative
#'
double_richards_f_second_d <- function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  delta1 <- theta[["delta1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]
  delta2 <- theta[["delta2"]]

  K1 * (r1^2 * (-1/delta1 - 1) * delta1 * (-exp(-2 * r1 * (x - tau1))) *
          (delta1 * exp(-r1 * (x - tau1)) + 1)^(-1/delta1 - 2) - r1^2 *
          exp(-r1 * (x - tau1)) *
          (delta1 * exp(-r1 * (x - tau1)) + 1)^(-1/delta1 - 1)) +

    (K2 - K1) *
    (r2^2 * (-1 / delta2 - 1) *
       delta2 * (-exp(-2 * r2 * (x - tau2))) *
       (delta2 * exp(-r2 * (x - tau2)) + 1)^(-1/delta2 - 2) -
       r2^2 *
       exp(-r2 * (x - tau2)) *
       (delta2 * exp(-r2 * (x - tau2)) + 1)^(-1/delta2 - 1))
}
```

Let us make some illustrations by varying r_1 and r_2.

```
situations <-
  expand_grid(r1 = c(.2, .5), r2 = c(.2, .5)) |>
  mutate(
    K1     = 2500,
    K2     = 5000,
    tau1   = 20,
    tau2   = 40,
    delta1 = .5,
    delta2 = .5
  )
```

The function that simulates $C(t)$ and its first and second derivatives with respect to time for a single scenario:

```
#' Simulation of the logistic model for some parameters
#'
#' @param i row number of situations
simu <- function(i) {
  current_params <- situations |> slice(i)
  current_params <- c(as_vector(current_params)) |> as.list()

  n <- 60
  step <- .01
  sim <-  double_richards_f(
    theta = current_params, x = seq(1,n, by = step))
  simD <-  double_richards_f_first_d(
    theta = current_params, x = seq(1,n, by = step))
  simD2 <-  double_richards_f_second_d(
    theta = current_params, x = seq(1,n, by = step))

  tibble(
    t = seq(1, n, by = step),
    K1 = current_params[["K1"]],
    r1 = current_params[["r1"]],
    tau1 = current_params[["tau1"]],
    delta1 = current_params[["delta1"]],
    K2 = current_params[["K2"]],
    r2 = current_params["r2"],
    tau2 = current_params[["tau2"]],
    delta2 = current_params[["delta2"]],
    sim = sim) |>
    mutate(simD = simD,
              simD2 = simD2)
}
```

Let us apply this function to all scenarios:

```
simu_res <- map_df(1:nrow(situations), simu)
```

The thresholds:

```
threshold_times <-
  simu_res |>
  group_by(K1, tau1, r1, K2, tau2, r2, delta1, delta2) |>
  summarise(threshold_time_1 = unique(tau1),
            threshold_time_2 = unique(tau2)) |>
  ungroup() |>
  mutate(
    r1 = str_c("$\\r_1 = ", r1, "$"),
    r2 = str_c("$\\r_2 = ", r2, "$")
  )
```

```
`summarise()` has grouped output by 'K1', 'tau1', 'r1', 'K2', 'tau2', 'r2',
'delta1'. You can override using the `.groups` argument.
```

And the the plots:

```
p_simu_double_richards <-
  ggplot(
    data =  simu_res |>
      mutate(r1 = str_c("$\\r_1 = ", r1, "$"),
             r2 = str_c("$\\r_2 = ", r2, "$")
      ),
    mapping = aes(x = t)) +
  geom_line(aes(y = sim, linetype = "sim")) +
  geom_line(aes(y = simD*10, linetype = "simD")) +
  geom_line(aes(y = simD2*10, linetype = "simD2")) +
  geom_vline(
    data = threshold_times,
    mapping = aes(xintercept = threshold_time_1),
    colour = "red", linetype = "dotted") +
  geom_vline(
    data = threshold_times,
    mapping = aes(xintercept = threshold_time_2),
    colour = "red", linetype = "dotted") +
  facet_grid(
    r2 ~ r1,
    labeller = as_labeller(
      latex2exp::TeX,
      default = label_parsed
    )
  ) +
  labs(x = "Time", y = latex2exp::TeX("$F(t)$")) +
  scale_y_continuous(
    labels = comma,
    sec.axis = sec_axis(
      ~./10,
      name = latex2exp::TeX(
        "$\\partial F(t) / \\partial t$, $\\partial^2 F(t) / \\partial t^2$"
      ),
      labels = comma)) +
  scale_linetype_manual(
    NULL,
    values = c("sim" = "solid", "simD" = "dashed", "simD2" = "dotdash"),
    labels = c("sim" = latex2exp::TeX("$F(t)$"),
               "simD" = latex2exp::TeX("$\\partial F(t) / \\partial t$"),
               "simD2" = latex2exp::TeX("$\\partial^2 F(t) / \\partial t^2$"))
    ) +
  theme(axis.ticks.y = element_blank())

p_simu_double_richards +
  theme_minimal() +
  theme(legend.position = "bottom")
```

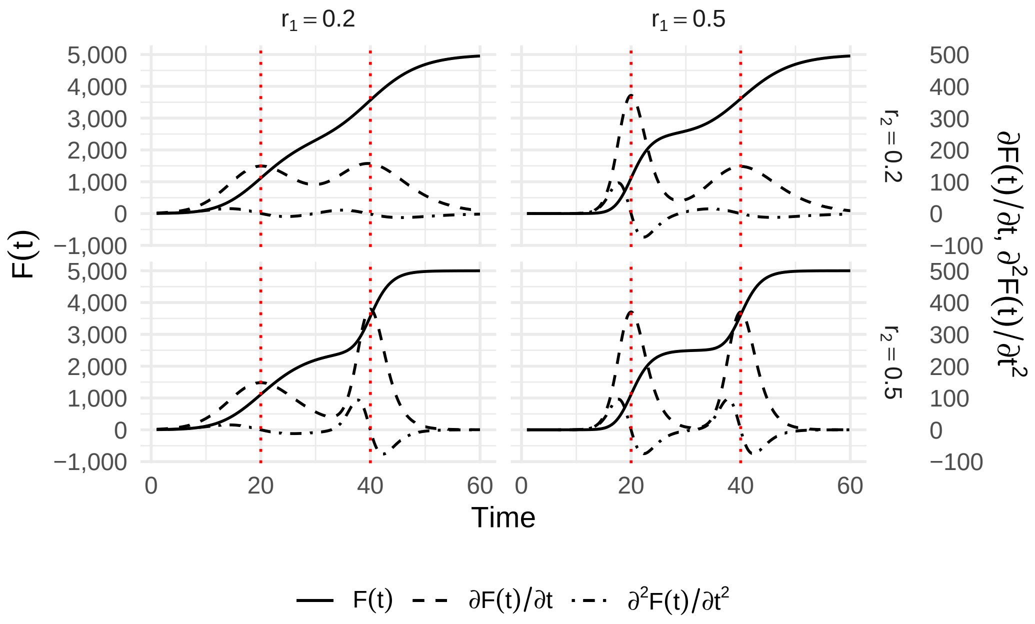


Figure 6.7: Illustration of a Double Richards model for different values for r_1 and r_2, with $K_1 = 2,500$, $K_2 = 5,000$, $\tau_1 = 20$, $\tau_2 = 40$“, $\delta_1 = \delta_2 = .5$

7 Phenomenological Models

This chapter shows R codes that can be used to fit the phenomenological models, presented in Chapter 6, on observed confirmed cases. It shows the codes to model a single wave, and those that can be used to model two waves.

```
# FOR UNIX USERS
Sys.setlocale("LC_ALL", "en_US.UTF-8")
```

```
[1] "en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8"
```

```
# FOR WINDOWS USERS
# Sys.setlocale("LC_ALL", "English_United States")
```

Some packages that will be used:

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.2     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```
library(scales)
```

```
Attaching package: 'scales'
```

```
The following object is masked from 'package:purrr':

    discard

The following object is masked from 'package:readr':

    col_factor
```

```r
library(minpack.lm)
library(mvtnorm)
```

Let us load the data (results obtained from Chapter 4).

```r
load("data/data_after_load.rda")
```

7.1 Simple phenomenological models

7.1.1 Logistic Model

In R, we define the logistic function as follows:

```r
#' Logistic function
#'
#' @param theta vector of named parameters
#' @param x time
logistic_f <-  function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]
  k / ( 1+exp( -r*( x - tau ) ) )
}

#' First derivative of the logistic function
#'
#' @param theta vector of named parameters
#' @param x time
logistic_f_first_d <- function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]

  (k * r * exp( -r * (x - tau) )) / (1+ exp( -r * (x - tau) ))^2
}

#' Second derivative of the logistic function
#'
#' @param theta vector of named parameters
#' @param x time
logistic_f_second_d <- function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]

  k*((2*r^2 * exp(-2*r*(x - tau)))/
       (exp(-r*(x - tau)) + 1)^3 -
       (r^2*exp(-r*(x - tau)))/(exp(-r*(x - tau)) + 1)^2)
}
```

7.1.2 Gompertz Model

In R, the Gompertz Model can be written as:

```r
#' Gompertz function with three parameters
#'
#' @param theta vector of named parameters
#' @param x time
gompertz_f <- function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]
  k*exp( -exp( -r * (x - tau) ) )
}
#' First order derivative of Gompertz wrt x
#'
#' @param theta vector of named parameters
#' @param x time
gompertz_f_first_d <- function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]

  k * (x - tau) * exp(r * (tau - x) - exp(r * (tau - x)))
}

#' Second order derivative of Gompertz
#'
#' @param theta vector of named parameters
#' @param x time
gompertz_f_second_d <- function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]

  -k * r^2 * exp(-r * (x - tau)) *
    exp(-exp(-r * (x - tau))) +
    k * r^2 * (exp(-r * (x - tau)))^2 * exp(-exp(-r * (x - tau)))
}
```

7.1.3 Richards Model

In R, Richards' Model can be coded as follows:

```
#' Richards function with four parameters
#'
#' @param theta vector of named parameters
#' @param x time
richards_f <- function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]
  delta <- theta[["delta"]]

  k / (1 + delta * exp(-r * delta * (x - tau)))^(1 / delta)
}


#' First order derivative of Richards function wrt time (x)
#'
#' @param theta vector of named parameters
```

```r
#' @param x time
richards_f_first_d <- function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]
  delta <- theta[["delta"]]

  delta * k * r * exp(delta * (-r) * (x - tau)) *
    (delta * exp(delta * (-r) * (x - tau)) + 1)^(-1 / delta - 1)
}


#' Second order derivative of Richards function wrt time (x)
#'
#' @param theta vector of named parameters
#' @param x time
richards_f_second_d <- function(theta, x) {
  k <- theta[["k"]]
  tau <- theta[["tau"]]
  r <- theta[["r"]]
  delta <- theta[["delta"]]

  k * ((-1 / delta - 1) *
         delta^3 * r^2 * (-exp(-2 * delta * r * (x - tau))) *
         (delta * exp(delta * (-r) * (x - tau)) + 1)^(-1 / delta - 2) -
         delta^2 * r^2 * exp(delta * (-r) * (x - tau)) *
         (delta * exp(delta * (-r) * (x - tau)) + 1)^(-1 / delta - 1))
}
```

7.1.4 Help functions

Some functions can be helpful to fit the models both for the number of cases and deaths. First of all, we can create a function that gives the dates for the different periods:

- `start_first_wave`: start of the first wave, defined as the first date when the cumulative number of cases is greater than 1
- `start_high_stringency`: date at which the stringency index reaches its maximum value during the first 100 days of the sample
- `start_reduce_restrict`: moment at which the restrictions of the first wave starts to lower
- `start_date_sample_second_wave`: 60 days after the relaxation of restrictions (60 days after after `start_reduce_restrict`)

- `length_high_stringency`: number of days between `start_high_stringency` and start_reduce_restrict‘.

```
#' Gives the dates of the different periods (first wave, start of containment, ...)
#'
#' @param country_name name of the country
#' @param type if `"deaths"` returns the number of deaths, otherwise the number of cases
get_dates <- function(country_name,
                      type = c("cases", "deaths")) {
  if (type == "deaths") {
    df_country <- deaths_df |> filter(country == !!country_name)
  } else {
    df_country <- confirmed_df |> filter(country == !!country_name)
  }

  # Start of the first wave
  start_first_wave <-
    df_country |>
    arrange(date) |>
    filter(value > 0) |>
    slice(1) |>
    magrittr::extract2("date")

  # Start of period with severity greater or equal than 70 index among the first 100 days
  start_high_stringency <-
    df_country |>
    slice(1:100) |>
    filter(stringency_index >= 70) |>
    slice(1) |>
    magrittr::extract2("date")

  # Max for Sweden
  if (country_name == "Sweden") {
    start_high_stringency <-
      df_country |>
      slice(1:100) |>
      arrange(desc(stringency_index), date) |>
      slice(1) |>
      magrittr::extract2("date")
  }

  # Max stringency first 100 days
```

```
    start_max_stringency <-
      df_country |>
      slice(1:100) |>
      arrange(desc(stringency_index), date) |>
      slice(1) |>
      magrittr::extract2("date")

    # Moment at which the restrictions of the first wave starts to lower
    start_reduce_restrict <-
      df_country |>
      arrange(date) |>
      filter(date >= start_max_stringency) |>
      mutate(tmp = dplyr::lag(stringency_index)) |>
      mutate(same_strin = stringency_index == tmp) |>
      mutate(same_strin = ifelse(row_number() == 1, TRUE, same_strin)) |>
      filter(same_strin == FALSE) |>
      slice(1) |>
      magrittr::extract2("date")

    start_date_sample_second_wave <- start_reduce_restrict +
      lubridate::ddays(60)


    # Length of high stringency period
    length_high_stringency <- lubridate::interval(
      start_high_stringency, start_reduce_restrict) /
      lubridate::ddays(1)


    tibble(
      country = country_name,
      start_first_wave = start_first_wave,
      start_high_stringency = start_high_stringency,
      start_reduce_restrict = start_reduce_restrict,
      start_date_sample_second_wave = start_date_sample_second_wave,
      length_high_stringency = length_high_stringency
    )
  }
```

We can then create a function to prepare the dataset fot a given country. This function allows to provide data for:

- the first wave sample: between the first case and 60 days after the day at which the sever-

  ity index reaches its max value (during the first 100 days) ; we also add some data for the out-of-sample prediction (of length set outside the function by `out_of_sample_horizon`)
- the second wave sample: data posterior to 60 days after the relaxation of restrictions
- all available data

The argument `type` of the function controls whether we get number of cases or deaths.

```r
#' Extracts the cases data for a country
#'
#' @description
#' If one only wants the first wave (first_wave=TRUE) the following start and
#' end date are used:
#' - start: date of the first case
#' - end: when the severity index is greater than 70
#'
#' @param country_name name of the country
#' @param sample if `"first"`, returns the sample for the first wave; if "second",
#' for the second wave, and `"all"` for all data up to the end
#' @param type if `"deaths"` returns the number of deaths, otherwise the number of cases
get_values_country <- function(country_name,
                               sample = c("first", "second", "all"),
                               type = c("cases", "deaths")) {
  if (type == "deaths") {
    df_country <- deaths_df |> filter(country == !!country_name)
  } else {
    df_country <- confirmed_df |> filter(country == !!country_name)
  }
  dates_country <- get_dates(country_name, type = type)

  # Maximum of the severity index
  max_severity <- max(df_country$stringency_index, na.rm=TRUE)
  dates_country$max_severity <- max_severity

  if (sample == "first") {
    df_country <-
      df_country |>
      filter(date >= dates_country$start_first_wave,
             # 60 days after end max stringency in the first interval of 100
             # days + `out_of_sample_horizon` more days for out-of-sample pred
             date <= dates_country$start_date_sample_second_wave +
               lubridate::ddays(out_of_sample_horizon)
             )
```

```r
} else if (sample == "second") {
  df_country <-
    df_country |>
    filter(date >= dates_country$start_date_sample_second_wave)

  # Let us remove the number of cases of the first date of this sample
  # to all observation (translation to 1)
  start_val_cases <- df_country$value[1]
  df_country <-
    df_country |>
    mutate(value = value - start_val_cases + 1)
} else {
  df_country <-
    df_country |>
    filter(date >= dates_country$start_first_wave)
}

# Moving Average for missing values (i.e., for Ireland)
if (any(is.na(df_country$value))) {
  replacement_values <- round(
    zoo::rollapply(
      df_country$value,
      width=3,
      FUN=function(x) mean(x, na.rm=TRUE),
      by=1,
      by.column=TRUE,
      partial=TRUE,
      fill=NA,
      align="center")
  )

  # Replace only missing values
  df_country <-
    df_country |>
    mutate(replacement_values = !!replacement_values) |>
    mutate(
      value = ifelse(
        is.na(value),
        yes = replacement_values,
        no = value)
    ) |>
```

```r
      select(-replacement_values)
  }

  df_country <-
    df_country |>
    mutate(t = row_number()-1) |>
    mutate(y = value)

  list(df_country = df_country, dates_country = dates_country)
}
```

We need to define a loss function that will be minimized to get the estimates of the models.

```r
#' Loss function
#'
#' @param theta vector of named parameters of the model
#' @param fun prediction function of the model
#' @param y target variable
#' @param t time component (feature)
loss_function <- function(theta,
                          fun,
                          y,
                          t) {
  (y - fun(theta = theta, x = t))
}
```

Once the model are estimated, we can compute some goodness of fit criteria. Let us create a function that computes the AIC, the BIC and the RMSE for a specific model. The function expects three arguments: the prediction function of the model (`f`), the values for the parameters of the model (in a named vector – `theta`), and the observations (`data`).

```r
#' Compute some goodness of fit criteria
#'
#' @param f prediction function of the model
#' @param data data that contains the two columns `y` and `t`
#' @param theta estimated coefficients for the model (used in `f`)
get_criteria <- function(f,
                         data,
                         theta) {
  n <- nrow(data)
  k <- length(theta)
  w <- rep(1, n)
  errors <- loss_function(theta = theta, fun = f, y = data$y, t = data$t)

  mse <- sum(errors^2) / n
  rmse <- sqrt(mse)

  # Log-likelihood
  ll <- 0.5 * (sum(log(w)) - n *
                 (log(2 * pi) + 1 - log(n) + log(sum(w * errors^2))))
  aic <- 2 * (k+1) - 2*ll
  bic <- -2 * ll + log(n) * (k+1)

  c(AIC = aic, BIC = bic, RMSE = rmse)

}
```

7.2 Predictions at the end of the first wave

Let us turn to the estimation of the number of cases and the estimation of the number of deaths for the first wave. Recall that the sample used is defined as follows: from the first date where the cumulative number of cases (deaths) is greater or equal to 1 to 60 days after the maximum value of the stringency index (start of containment).

Let us focus here first on the estimation of the Gompertz model, for the number of cased, in the UK. Then, we can wrap up the code and estimate all models to all countries.

7.2.1 Example for the UK

We would like to make out-of-sample predictions, to assess how the models perform with unseen data. To that end, let us define a horizon at which to look at (30 days):

```r
out_of_sample_horizon <- 30
```

Let us also define a limit of the number of fitted values to return:

```r
horizon_pred <- 250
```

We want to focus on the UK:

```r
country_name <- "United Kingdom"
pop_country <- population |>
  filter(country == !!country_name) |>
  magrittr::extract2("pop")
pop_country
```

```
[1] 6.7e+07
```

The data for the sample can be obtained using the `get_values_country()` function.

```r
data_country <-
  get_values_country(
    country_name = country_name,
    sample = "first",
    type = "cases"
  )
data_country
```

```
$df_country
# A tibble: 192 x 8
   country   country_code date       value stringency_index days_since_2020_01_22
   <chr>     <chr>        <date>     <int>            <dbl>                 <dbl>
 1 United ~ GBR          2020-01-31     2             8.33                     9
 2 United ~ GBR          2020-02-01     2             8.33                    10
 3 United ~ GBR          2020-02-02     2            11.1                     11
 4 United ~ GBR          2020-02-03     8            11.1                     12
 5 United ~ GBR          2020-02-04     8            11.1                     13
 6 United ~ GBR          2020-02-05     9            11.1                     14
 7 United ~ GBR          2020-02-06     9            11.1                     15
 8 United ~ GBR          2020-02-07     9            11.1                     16
 9 United ~ GBR          2020-02-08    13            11.1                     17
10 United ~ GBR          2020-02-09    14            11.1                     18
# i 182 more rows
# i 2 more variables: t <dbl>, y <int>

$dates_country
# A tibble: 1 x 7
  country        start_first_wave start_high_stringency start_reduce_restrict
  <chr>          <date>           <date>                <date>
1 United Kingdom 2020-01-31       2020-03-23            2020-05-11
# i 3 more variables: start_date_sample_second_wave <date>,
#   length_high_stringency <dbl>, max_severity <dbl>
```

We obtained both the sample to learn from and the dates of interest, in a list. Let us store those in single objects:

```
df_country <- data_country$df_country
dates_country <- data_country$dates_country
```

The model we wish to estimate is the Gompertz model:

```
model_name <- "Gompertz"
```

The different functions of the model are the following:

```
model_function <- gompertz_f
model_function_d <- gompertz_f_first_d
model_function_dd <- gompertz_f_second_d
```

The training sample:

```
# Training sample
df_country_training <-
  df_country |>
  slice(1:(nrow(df_country) - out_of_sample_horizon))
df_country_training
```

```
# A tibble: 162 x 8
   country   country_code date       value stringency_index days_since_2020_01_22
   <chr>     <chr>        <date>     <int>            <dbl>                 <dbl>
 1 United ~ GBR          2020-01-31     2             8.33                     9
 2 United ~ GBR          2020-02-01     2             8.33                    10
 3 United ~ GBR          2020-02-02     2            11.1                     11
 4 United ~ GBR          2020-02-03     8            11.1                     12
 5 United ~ GBR          2020-02-04     8            11.1                     13
 6 United ~ GBR          2020-02-05     9            11.1                     14
 7 United ~ GBR          2020-02-06     9            11.1                     15
 8 United ~ GBR          2020-02-07     9            11.1                     16
 9 United ~ GBR          2020-02-08    13            11.1                     17
10 United ~ GBR          2020-02-09    14            11.1                     18
# i 152 more rows
# i 2 more variables: t <dbl>, y <int>
```

Then, we can fit the model. But first, we need starting values for the optimization algorithm:

```r
# The starting values
start <- list(
  k   = pop_country,
  tau = 80,
  r   = .24
)
```

Then we can proceed with the estimation of the model:

```r
out <- nls.lm(
  par = start,
  fn = loss_function,
  y = df_country$y,
  t = df_country$t,
  fun = model_function,
  nls.lm.control(maxiter = 100),
  jac = NULL,
  lower = NULL,
  upper = NULL)
```

Here are the results:

```r
summary(out)
```

```
Parameters:
     Estimate Std. Error t value Pr(>|t|)
k   2.992e+05  6.512e+02   459.4   <2e-16 ***
tau 7.587e+01  1.410e-01   538.2   <2e-16 ***
r   4.296e-02  3.861e-04   111.3   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3738 on 189 degrees of freedom
Number of iterations to termination: 12
Reason for termination: Relative error in the sum of squares is at most `ftol'.
```

Let us store the estimated parameters of the model:

```
params <- tibble(
  model_type = model_name,
  country = country_name,
  coef_estimate_name = names(coef(out)),
  coef_estimate = coef(out)
)
params
```

```
# A tibble: 3 x 4
  model_type country        coef_estimate_name coef_estimate
  <chr>      <chr>          <chr>                      <dbl>
1 Gompertz   United Kingdom k                      299189.
2 Gompertz   United Kingdom tau                        75.9
3 Gompertz   United Kingdom r                           0.0430
```

The goodness of fit can be obtained using the previously defined function `get_criteria()`, and the results can be saved in a table:

```
# Goodness of fit
crit <- get_criteria(
  f = model_function,
  data = df_country_training,
  theta = params$coef_estimate
)
criteria <- tibble(
  model_type = model_name,
  country = country_name,
  bind_rows(crit)
)
criteria
```

```
# A tibble: 1 x 5
  model_type country          AIC   BIC  RMSE
  <chr>      <chr>          <dbl> <dbl> <dbl>
1 Gompertz   United Kingdom 3049. 3062. 2885.
```

Let us now turn to the out-of-sample predictions. We need to compare the predictions of the model 30 days after the end of the sample, with those that were observed (but not used in the estimation). The test sample becomes:

```r
df_country_test <-
  df_country |>
  slice((nrow(df_country) - out_of_sample_horizon + 1):(nrow(df_country)))
df_country_test
```

```
# A tibble: 30 x 8
   country country_code date        value stringency_index days_since_2020_01_22
   <chr>   <chr>        <date>      <int>            <dbl>                <dbl>
 1 United~ GBR          2020-07-11 288953             64.4                  171
 2 United~ GBR          2020-07-12 289603             64.4                  172
 3 United~ GBR          2020-07-13 290133             64.4                  173
 4 United~ GBR          2020-07-14 291373             64.4                  174
 5 United~ GBR          2020-07-15 291911             64.4                  175
 6 United~ GBR          2020-07-16 292552             64.4                  176
 7 United~ GBR          2020-07-17 293239             64.4                  177
 8 United~ GBR          2020-07-18 294066             64.4                  178
 9 United~ GBR          2020-07-19 294792             64.4                  179
10 United~ GBR          2020-07-20 295372             64.4                  180
# i 20 more rows
# i 2 more variables: t <dbl>, y <int>
```

The different criteria can be computed on the predictions made for that 30 days interval:

```r
crit_oos <- get_criteria(
  f = model_function,
  data = df_country_test,
  theta = params$coef_estimate
)
criteria_oos <- tibble(
  model_type = model_name,
  country = country_name,
  bind_rows(crit_oos)
)
criteria_oos
```

```
# A tibble: 1 x 5
  model_type country          AIC   BIC  RMSE
  <chr>      <chr>          <dbl> <dbl> <dbl>
1 Gompertz   United Kingdom  620.  626. 6564.
```

Let us now process the data so that we know, for each observations, whether the prediction corresponds to seen or unseen data:

```r
obs <- df_country_training$value
type_obs <- rep("obs", length(obs))
if (length(obs) < horizon_pred) {
  obs <- c(obs, rep(NA, horizon_pred-length(obs)))
  type_obs <- c(
    type_obs,
    rep("out_of_sample", horizon_pred - length(type_obs))
  )
}
head(obs)
```

```
[1] 2 2 2 8 8 9
```

```r
tail(obs)
```

```
[1] NA NA NA NA NA NA
```

```r
length(obs)
```

```
[1] 250
```

```r
head(type_obs)
```

```
[1] "obs" "obs" "obs" "obs" "obs" "obs"
```

```r
tail(type_obs)
```

```
[1] "out_of_sample" "out_of_sample" "out_of_sample" "out_of_sample"
[5] "out_of_sample" "out_of_sample"
```

```r
length(type_obs)
```

```
[1] 250
```

Let us create a vector of corresponding dates for those:

```r
dates <- df_country_training$date
if (length(dates) < horizon_pred) {
  dates <- dates[1] + lubridate::ddays(seq_len(horizon_pred) - 1)
}
head(dates)
```

```
[1] "2020-01-31" "2020-02-01" "2020-02-02" "2020-02-03" "2020-02-04"
[6] "2020-02-05"
```

```r
tail(dates)
```

```
[1] "2020-10-01" "2020-10-02" "2020-10-03" "2020-10-04" "2020-10-05"
[6] "2020-10-06"
```

```r
length(dates)
```

```
[1] 250
```

Then, the predictions obtained with the model can be saved in a table:

```r
fitted_val <- tibble(
  country  = !!country_name,
  index    = seq_len(horizon_pred) - 1,
  value    = obs,
  type_obs = type_obs,
  date     = dates
) |>
  mutate(
    model_type   = model_name,
    fitted_value = model_function(
      theta = params$coef_estimate,
      x = index)
  )
fitted_val
```

```
# A tibble: 250 x 7
   country        index value type_obs date       model_type fitted_value
   <chr>          <dbl> <int> <chr>    <date>     <chr>             <dbl>
```

```
 1 United Kingdom     0      2 obs      2020-01-31 Gompertz     0.00000149
 2 United Kingdom     1      2 obs      2020-02-01 Gompertz     0.00000446
 3 United Kingdom     2      2 obs      2020-02-02 Gompertz     0.0000127
 4 United Kingdom     3      8 obs      2020-02-03 Gompertz     0.0000347
 5 United Kingdom     4      8 obs      2020-02-04 Gompertz     0.0000909
 6 United Kingdom     5      9 obs      2020-02-05 Gompertz     0.000228
 7 United Kingdom     6      9 obs      2020-02-06 Gompertz     0.000552
 8 United Kingdom     7      9 obs      2020-02-07 Gompertz     0.00129
 9 United Kingdom     8     13 obs      2020-02-08 Gompertz     0.00289
10 United Kingdom     9     14 obs      2020-02-09 Gompertz     0.00628
# i 240 more rows
```

Using the second order derivative with respect to time of the model, we can compute the three key moments, and store those in a table:

```r
# Key moments
tau <- out$par$tau

# First wave
acceleration <- model_function_dd(
  theta = out$par, x = seq(1, tau)
) |> which.max()

deceleration <- tau +
  model_function_dd(
    theta = out$par,
    x = seq(tau, horizon_pred)
  ) |> which.min()

# Peak
abs_second_d_vals <- abs(
  model_function_dd(
    theta = out$par,
    seq(acceleration, deceleration)
  )
)
abs_second_d_vals <- abs_second_d_vals / min(abs_second_d_vals)
peak <- mean(which(abs_second_d_vals == 1)) + acceleration

# Relative speed at max speed
relative_speed <- model_function_d(theta = out$par, x = peak) /
  model_function(theta = out$par, x = peak)
```

```r
key_moments <- tibble(
  country  = country_name,
  model_type = model_name,
  t = round(c(acceleration, peak, deceleration)),
  value = model_function(theta = out$par, t),
  moment = c("acceleration", "peak", "deceleration")
) |>
  bind_rows(
    tibble(
      country = country_name,
      model_type = model_name,
      t = peak,
      value = relative_speed,
      moment = "relative_speed"
    )
  ) |>
  mutate(
    date = first(df_country_training$date) + lubridate::ddays(t) - 1
  )
key_moments
```

```
# A tibble: 4 x 6
  country        model_type     t     value moment         date
  <chr>          <chr>      <dbl>     <dbl> <chr>          <date>
1 United Kingdom Gompertz      53  20708.   acceleration   2020-03-23
2 United Kingdom Gompertz      77 115420.   peak           2020-04-16
3 United Kingdom Gompertz      99 206618.   deceleration   2020-05-08
4 United Kingdom Gompertz      77      1.08 relative_speed 2020-04-16
```

We can plot the results showing observed values and predicted ones. First, let us prepare the data.

```r
the_dates <- map_df("United Kingdom", get_dates, type = "cases")
df_key_moments <-
    key_moments |>
    left_join(colour_table, by = "country")
df_plot <-
  fitted_val |>
  pivot_longer(cols = c(value, fitted_value)) |>
  filter(! (type_obs == "out_of_sample" & name == "value") ) |>
```

```r
    mutate(linetype = str_c(type_obs, "_", name)) |>
    mutate(
      linetype = ifelse(
        linetype %in% c("obs_value", "out_of_sample_value"),
        yes = "obs",
        no = linetype
      )
    ) |>
    select(country, date, value, linetype)

  df_plot <-
    df_plot |>
    left_join(colour_table, by = "country") |>
    mutate(
      linetype = factor(
        linetype,
        levels = c("obs", "obs_fitted_value", "out_of_sample_fitted_value"),
        labels = c("Obs.", "Fitted values", "Predictions")
      )
    )

  df_dots <-
    the_dates |>
    mutate(end_sample = start_reduce_restrict + lubridate::ddays(60)) |>
    left_join(
      df_plot |>
        filter(linetype == "Obs.") |>
        select(country, date, value),
      by = c("end_sample" = "date", "country" = "country")
    ) |>
    left_join(colour_table, by = "country")
```

And then we can plot:

```r
  ggplot(data = df_plot) +
      geom_line(
        mapping = aes(
          x = date, y = value,
          colour = country_code, linetype = linetype)
      ) +
      geom_point(
        data = df_dots,
        mapping = aes(x = end_sample, y = value,
                      colour = country_code),
        shape = 2,
        show.legend = FALSE
      ) +
      geom_point(
        data = df_key_moments,
        mapping = aes(
          x = date, y = value,
          colour = country_code
        )
      ) +
      scale_colour_manual(NULL, values = colour_countries) +
      scale_y_continuous(labels = comma) +
      labs(x = NULL, y = "Cases") +
      scale_x_date(
        labels = date_format("%b %d"),
        breaks = lubridate::ymd(lubridate::pretty_dates(df_plot$date, n = 5))) +
      scale_linetype_manual(
        NULL,
        values = c("Obs." = "solid",
                   "Fitted values" = "dashed",
                   "Predictions" = "dotted")) +
      theme_paper()
```

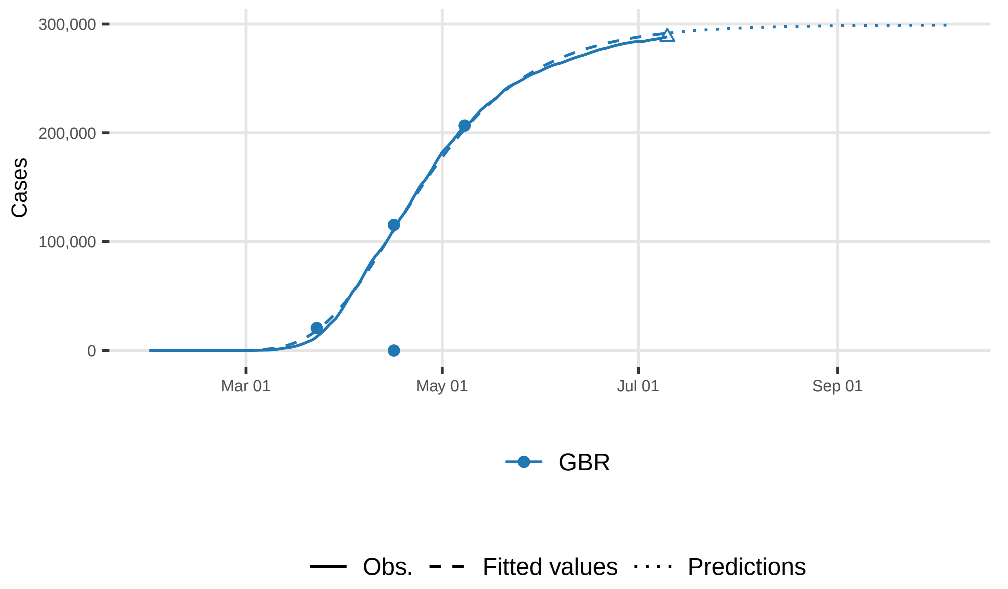


Figure 7.1: Prediction of cases using the Gompertz model, for the UK (single wave)

In Figure 7.1, big dots represent the three stages of the epidemic. A triangle indicates the end of the estimation period corresponding to 60 days after the end of the most severe confinement. The Solid line corresponds to the observed series and dotted lines the predictions using Gompertz model.

7.3 Double sigmoid functions to account for a second wave

Let us now turn to double sigmoid functions to model two waves of the epidemic.

7.3.1 Double Logistic

The double logistic function, its first and second derivatives with respect to time can be coded as follows:

```
# Double logistic
double_logistic_f <-  function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
```

```r
  r2 <- theta[["r2"]]

  (K1/(1 + exp(-r1 * (x - tau1)))) + ((K2 - K1)/(1 + exp(-r2 * (x - tau2))))
}
# Double logistic in two parts
double_logistic_f_2 <- function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]

  f_1 <- K1 / (1 + exp(-r1 * (x - tau1)))
  f_2 <- (K2 - K1)/(1 + exp(-r2 * (x - tau2)))
  tibble(x = x, f_1 = f_1, f_2 = f_2)
}

# First derivative
double_logistic_f_first_d <-  function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]

  (r1 * K1 * exp(-r1 * (x - tau1))) / ((exp(-r1 * (x-tau1)) + 1)^2) +
    (r2 * (K2-K1) * exp(-r2 * (x - tau2))) / ((exp(-r2 * (x-tau2)) + 1)^2)
}
# Second derivative
double_logistic_f_second_d <-  function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]

  K1 *  ((2 * r1^2 * exp(-2 * r1 * (x - tau1)))/(exp(-r1 * (x - tau1)) + 1)^3 -
          ( r1^2 * exp(- r1 * (x - tau1)))/(exp(-r1 * (x - tau1)) + 1)^2) +
    (K2-K1) *
    ((2 * r2^2 * exp(-2 * r2 * (x - tau2)))/(exp(-r2 * (x - tau2)) + 1)^3 -
       ( r2^2 * exp(- r2 * (x - tau2)))/(exp(-r2 * (x - tau2)) + 1)^2)

}
```

7.3.2 Double Gompertz Model

The double Gompertz function, its first and second derivatives with respect to time can be coded as follows:

```
# Double-Gompertz function
#' @param theta vector of named parameters
#' @param x observation / training example
double_gompertz_f <- function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]

  f_1 <- K1 * exp(-exp(-r1 * (x - tau1)))
  f_2 <- (K2-K1) * exp(-exp(-r2 * (x - tau2)))

  f_1 + f_2
}

#' Double-Gompertz function in two parts
#'
double_gompertz_f_2 <- function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]

  f_1 <- K1 * exp(-exp(-r1 * (x - tau1)))
  f_2 <- (K2-K1) * exp(-exp(-r2 * (x-tau2)))
  tibble(x = x, f_1 = f_1, f_2 = f_2)
}

#' First derivative
#'
double_gompertz_f_first_d <- function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]

  f_1_d <- r1 * K1 * exp(-r1 * (x - tau1) - exp(- r1 * (x - tau1)))
  f_2_d <- r2 * (K2-K1) * exp(-r2 * (x - tau2) - exp(- r2 * (x - tau2)))

  f_1_d + f_2_d
}

#' Second derivative
double_gompertz_f_second_d <- function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]

  -K1 * r1^2 * exp(-r1*(x - tau1)) * exp(-exp(-r1*(x - tau1))) +
    K1 * r1^2 * (exp( -r1*(x - tau1)))^2 * exp(-exp(-r1*(x - tau1))) -
    (K2 - K1) * r2^2 * exp( -r2 * (x - tau2)) * exp( -exp(-r2*(x - tau2))) +
    (K2 - K1) * r2^2 * (exp( -r2 * (x - tau2)))^2 * exp(-exp(-r2*(x-tau2)))
}
```

7.3.3 Double Richards Model

The double Richards function, its first and second derivatives with respect to time can be coded as follows:

```r
#' Double Richards
#'
double_richards_f <- function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  delta1 <- theta[["delta1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]
  delta2 <- theta[["delta2"]]

  K1 * (1 + delta1 * exp(-r1 * (x - tau1)))^(-1 / delta1) +
    (K2 - K1) * (1 + delta2 * exp(-r2 * (x - tau2)))^(-1 / delta2)
}

#' Double Richards in two parts
#'
double_richards_f_2 <- function(theta, x) {

  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  delta1 <- theta[["delta1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]
  delta2 <- theta[["delta2"]]

  f_1 <- K1 * (1 + delta1 * exp(-r1 * (x - tau1)))^(-1 / delta1)
  f_2 <- (K2 - K1) * (1 + delta2 * exp(-r2 * (x - tau2)))^(-1 / delta2)

  tibble(x = x, f_1 = f_1, f_2 = f_2)
}

#' First derivative
#'
double_richards_f_first_d <- function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  delta1 <- theta[["delta1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]
  delta2 <- theta[["delta2"]]

  r1 * K1 * exp(-r1 * (x - tau1)) *
    (delta1 * exp(-r1 * (x - tau1)) + 1)^(-1/delta1 - 1) +
    r2 * (K2-K1) * exp(-r2 * (x - tau2)) *
    (delta2 * exp(-r2 * (x - tau2)) + 1)^(-1/delta2 - 1)
}

#' Second derivative
#'
double_richards_f_second_d <- function(theta, x) {
  K1 <- theta[["K1"]]
  tau1 <- theta[["tau1"]]
  r1 <- theta[["r1"]]
  delta1 <- theta[["delta1"]]
  K2 <- theta[["K2"]]
  tau2 <- theta[["tau2"]]
  r2 <- theta[["r2"]]
  delta2 <- theta[["delta2"]]

  K1 * (r1^2 * (-1/delta1 - 1) * delta1 * (-exp(-2 * r1 * (x - tau1))) *
          (delta1 * exp(-r1 * (x - tau1)) + 1)^(-1/delta1 - 2) - r1^2 *
          exp(-r1 * (x - tau1)) *
          (delta1 * exp(-r1 * (x - tau1)) + 1)^(-1/delta1 - 1)) +

    (K2 - K1) *
    (r2^2 * (-1 / delta2 - 1) *
       delta2 * (-exp(-2 * r2 * (x - tau2))) *
       (delta2 * exp(-r2 * (x - tau2)) + 1)^(-1/delta2 - 2) -
       r2^2 *
       exp(-r2 * (x - tau2)) *
       (delta2 * exp(-r2 * (x - tau2)) + 1)^(-1/delta2 - 1))
}
```

7.3.4 Help functions

Once again, we will rely on some help functions. These were defined previously:

- `get_dates()`: returns the dates relative to the sample of a specific country
- `get_values_country()`: returns the sample data for a single country
- `loss_function()`: the loss function that is minimized to find the model estimates
- `get_criteria()`: returns goodness of fit criteria for a model estimated with nls.lm.

7.3.5 Example for the UK

Let us consider an example, as was done for the first wave: that of the UK. The next section will show how the code can be used to estimate all the three double sigmoid models to each country.

First, we need to decide the time horizon for the fitted values that will be returned. If the horizon is greater than the end of observed values, we will return out-of-sample predictions.

```r
end_date_sample <- lubridate::ymd("2020-09-28")
end_date_data <- max(confirmed_df$date)
out_of_sample_horizon <- seq(end_date_sample, end_date_data, by = "day") |>
  length()

horizon_pred <- 310
```

So, we want to focus on the UK:

```r
country_name <- "United Kingdom"
pop_country <-
  population |> filter(country == !!country_name) |>
  magrittr::extract2("pop")
pop_country
```

```
[1] 6.7e+07
```

Let us show here how we can fit a Double Gompertz model, on the number of confirmed cases:

```r
model_name <- "Double_Gompertz"
type <- "cases"
```

Setting the `sample` argument in our function `get_values_country()` allows us to extract the whole sample for a specific country. We can then store each element of the returned list in a single variable:

```r
data_country <- get_values_country(
  country_name,
  sample = "all",
  type = type
)
```

The training sample:

```r
df_country <- data_country$df_country
df_country
```

```
# A tibble: 275 x 8
   country   country_code date       value stringency_index days_since_2020_01_22
   <chr>     <chr>        <date>     <int>            <dbl>                 <dbl>
 1 United ~ GBR          2020-01-31     2             8.33                     9
 2 United ~ GBR          2020-02-01     2             8.33                    10
 3 United ~ GBR          2020-02-02     2            11.1                     11
 4 United ~ GBR          2020-02-03     8            11.1                     12
 5 United ~ GBR          2020-02-04     8            11.1                     13
 6 United ~ GBR          2020-02-05     9            11.1                     14
 7 United ~ GBR          2020-02-06     9            11.1                     15
 8 United ~ GBR          2020-02-07     9            11.1                     16
 9 United ~ GBR          2020-02-08    13            11.1                     17
10 United ~ GBR          2020-02-09    14            11.1                     18
# i 265 more rows
# i 2 more variables: t <dbl>, y <int>
```

And the dates of interest:

```r
dates_country <- data_country$dates_country
dates_country
```

```
# A tibble: 1 x 7
  country        start_first_wave start_high_stringency start_reduce_restrict
  <chr>          <date>           <date>                <date>
1 United Kingdom 2020-01-31       2020-03-23            2020-05-11
# i 3 more variables: start_date_sample_second_wave <date>,
#   length_high_stringency <dbl>, max_severity <dbl>
```

We previoulsy defined the Double Gompertz function and its first and second order derivatives. Let us use these:

```
model_function <- double_gompertz_f
model_function_2 <- double_logistic_f_2
model_function_d <- double_gompertz_f_first_d
model_function_dd <- double_gompertz_f_second_d
```

The training sample is the whole sample here:

```
df_country_training <- df_country
```

Let us set a bounding value for τ_2:

```
nf_tau <- 280
```

Some starting values for the optimization algorithm:

```
start <- list(
  K1     = last(df_country$y)/2,
  tau1   = 70,
  r1     = .12,
  K2     = last(df_country$y),
  tau2   = 200,
  r2     = 0.05
)
```

Here are the constraints that we set on the parameters of the model:

```
lower_c <- c(K1 = 0, tau1 = 50, r1 = 0,
             K2 = 0, tau2 = 0,  r2 = 0)

upper_c <- c(K1 = pop_country, tau1 = Inf,    r1 = Inf,
             K2 = pop_country, tau2 = nf_tau, r2 = Inf)
```

We can then use the `nls.lm()` function to minimize the loss function (`loss_function()`) for the Double Gompertz model whose function is stored in `model_function()`:

```
out <- nls.lm(
  par = start,
  fn = loss_function,
  y = df_country$y,
  t = df_country$t,
  fun = model_function,
  lower = lower_c,
  upper = upper_c,
  nls.lm.control(maxiter = 250),
  jac=NULL
)
```

The estimates can be stored in a table:

```
params <- tibble(
  model_type = model_name,
  country = country_name,
  coef_estimate_name = names(coef(out)),
  coef_estimate = coef(out)
)
params
```

```
# A tibble: 6 x 4
  model_type      country        coef_estimate_name coef_estimate
  <chr>           <chr>          <chr>                      <dbl>
1 Double_Gompertz United Kingdom K1                    315516.
2 Double_Gompertz United Kingdom tau1                      77.6
3 Double_Gompertz United Kingdom r1                         0.0371
4 Double_Gompertz United Kingdom K2                   2510392.
5 Double_Gompertz United Kingdom tau2                     280
6 Double_Gompertz United Kingdom r2                         0.0266
```

The goodness of fit can be obtained using the `get_criteria()` function that was previously defined:

```
crit <- get_criteria(
  f = model_function,
  data = df_country_training,
  theta = params$coef_estimate
)

criteria <- tibble(
  model_type = model_name,
  country = country_name,
  bind_rows(crit)
)
criteria
```

```
# A tibble: 1 x 5
  model_type      country           AIC   BIC  RMSE
  <chr>           <chr>           <dbl> <dbl> <dbl>
1 Double_Gompertz United Kingdom 5817. 5842. 9239.
```

Now, let us get the fitted values, up to the desired horizon as set in `horizon_pred`:

```
# Observed values
obs <- df_country_training$value
type_obs <- rep("obs", length(obs))
if (length(obs) < horizon_pred) {
  obs <- c(obs, rep(NA, horizon_pred-length(obs)))
  type_obs <- c(
    type_obs,
    rep("out_of_sample", horizon_pred-length(type_obs))
  )
}
head(obs)
```

```
[1] 2 2 2 8 8 9
```

```
tail(obs)
```

```
[1] NA NA NA NA NA NA
```

```
length(obs)
```

```
[1] 310
```

```
head(type_obs)
```

```
[1] "obs" "obs" "obs" "obs" "obs" "obs"
```

```
tail(type_obs)
```

```
[1] "out_of_sample" "out_of_sample" "out_of_sample" "out_of_sample"
[5] "out_of_sample" "out_of_sample"
```

```r
length(type_obs)
```

```
[1] 310
```

The corresponding dates:

```r
dates <- df_country_training$date
if (length(dates) < horizon_pred) {
  dates <- dates[1] + lubridate::ddays(seq_len(horizon_pred) - 1)
}
head(dates)
```

```
[1] "2020-01-31" "2020-02-01" "2020-02-02" "2020-02-03" "2020-02-04"
[6] "2020-02-05"
```

```r
tail(dates)
```

```
[1] "2020-11-30" "2020-12-01" "2020-12-02" "2020-12-03" "2020-12-04"
[6] "2020-12-05"
```

```r
length(dates)
```

```
[1] 310
```

The fitted values can be stored in a table:

```r
fitted_val <- tibble(
  country  = !!country_name,
  index    = seq_len(horizon_pred) - 1,
  value    = obs,
  type_obs = type_obs,
  date     = dates
) |>
  mutate(
    model_type   = model_name,
    fitted_value = model_function(theta = params$coef_estimate, x = index)
  )
```

```r
fitted_val
```

```
# A tibble: 310 x 7
   country        index value type_obs date       model_type     fitted_value
   <chr>          <dbl> <int> <chr>    <date>     <chr>                 <dbl>
 1 United Kingdom     0     2 obs      2020-01-31 Double_Gompertz     0.00630
 2 United Kingdom     1     2 obs      2020-02-01 Double_Gompertz     0.0120
 3 United Kingdom     2     2 obs      2020-02-02 Double_Gompertz     0.0224
 4 United Kingdom     3     8 obs      2020-02-03 Double_Gompertz     0.0407
 5 United Kingdom     4     8 obs      2020-02-04 Double_Gompertz     0.0725
 6 United Kingdom     5     9 obs      2020-02-05 Double_Gompertz     0.126
 7 United Kingdom     6     9 obs      2020-02-06 Double_Gompertz     0.216
 8 United Kingdom     7     9 obs      2020-02-07 Double_Gompertz     0.362
 9 United Kingdom     8    13 obs      2020-02-08 Double_Gompertz     0.596
10 United Kingdom     9    14 obs      2020-02-09 Double_Gompertz     0.963
# i 300 more rows
```

We can also add the two components of the sigmoid, separately:

```r
fitted_val <-
  fitted_val |>
  mutate(
    f_1 = model_function_2(x = index, theta = params$coef_estimate)[["f_1"]],
    f_2 = model_function_2(x = index, theta = params$coef_estimate)[["f_2"]]
  )
```

The key moments can be computed, using the second derivative of the model function.

For the first wave, the acceleration point corresponds to the moment where the second derivative reaches its maximum value. We therefore look at the time between the outbreak and the midpoint of the second wave where the second derivative reaches its maximum.

Let us first get the midpoints of the sigmoids:

```r
# Estimated midpoints of the sigmoids
tau_1 <- out$par$tau1
tau_2 <- out$par$tau2
print(tau_1)
```

```
[1] 77.56061
```

```r
print(tau_2)
```

```
[1] 280
```

Then we can compute the acceleration point:

```r
acceleration_wave_1 <- model_function_dd(
  theta = out$par,
  x = seq(1, tau_1)
) |>
  which.max()
acceleration_wave_1
```

```
[1] 52
```

The deceleration point corresponds to the moment at which we observe the minimum acceleration:

```r
deceleration_wave_1 <-
  tau_1 + model_function_dd(
    theta = out$par,
    x = seq(tau_1, tau_2)
  ) |>
  which.min()
deceleration_wave_1
```

```
[1] 104.5606
```

Then we can compute the peak of the first wave, where the acceleration is null:

```r
abs_second_d_vals <- abs(
  model_function_dd(
    theta = out$par,
    seq(acceleration_wave_1, deceleration_wave_1)
  )
)
abs_second_d_vals <- abs_second_d_vals / min(abs_second_d_vals)
peak_1 <- mean(which(abs_second_d_vals == 1)) + acceleration_wave_1
peak_1
```

```
[1] 79
```

For the second wave, the acceleration is the moment where we observe the maximum acceleration. We look at the second derivative of the model after the deleceration of the first wave:

```
acceleration_wave_2 <-
  deceleration_wave_1 +
  model_function_dd(
    theta = out$par,
    seq(deceleration_wave_1, last(df_country_training$t))
  ) |>
  which.max()
```

The deceleration:

```
deceleration_wave_2 <-
  acceleration_wave_2 +
  model_function_dd(
    theta = out$par,
    seq(acceleration_wave_2, horizon_pred)
  ) |>
  which.min()
deceleration_wave_2
```

```
[1] 310.5606
```

And lastly, the peak of the second wave:

```
peak_2 <-
  model_function_dd(
    theta = out$par,
    seq(acceleration_wave_2, deceleration_wave_2)
  ) |>
  abs()
peak_2 <- which(peak_2 < 10) |>
  mean()
peak_2 <- peak_2 + acceleration_wave_2
peak_2
```

```
[1] 281.0606
```

We can also add the relative speed at each peak. For the first wave:

```r
relative_speed_1 <- model_function_d(theta = out$par, x = peak_1) /
  model_function(theta = out$par, x = peak_1)
relative_speed_1
```

```
[1] 0.03514423
```

and for the second:

```r
relative_speed_2 <- model_function_d(theta = out$par, x = peak_2) /
  model_function(theta = out$par, x = peak_2)
relative_speed_2
```

```
[1] 0.01873886
```

These moments can be stored in a table

```r
key_moments <- tibble(
  country  = country_name,
  model_type = model_name,
  t = c(
    acceleration_wave_1, peak_1, deceleration_wave_1,
    acceleration_wave_2, peak_2, deceleration_wave_2
  ),
  value = model_function(theta = out$par, t),
  moment = c(
    "acceleration 1", "peak 1", "deceleration 1",
    "acceleration 2", "peak 2", "deceleration 2"
  )
) |>
  bind_rows(
    tibble(
      country = country_name,
      model_type = model_name,
      t = c(peak_1, peak_2),
      value = c(relative_speed_1, relative_speed_2),
      moment = c("relative speed 1", "relative speed 2")
    )
  ) |>
  mutate(date = first(df_country_training$date) + lubridate::ddays(t) - 1)
key_moments
```

```
# A tibble: 8 x 6
  country        model_type           t       value moment    date
  <chr>          <chr>            <dbl>       <dbl> <chr>     <dttm>
1 United Kingdom Double_Gompertz   52    23923.     acceler~ 2020-03-22 23:59:59
2 United Kingdom Double_Gompertz   79   122263.     peak 1   2020-04-18 23:59:59
3 United Kingdom Double_Gompertz  105.  218473.     deceler~ 2020-05-14 13:27:15
4 United Kingdom Double_Gompertz  245.  483560.     acceler~ 2020-10-01 13:27:15
5 United Kingdom Double_Gompertz  281. 1145566.     peak 2   2020-11-07 01:27:15
6 United Kingdom Double_Gompertz  311. 1723791.     deceler~ 2020-12-06 13:27:15
7 United Kingdom Double_Gompertz   79        0.0351 relativ~ 2020-04-18 23:59:59
8 United Kingdom Double_Gompertz  281.       0.0187 relativ~ 2020-11-07 01:27:15
```

Let us prepare the data to make a plot.

```
the_dates <- map_df("United Kingdom", get_dates, type = "cases")
df_plot <-
  fitted_val |>
  pivot_longer(cols = c(value, fitted_value)) |>
  filter(! (type_obs == "out_of_sample" & name == "value") ) |>
  mutate(linetype = str_c(type_obs, "_", name)) |>
  mutate(
    linetype = factor(
      linetype,
      levels = c(
        "obs_value",
        "obs_fitted_value",
        "out_of_sample_fitted_value"
      ),
      labels = c(
        "Obs.",
        "Double Gompertz",
        "Double Gompertz"
      )
    )
  )
df_plot <-
  df_plot |>
  left_join(
```

```r
    colour_table,
     by = c("country")
  )

date_end_obs <-
  df_plot |>
  filter(linetype == "Obs.") |>
  arrange(desc(date)) |>
  slice(1) |>
  magrittr::extract2("date")

date_end_sample <- max(df_plot$date)
```

And the plot. The shaded area corresponds to out-of-sample predictions, starting with the end of the observed data (*i.e.*, September 28).

```r
ggplot(data = df_plot) +
  geom_line(
    mapping = aes(
      x = date, y = value,
      colour = country_code, linetype = linetype
    )
  ) +
    scale_colour_manual(NULL, values = colour_countries) +
    scale_y_continuous(labels = comma) +
    labs(x = NULL, y = "Cases") +
    scale_x_date(
      labels = date_format("%b %d %Y"),
      breaks = lubridate::ymd(lubridate::pretty_dates(df_plot$date, n = 5))
    ) +
    # scale_linetype_manual(
    #   NULL,
    #   values = c(
    #     "Obs." = "solid",
    #     "Double Richards" = "dashed",
    #     "Double Gompertz" = "dotted"
    #   )
    # ) +
    annotate(
      geom = "rect",
      xmin = date_end_obs, xmax = date_end_sample,
      ymin = -Inf, ymax = Inf,
      alpha = .2, fill = "grey"
    ) +
  theme_paper()
```

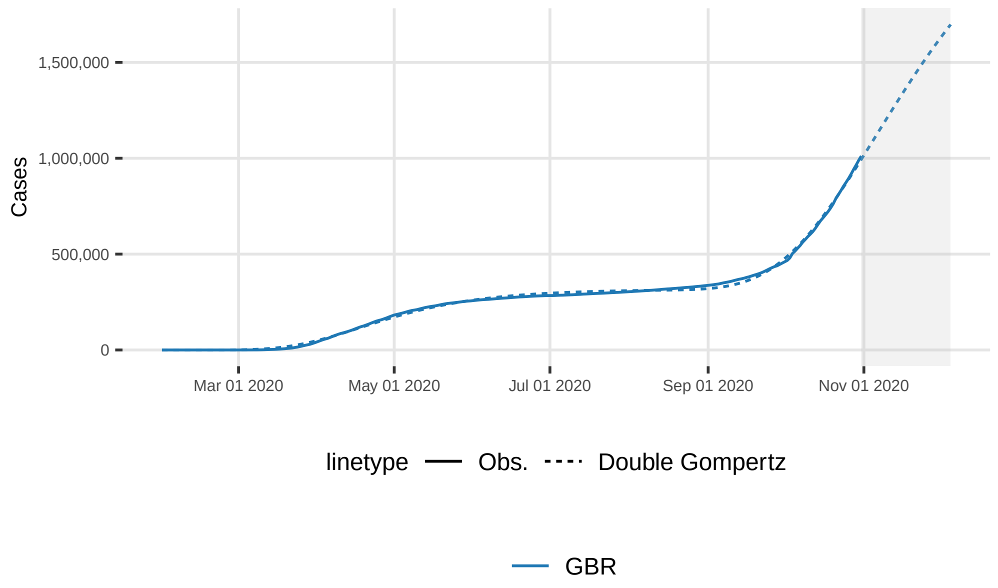


Figure 7.2: Predictions of the number of cases for the UK, with a double Gompertz model.

References

Adam, David. 2020. "Special Report: The Simulations Driving the World's Response to COVID-19." *Nature* 580: 316–18.

Bock, R. Darrell, Howard Wainer, Anne Petersen, David Thissen, James Murray, and Alex Roche. 1973. "A Parameterization for Individual Human Growth Curves." *Human Biology* 45 (1): 63–80. http://www.jstor.org/stable/41459847.

Cori, Anne, Neil M. Ferguson, Christophe Fraser, and Simon Cauchemez. 2013. "A New Framework and Software to Estimate Time-Varying Reproduction Numbers During Epidemics." *American Journal of Epidemiology* 178 (9): 1505–12. https://doi.org/10.1093/aje/kwt133.

Gompertz, Benjamin. 1825. "On the Nature of the Function Expressive of the Law of Human Mortality, and on a New Mode of Determining the Value of Life Contingencies." *Philosophical Transactions of the Royal Society of London* 115 (December): 513–83. https://doi.org/10.1098/rstl.1825.0026.

Kermack, William Ogilvy, and A. G. McKendrick. 1927. "A Contribution to the Mathematical Theory of Epidemics." *Proceedings of the Royal Society A* 115 (772): 700–721.

Lee, Se Yoon, Bowen Lei, and Bani Mallick. 2020. "Estimation of COVID-19 Spread Curves Integrating Global Data and Borrowing Information." *PLOS ONE* 15 (7): e0236860. https://doi.org/10.1371/journal.pone.0236860.

Li, Qun, Xuhua Guan, Peng Wu, Xiaoye Wang, Lei Zhou, Yeqing Tong, Ruiqi Ren, et al. 2020. "Early Transmission Dynamics in Wuhan, China, of Novel Coronavirus-Infected Pneumonia." *New England Journal of Medicine* 382 (13): 1199–1207. https://doi.org/10.1056/NEJMoa2001316.

Lipovetsky, Stan. 2010. "Double Logistic Curve in Regression Modeling." *Journal of Applied Statistics* 37 (11): 1785–93. https://doi.org/10.1080/02664760903093633.

Ma, Junling. 2020. "Estimating Epidemic Exponential Growth Rate and Basic Reproduction Number." *Infectious Disease Modelling* 5 (January): 129–41. https://doi.org/10.1016/j.idm.2019.12.009.

Moll, Benjamin. 2020. "Lockdowns in SIR Models." LSE.

Oswald, Stephen A., Ian C. T. Nisbet, Andre Chiaradia, and Jennifer M. Arnold. 2012. "FlexParamCurve: R Package for Flexible Fitting of Nonlinear Parametric Curves." *Methods in Ecology and Evolution* 3 (6): 1073–77. https://doi.org/10.1111/j.2041-210x.2012.00231.x.

Park, M., A. R. Cook, J. T. Lim, Y. Sun, and B. L. Dickens. 2020. "A Systematic Review of COVID-19 Epidemiology Based on Current Evidence." *Journal of Clinical Medicine* 9 (4). https://doi.org/10.3390/jcm9040967.

Richards, F. J. 1959. “A Flexible Growth Function for Empirical Use.” *Journal of Experimental Botany* 10 (2): 290–301. https://doi.org/10.1093/jxb/10.2.290.

Thissen, David, R. Darrell Bock, Howard Wainer, and Alex F. Roche. 1976. “Individual Growth in Stature: A Comparison of Four Growth Studies in the U.S.A.” *Annals of Human Biology* 3 (6): 529–42. https://doi.org/10.1080/03014467600001791.

Tjørve, Kathleen M. C., and Even Tjørve. 2017. “The Use of Gompertz Models in Growth Analyses, and New Gompertz-Model Approach: An Addition to the Unified-Richards Family.” *PLOS ONE* 12 (6): e0178691. https://doi.org/10.1371/journal.pone.0178691.

Toda, Alexis Akira. 2020. “Susceptible-Infected-Recovered (SIR) Dynamics of COVID-19 and Economic Impact.” arXiv:2003.11221v2.

Tsoularis, A., and J. Wallace. 2002. “Analysis of Logistic Growth Models.” *Mathematical Biosciences* 179 (1): 21–55. https://doi.org/10.1016/S0025-5564(02)00096-2.

Verhulst, P. F. 1845. “Recherches Mathématiques Sur La Loi d’accroissement de La Population.” *Nouveaux mémoires de l’Académie Royale Des Sciences Et Belles-Lettres de Bruxelles* 18: 14–54. https://eudml.org/doc/182533.

Wang, Huwen, Zezhou Wang, Yinqiao Dong, Ruijie Chang, Chen Xu, Xiaoyue Yu, Shuxian Zhang, et al. 2020. “Phase-Adjusted Estimation of the Number of Coronavirus Disease 2019 Cases in Wuhan, China.” *Cell Discovery* 6 (1): 1–8. https://doi.org/10.1038/s41421-020-0148-0.

Wang, Xiang-Sheng, Jianhong Wu, and Yong Yang. 2012. “Richards Model Revisited: Validation by and Application to Infection Dynamics.” *Journal of Theoretical Biology* 313: 12–19. https://doi.org/10.1016/j.jtbi.2012.07.024.