San Francisco data - New Model

Contents

1 Supporting tools

First, we load the libraries and custom functions that support the analysis.

```
library(bayesplot)
library(cmdstanr)
library(dplyr)
library(extraDistr)
library(GGally)
library(ggplot2)
library(reshape2)
library(ggpubr)
library(ggridges)
library(gridExtra)
library(kableExtra)
library(lubridate)
library(Metrics)
library(patchwork)
library(posterior)
library(purrr)
library(readr)
library(readsdr)
library(readxl)
library(writexl)
library(scales)
library(stringr)
library(tidyr)
library(viridisLite)

# Custom functions
source("./R/helpers.R")
source("./R/plots.R")
# source("./R/incidence_comparison.R")

# Custom model functions
source("./R/mdls/generate_deSolve_components.R")
source("./R/mdls/utils.R")
source("./R/mdls/arrange_variables.R")
source("./R/mdls/extract_variables.R")
source("./R/mdls/stan_ode_function_2.R")
source("./R/mdls/plots_2.R")

# Backup folder
fldr <- "./backup_objs/san_francisco_ext_202210"
dir.create(fldr, showWarnings = FALSE, recursive = TRUE)
```

2 Data

We read the *xls* file that contains the incidence data and transform it into a format suitable for the analysis.

```
# Change according to input data
flu_data <- read_xls("./data/rsif20060161s03.xls", range = "A6:C69") %>%
  rename(time = Time, y = Cases) %>%
  mutate(time = time + 1,
```

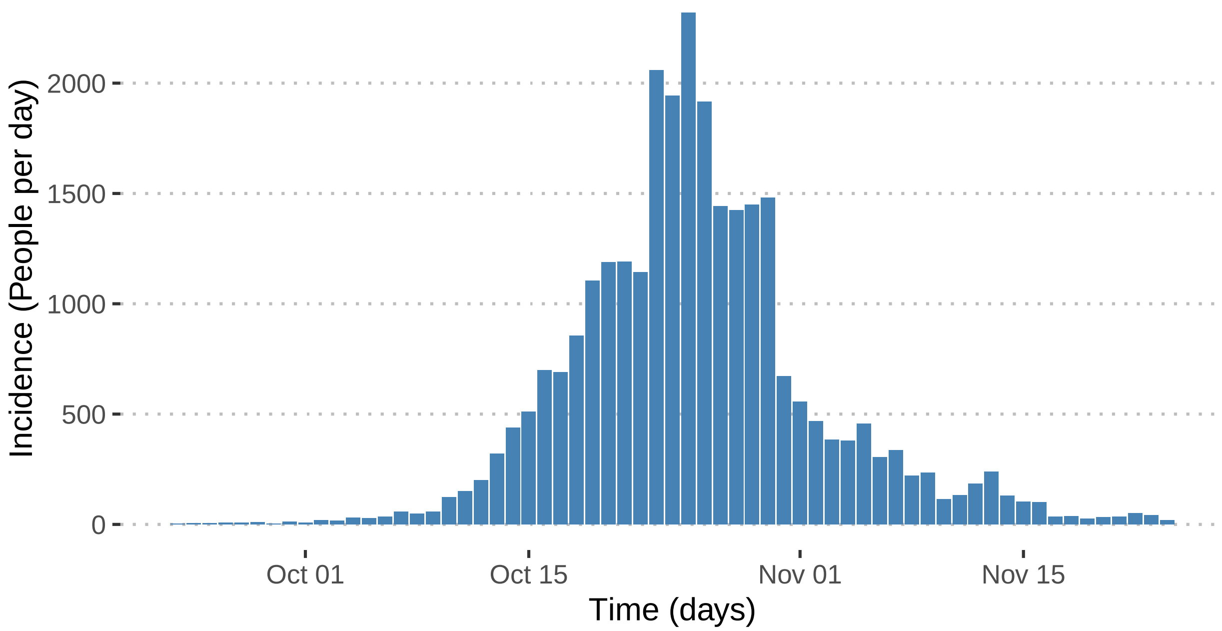
```
         Date = ymd(Date),
         Week = epiweek(Date))
flu_data
```

```
## # A tibble: 63 x 4
##    Date        time     y  Week
##    <date>     <dbl> <dbl> <dbl>
##  1 1918-09-23     1     4    39
##  2 1918-09-24     2     5    39
##  3 1918-09-25     3     5    39
##  4 1918-09-26     4     7    39
##  5 1918-09-27     5     9    39
##  6 1918-09-28     6    10    39
##  7 1918-09-29     7     4    40
##  8 1918-09-30     8    13    40
##  9 1918-10-01     9     9    40
## 10 1918-10-02    10    20    40
## # ... with 53 more rows
```

2.1 Daily data

For this graph and the remaining figures, *ggplot2* provides the framework for creating visualisations. Here, we show the daily case notifications during the autumn wave of the influenza pandemic (Spanish flu) in the city of San Francisco, California, from 1918 to 1919.

```
g <- ggplot(flu_data, aes(x = Date, y = y)) +
  geom_col(fill = "steelblue") +
  theme_pubclean() +
  labs(x = "Time (days)",
       y = "Incidence (People per day)")

print(g)
```



3 Calibration workflow

3.1 Prior information - $\pi(\theta)$

The following list corresponds to the time-independent variables and initial conditions of the SEIR model that will be fitted to the San Francisco data. For each parameter, we indicate their prior knowledge. We construct prior distributions for parameters for which we cannot obtain direct estimates and present them in a density plot.

Fisrt of all, as one of the parameters (the recovery rate for hospitalized class) of the model is calculated as follows:

$\gamma_2 = 1/(1/\gamma_1 - 1/\alpha)$

If γ_1 is greater than α, the parameter would take a negative value, which is impossible.

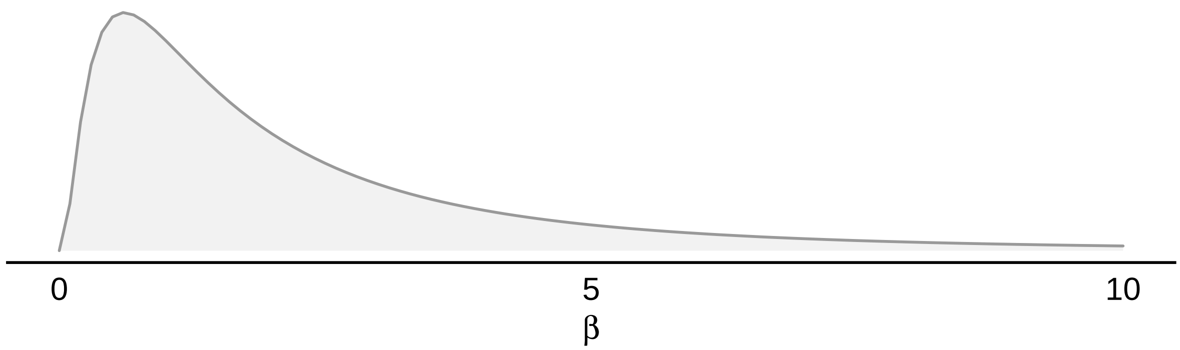
To solve this, the following can be done:

$\gamma_1 = \omega * \alpha$

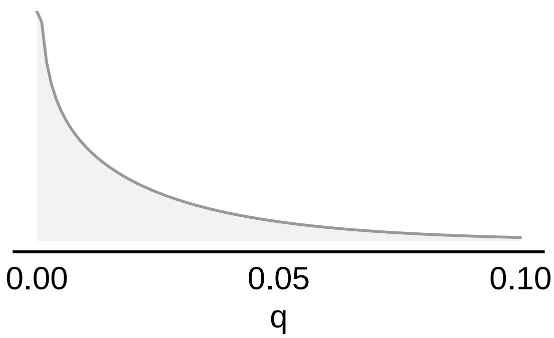
Where ω is a new parameter bounded between 0 and 1. In this way, we always make sure that γ_1 is never greater that α. We would estimate ω instead of γ_1 so that we could work with less stringent priors.

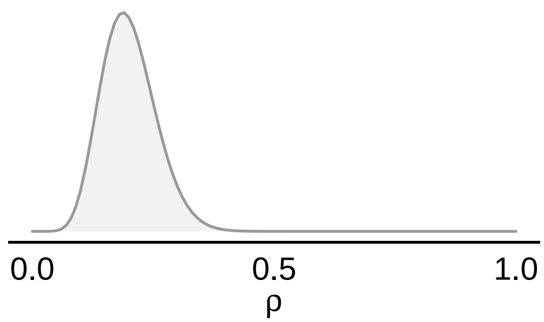
- Initial population: $N(0) = 550000$ [People]
- Initial death: $D(0) = 0$ [People]
- Initial recovered: $R(0) = 0$ [People]
- Initial diagnosed and reported: $J(0) = 0$ [People]
- Initial asymptomatic and partially infectious: $A(0) = 0$ [People]
- Initial susceptible: $S(0)) = N(0)$ - $E(0)$ - $I(0)$ [People]
-
- Birth and natural death rates: $\mu = 1$ / (60 * 365) [1 / day]
- Rate of progression to infectious: $k = 1$ / 1.9 [1 / day]
-
- Recovery rate: $\gamma_1 = \omega * \alpha$ [1 / day]
- Recovery rate for hospitalized class: $\gamma_2 = \frac{1}{\frac{1}{\gamma_1} - \frac{1}{\alpha}}$ [1 / day]
- Mortality rate: $\delta = \frac{CFP}{1-CFP}(\mu + \gamma_2)$ [1 / day] (being Case fatality proportion CFP = deaths / cases = $\frac{1908}{28310} = 0.067$)
-
-
- Initial infected: $I(0) \sim lognormal(1, 1)$ [People]
- Initial exposed: $E(0) \sim lognormal(1, 1)$ [People]
- Rate of effective contacts per infected individual: $\beta \sim lognormal(0.5, 1)$ [1 / day]
- Relative infectiousness of the asymptomatic class: $q \sim beta(0.75, 30)$
- Proportion of clinical infections: $\rho \sim beta(10, 40)$
- Diagnostic rate $\alpha \sim lognormal(-0.75, 0.1)$ [1 / day]
- Auxiliary parameter: $\omega \sim beta(2, 2)$

```r
g1 <- ggplot(NULL, aes(c(0, 10))) +
  geom_area(stat = "function", fun = dlnorm, fill = "grey95",
            colour = "grey60", args = list(meanlog = 0.5, sdlog = 1)) +
  scale_x_continuous(breaks = c(0, 5, 10)) +
  theme_pubr() +
  labs(y = "",
       x = bquote(beta)) +
  theme(axis.line.y = element_blank(),
        axis.ticks = element_blank(),
        axis.text.y = element_blank())

g1
```

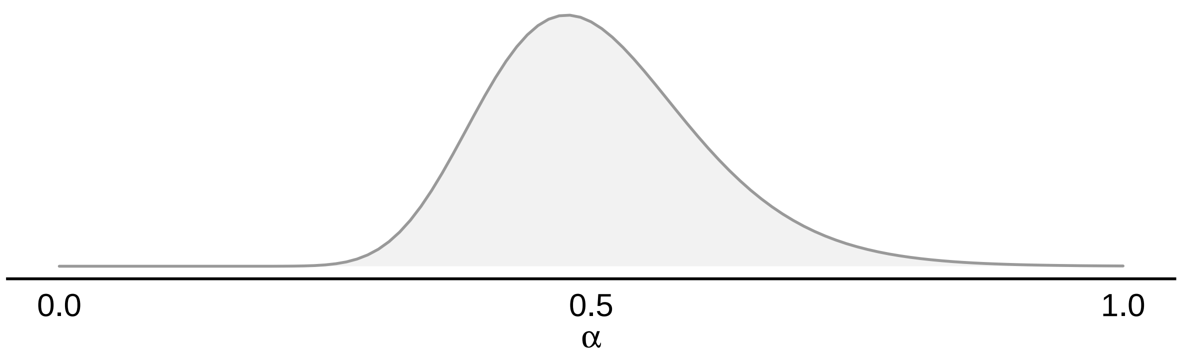



```r
g2   <- ggplot(NULL, aes(c(0, 0.1))) +
   geom_area(stat = "function", fun = dbeta, fill = "grey95",
            colour = "grey60", args = list(shape1 = 0.75, shape2 = 30)) +
  scale_x_continuous(breaks = c(0, 0.05, 0.1)) +
  theme_pubr() +
  labs(y = "",
       x = "q") +
  theme(axis.line.y = element_blank(),
        axis.ticks = element_blank(),
        axis.text.y = element_blank())

g3   <- ggplot(NULL, aes(c(0, 1))) +
   geom_area(stat = "function", fun = dbeta, fill = "grey95",
            colour = "grey60", args = list(shape1 = 10, shape2 = 40)) +
  scale_x_continuous(breaks = c(0, 0.5, 1)) +
  theme_pubr() +
  labs(y = "",
       x = bquote(rho)) +
  theme(axis.line.y = element_blank(),
        axis.ticks = element_blank(),
        axis.text.y = element_blank())

print(g2 + g3)
```

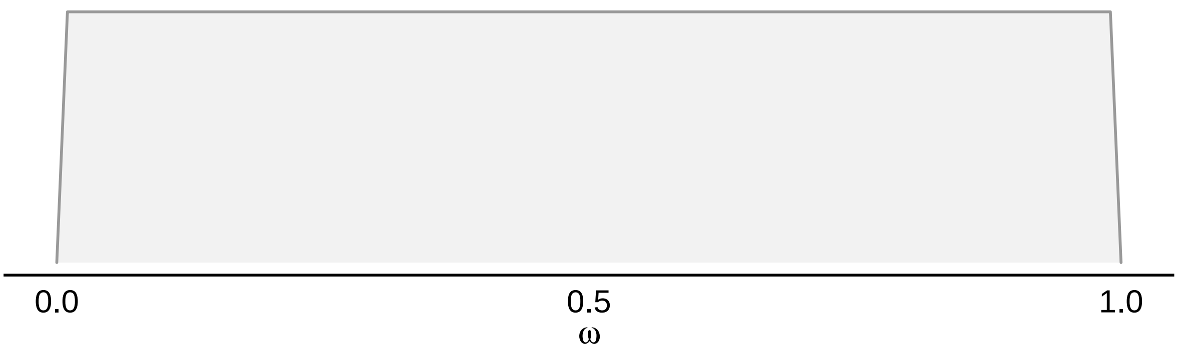




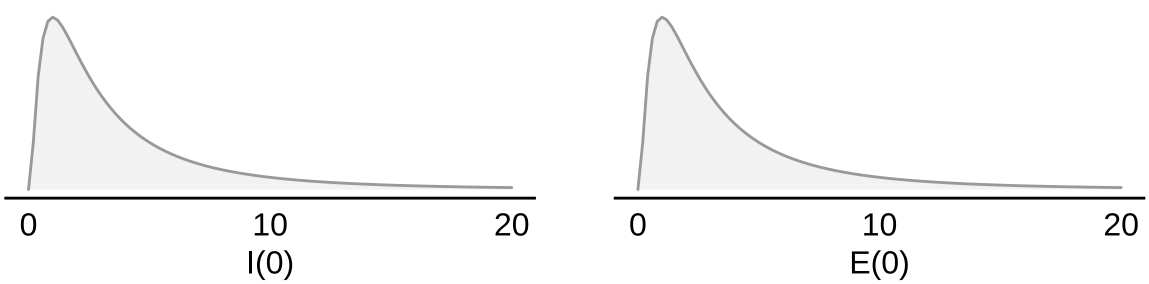

```
g4 <- ggplot(NULL, aes(c(0, 1))) +
  geom_area(stat = "function", fun = dlnorm, fill = "grey95",
            colour = "grey60", args = list(meanlog = -0.7, sdlog = 0.2)) +
  scale_x_continuous(breaks = c(0, 0.5, 1)) +
  theme_pubr() +
  labs(y = "",
       x = bquote(alpha)) +
  theme(axis.line.y = element_blank(),
        axis.ticks = element_blank(),
        axis.text.y = element_blank())

g4
```




```
g5   <- ggplot(NULL, aes(c(0, 1))) +
   geom_area(stat = "function", fun = dunif, fill = "grey95",
            colour = "grey60", args = list(min = 0.01, max = 0.99)) +
  scale_x_continuous(breaks = c(0, 0.5, 1)) +
  theme_pubr() +
  labs(y = "",
       x = bquote(omega)) +
  theme(axis.line.y = element_blank(),
        axis.ticks = element_blank(),
        axis.text.y = element_blank())

g5
```




```
g6 <- ggplot(NULL, aes(c(0, 20))) +
  geom_area(stat = "function", fun = dlnorm, fill = "grey95",
            colour = "grey60", args = list(meanlog = 1, sdlog = 1)) +
  scale_x_continuous(breaks = c(0, 10, 20)) +
  theme_pubr() +
  labs(y = "",
       x = "I(0)") +
  theme(axis.line.y = element_blank(),
        axis.ticks = element_blank(),
        axis.text.y = element_blank())

g7   <- g6 + labs(y = "",
       x = "E(0)",
       title = "")

print(g6 + g7)
```



3.2 Prior predictive checks

First, we describe the SEIR model.

```
# parameters
stp_time <- nrow(flu_data)
parameters <- list(start = 1, stop = stp_time-1, dt = 1/32)

#levels
levels <- list(
  # The entire population is assumed susceptible at the beginning of the pandemic wave
  list(name = "S", equation = "BR-IR-HR1", initValue = 550000),
  list(name = "E", equation = "IR-ER-HR2", initValue = 0),
  list(name = "A", equation = "AR-RR11-HR3", initValue = 0),
  list(name = "I", equation = "PR-DR-RR12-HR4", initValue = 0),
  list(name = "J", equation = "DR-RR2-MR-HR5", initValue = 0),
  list(name = "R", equation = "RR11+RR12+RR2-HR6", initValue = 0),
```

```
  list(name = "D", equation = "MR", initValue = 0),
  list(name = "C", equation = "DR", initValue = 0)
)

# variables
vars <- list(
  list(name = "gamma_1", equation = "omega*alpha"),
  list(name = "DR", equation = "I*alpha"),
  list(name = "delta", equation = "(cfp/(1-cfp))*(mu+gamma_2)"),
  list(name = "MR", equation = "J*delta"),
  list(name = "gamma_2", equation = "1/((1/gamma_1)-(1/alpha))"),
  list(name = "RR2", equation = "J*gamma_2"),
  list(name = "RR12", equation = "I*gamma_1"),
  list(name = "RR11", equation = "A*gamma_1"),
  list(name = "PR", equation = "rho*ER"),
  list(name = "AR", equation = "(1-rho)*ER"),
  list(name = "ER", equation = "E*k"),
  list(name = "lambda", equation = "((I+J+q*A)*beta)/N"),
  list(name = "IR", equation = "lambda*S"),
  list(name = "HR6", equation = "mu*R"),
  list(name = "HR5", equation = "mu*J"),
  list(name = "HR4", equation = "mu*I"),
  list(name = "HR3", equation = "mu*A"),
  list(name = "HR2", equation = "mu*E"),
  list(name = "HR1", equation = "mu*S"),
  list(name = "BR", equation = "mu*N"),
  list(name = "N", equation = "S+E+I+A+J+R")
)
variables <- arrange_variables(vars)

# constants
# According to the tests performed, these initial values do not affect the first part of the analysis.
# Check for the second part
constants <- list(
  list(name = "mu", value = 1/(60*365)),  # As specified in the paper
  list(name = "beta", value = 1.25),
  list(name = "q", value = 0.02),
  list(name = "k", value = 1/1.9),  # As specified in the paper
  list(name = "rho", value = 0.36),
  # list(name = "gamma_1", value = 0.4),
  list(name = "omega", value = 0.5),
  list(name = "alpha", value = 1),
  list(name = "cfp", value = 0.067),  # As specified in the paper
  list(name = "I0", value = 0),
  list(name = "E0", value = 0)
)

model_structure <- list(parameters = parameters,
                        levels = levels,
                        variables = variables,
                        constants = constants)

deSolve_components <- get_deSolve_elems(model_structure)
```

```r
model <- list(
  description = list(constants = constants),
  deSolve_components = deSolve_components
)
```

Then, we draw 500 samples from the prior distribution and simulate the model with these inputs. Given the computational burden of this process, we save the results in an RDS file.

```r
file_path <- file.path(fldr, "prior_sims.rds")

n_sims     <- 500
pop_size   <- 550000

if(!file.exists(file_path)) {
  set.seed(300194)

  E_0_sims      <- rlnorm(n_sims, 1, 1)
  I_0_sims      <- rlnorm(n_sims, 1, 1)
  beta_sims     <- rlnorm(n_sims, 0.5, 1)
  q_sims        <- rbeta(n_sims, 0.75, 30)
  rho_sims      <- rbeta(n_sims, 10, 40)
  alpha_sims    <- rlnorm(n_sims, -0.7, 0.2)
  omega_sims    <- runif(n_sims, 0.01, 0.99)

  consts_df  <- data.frame(beta     = beta_sims,
                           q        = q_sims,
                           rho      = rho_sims,
                           omega    = omega_sims,
                           alpha    = alpha_sims)

  stocks_df  <- data.frame(S = pop_size - E_0_sims - I_0_sims,
                           E = E_0_sims,
                           I = I_0_sims,
                           C = I_0_sims)

  sens_o <- sd_sensitivity_run(model$deSolve_components, start_time = 0,
                               stop_time = stp_time, timestep = 1 / 32,
                               multicore = TRUE, n_cores = 4,
                               integ_method = "rk4", stocks_df = stocks_df,
                               consts_df = consts_df)
  saveRDS(sens_o, file_path)
} else {
  sens_o <- readRDS(file_path)
}
```

We check if the data obtained (sens_o) contain any NA values

```r
anyNA(sens_o)
```

```
## [1] FALSE
```

And we show the values of the variables to see at a glance the range of values in which they move (minimum, maximum, average value…).

```r
summary(sens_o)
```

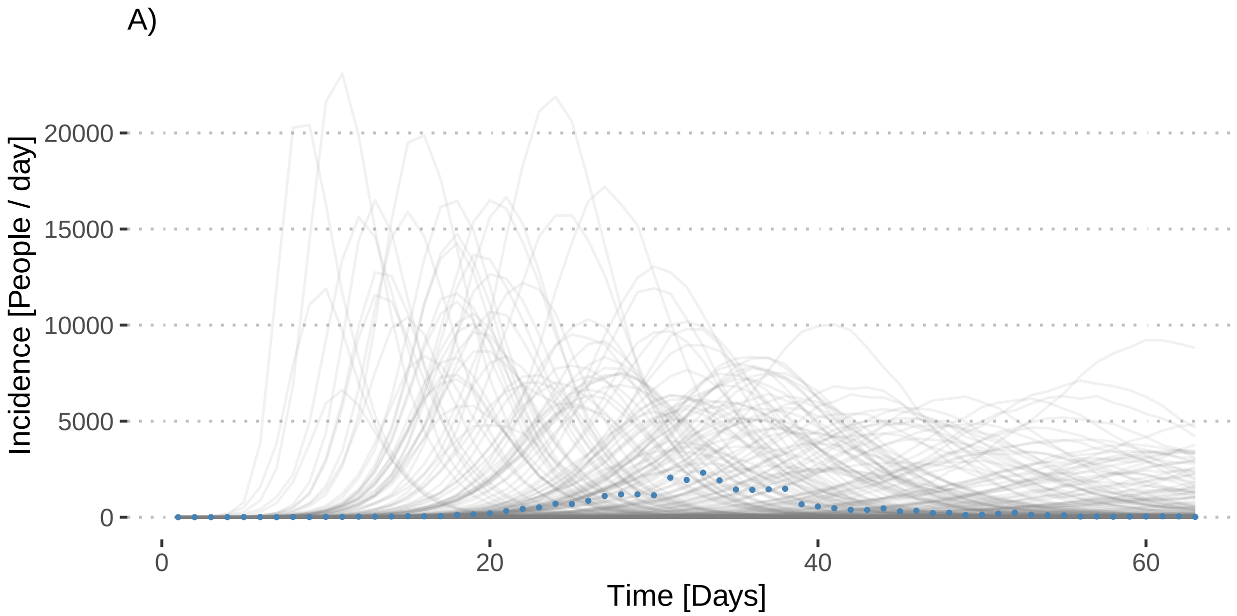
```
##       time             S                  E                   A
##  Min.   : 0.00   Min.   :     5.9   Min.   :     0.00   Min.   :     0.0
##  1st Qu.:15.75   1st Qu.:546028.7   1st Qu.:     0.07   1st Qu.:     0.2
##  Median :31.50   Median :549931.2   Median :     4.67   Median :     7.5
##  Mean   :31.50   Mean   :469079.4   Mean   :  4605.90   Mean   : 17669.4
##  3rd Qu.:47.25   3rd Qu.:549983.5   3rd Qu.:   181.24   3rd Qu.:   653.5
##  Max.   :63.00   Max.   :549999.4   Max.   :292989.00   Max.   :423648.6
##        I                  J                  R                  D
##  Min.   :    0.00   Min.   :     0.00   Min.   :     0.0   Min.   :    0.000
##  1st Qu.:    0.01   1st Qu.:     0.01   1st Qu.:    11.1   1st Qu.:    0.160
##  Median :    0.79   Median :     0.77   Median :    40.4   Median :    0.529
##  Mean   :  813.74   Mean   :  3292.57   Mean   : 53966.7   Mean   :  572.359
##  3rd Qu.:   28.83   3rd Qu.:    51.29   3rd Qu.:  1222.8   3rd Qu.:   11.664
##  Max.   :43965.77   Max.   :159863.95   Max.   :546642.8   Max.   :10047.971
##        C                gamma_1              DR                gamma_2
##  Min.   :     0.10   Min.   :0.005775   Min.   :    0.000   Min.   : 0.00585
##  1st Qu.:     5.40   1st Qu.:0.122716   1st Qu.:    0.005   1st Qu.: 0.16214
##  Median :    15.37   Median :0.243265   Median :    0.380   Median : 0.50080
##  Mean   : 11839.53   Mean   :0.251356   Mean   :  391.707   Mean   : 1.74781
##  3rd Qu.:   362.27   3rd Qu.:0.374964   3rd Qu.:   14.094   3rd Qu.: 1.46937
##  Max.   :194074.97   Max.   :0.667106   Max.   :23725.558   Max.   :46.74746
##       RR2                RR12                RR11                ER
##  Min.   :    0.000   Min.   :    0.000   Min.   :    0.00   Min.   :     0.00
##  1st Qu.:    0.006   1st Qu.:    0.003   1st Qu.:    0.06   1st Qu.:     0.04
##  Median :    0.275   Median :    0.137   Median :    1.38   Median :     2.46
##  Mean   :  300.166   Mean   :  115.106   Mean   : 1569.97   Mean   :  2424.16
##  3rd Qu.:   12.166   3rd Qu.:    3.864   3rd Qu.:   90.61   3rd Qu.:    95.39
##  Max.   :15621.565   Max.   :16214.361   Max.   :67041.21   Max.   :154204.74
##       HR6                HR5                 HR4                 HR3
##  Min.   : 0.000000   Min.   :0.000000   Min.   :0.0000000   Min.   : 0.000000
##  1st Qu.: 0.000506   1st Qu.:0.000000   1st Qu.:0.0000005   1st Qu.: 0.000009
##  Median : 0.001843   Median :0.000035   Median :0.0000362   Median : 0.000343
##  Mean   : 2.464233   Mean   :0.150345   Mean   :0.0371570   Mean   : 0.806821
##  3rd Qu.: 0.055837   3rd Qu.:0.002342   3rd Qu.:0.0013166   3rd Qu.: 0.029842
##  Max.   :24.960859   Max.   :7.299724   Max.   :2.0075696   Max.   :19.344683
##       HR2                 HR1                  N              delta
##  Min.   : 0.000000   Min.   : 0.000269   Min.   :539952   Min.   :0.000423
##  1st Qu.: 0.000003   1st Qu.:24.932817   1st Qu.:549988   1st Qu.:0.011647
##  Median : 0.000213   Median :25.111015   Median :550000   Median :0.035966
##  Mean   : 0.210315   Mean   :21.419149   Mean   :549428   Mean   :0.125516
##  3rd Qu.: 0.008276   3rd Qu.:25.113401   3rd Qu.:550000   3rd Qu.:0.105521
##  Max.   :13.378493   Max.   :25.114126   Max.   :550000   Max.   :3.357002
##        MR                 PR                 AR               lambda
##  Min.   :   0.0000   Min.   :    0.00   Min.   :     0.00   Min.   :0.000000
##  1st Qu.:   0.0004   1st Qu.:    0.01   1st Qu.:     0.03   1st Qu.:0.000000
##  Median :   0.0198   Median :    0.46   Median :     1.97   Median :0.000005
##  Mean   :  21.5661   Mean   :  514.47   Mean   :  1909.69   Mean   :0.038166
##  3rd Qu.:   0.8741   3rd Qu.:   19.48   3rd Qu.:    75.33   3rd Qu.:0.000617
##  Max.   :1121.8093   Max.   :39060.49   Max.   :115144.25   Max.   :4.470358
##        IR                 BR              mu                 beta
##  Min.   :     0.00   Min.   :24.66   Min.   :4.566e-05   Min.   : 0.04298
##  1st Qu.:     0.02   1st Qu.:25.11   1st Qu.:4.566e-05   1st Qu.: 0.73783
##  Median :     2.31   Median :25.11   Median :4.566e-05   Median : 1.56744
##  Mean   :  2463.75   Mean   :25.09   Mean   :4.566e-05   Mean   : 2.52610
```

```
##  3rd Qu.:    90.39   3rd Qu.:25.11   3rd Qu.:4.566e-05   3rd Qu.: 2.79864
##  Max.   :291191.17   Max.   :25.11   Max.   :4.566e-05   Max.   :41.38231
##        q                   k                 rho                omega
##  Min.   :7.811e-05   Min.   :0.5263   Min.   :0.07457   Min.   :0.01303
##  1st Qu.:6.352e-03   1st Qu.:0.5263   1st Qu.:0.16445   1st Qu.:0.25914
##  Median :1.630e-02   Median :0.5263   Median :0.19684   Median :0.49874
##  Mean   :2.522e-02   Mean   :0.5263   Mean   :0.19996   Mean   :0.50033
##  3rd Qu.:3.565e-02   3rd Qu.:0.5263   3rd Qu.:0.23209   3rd Qu.:0.74175
##  Max.   :1.495e-01   Max.   :0.5263   Max.   :0.42402   Max.   :0.98887
##      alpha             cfp              I0          E0          iter
##  Min.   :0.2632   Min.   :0.067   Min.   :0   Min.   :0   Min.   :  1.0
##  1st Qu.:0.4385   1st Qu.:0.067   1st Qu.:0   1st Qu.:0   1st Qu.:125.8
##  Median :0.4918   Median :0.067   Median :0   Median :0   Median :250.5
##  Mean   :0.5067   Mean   :0.067   Mean   :0   Mean   :0   Mean   :250.5
##  3rd Qu.:0.5632   3rd Qu.:0.067   3rd Qu.:0   3rd Qu.:0   3rd Qu.:375.2
##  Max.   :0.8424   Max.   :0.067   Max.   :0   Max.   :0   Max.   :500.0
```

```
file_path <- file.path(fldr, "sens_inc.rds")

if(!file.exists(file_path)) {
  set.seed(346282)
  sens_inc <- predictive_checks(n_sims, sens_o)  # dist = "pois"

  saveRDS(sens_inc, file_path)
} else {
  sens_inc <- readRDS(file_path)
}
```

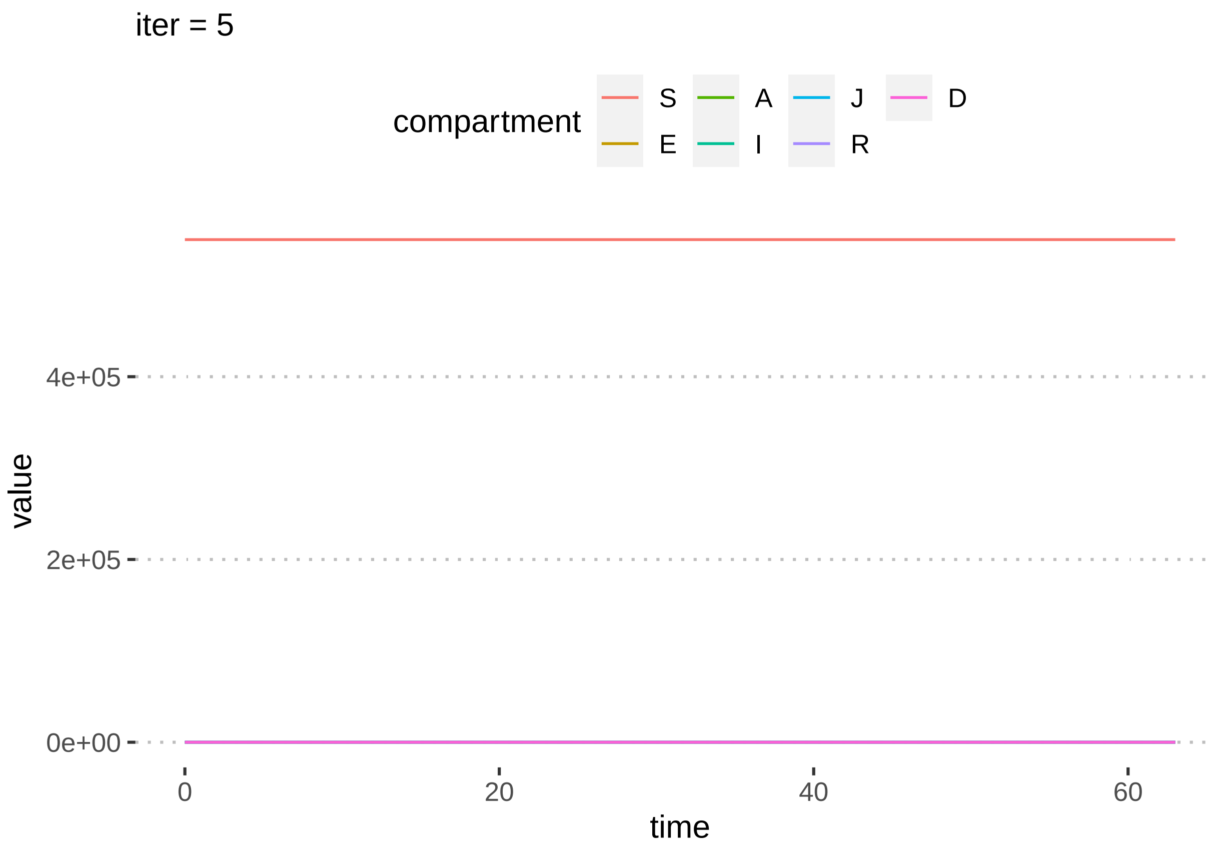
Here, we plot the results of the prior predictive checks. Dots indicate the actual data.

```
g1 <- ggplot(sens_inc, aes(x = time, y = y)) +
  geom_line(aes(group = iter), alpha = 0.1, colour = "grey50") +
  geom_point(data = flu_data, aes(x = time, y = y), size = 0.5,
             colour = "steelblue") +
  theme_pubclean() +
  labs(y        = "Incidence [People / day]",
       x        = "Time [Days]",
       subtitle = "A)")

print(g1)
```



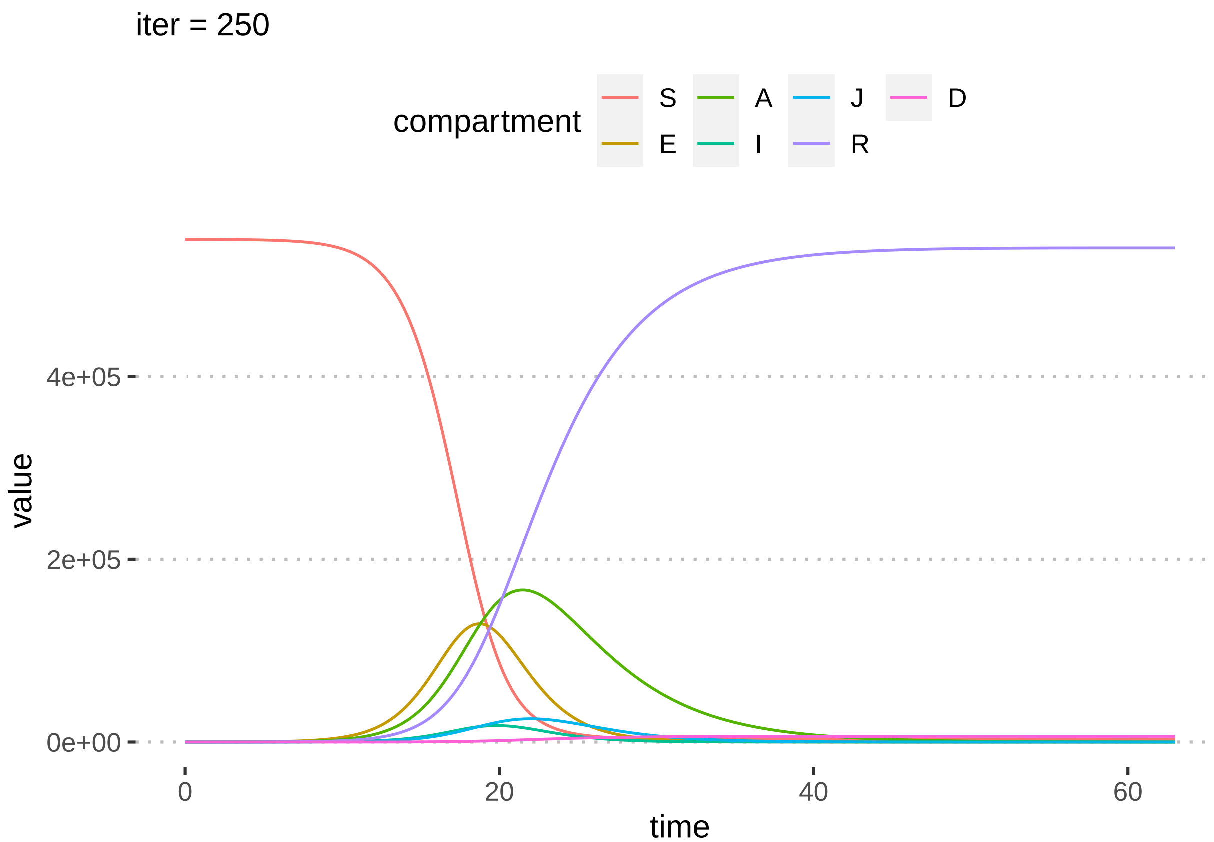
We can see that there are simulations that do not present changes in any of the compartments as shown in the following image:

```
sens_o5 <- filter(sens_o, iter == 5)

plt <- sens_o5[ , c("time", "S", "E", "A", "I", "J", "R", "D")]
plt <- melt(plt ,  id.vars = 'time', variable.name = 'compartment')

ggplot(plt, aes(x = time, y = value)) +
  geom_line(aes(colour = compartment)) +
  theme_pubclean() +
  labs(subtitle = "iter = 5")
```



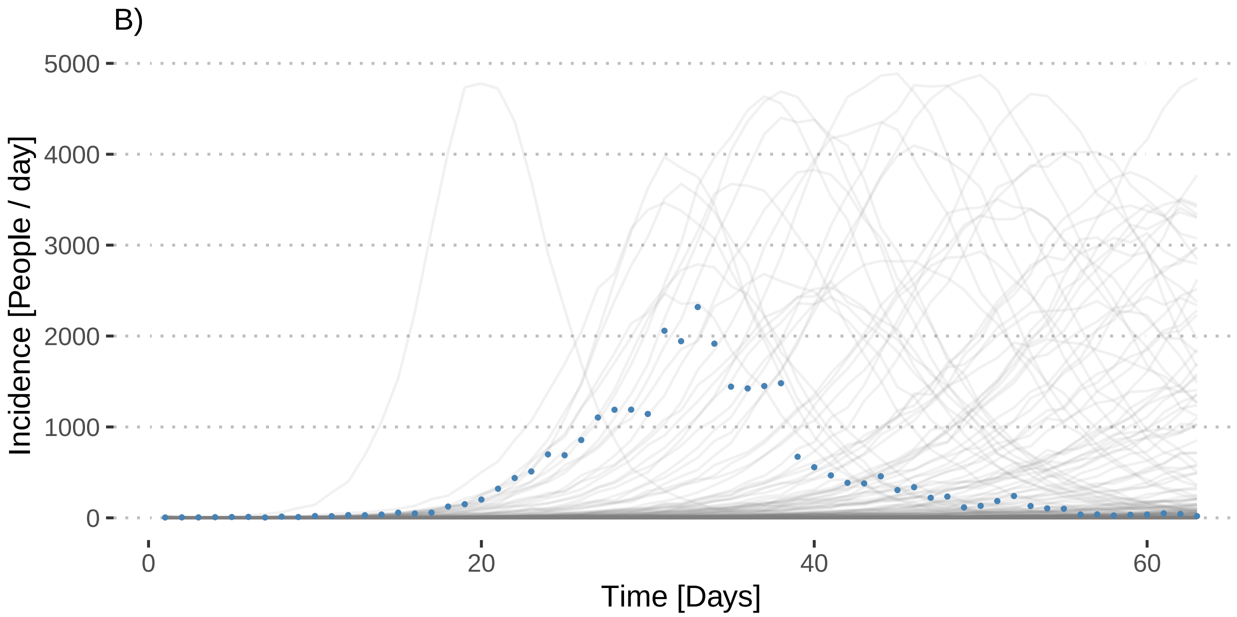
But there are also simulations that do show changes in the compartments. For example, iteration 250:

```
sens_o250 <- filter(sens_o, iter == 250)

plt <- sens_o250[ , c("time", "S", "E", "A", "I", "J", "R", "D")]
plt <- melt(plt ,  id.vars = 'time', variable.name = 'compartment')

ggplot(plt, aes(x = time, y = value)) +
  geom_line(aes(colour = compartment)) +
  theme_pubclean() +
  labs(subtitle = "iter = 250")
```



In the above simulations, it can be seen that the prior distribution produces simulations far off the actual data. To have a closer inspection, we filter out trajectories whose peak is greater than 5000 new cases in a day.

```
max_y_df    <- sens_inc %>% group_by(iter) %>% summarise(max_y = max(y))
filtered_df <- filter(max_y_df, max_y < 5000)
iters       <- filtered_df$iter

sens_inc2 <- filter(sens_inc, iter %in% iters)

g2 <- ggplot(sens_inc2, aes(x = time, y = y)) +
  geom_line(aes(group = iter), alpha = 0.1, colour = "grey50") +
  geom_point(data = flu_data, aes(x = time, y = y), size = 0.5,
             colour = "steelblue") +
  theme_pubclean() +
  labs(y        = "Incidence [People / day]",
       x        = "Time [Days]",
       subtitle = "B)")

print(g2)
```



3.2.1 Measurement model

In this section, we discuss the choice of the measurement model and how we use prior predictive checks to guide such a decision. Initially, we opt for the default choice, i.e., the normal distribution. This choice implies that, at every time step, the difference between the measurement and the true value (error) follows a normal distribution. Additionally, this distribution entails that the error across all time steps is similar (homoscedasticity). In other words, the magnitude of the error is indifferent to the magnitude of the true value, an assumption that may seem unrealistic. As could be expected, the normal distribution adds a new unknown, the standard deviation (δ). To test this model, we assume $\delta \sim Cauchy(0, 1)$, and simulate 500 trajectories.

Next, we consider the Poisson distribution given that it has been used in the empirical treatment of count data, particularly concerning counts of events per unit of time. This distribution lifts the constraint of equal variance across measurements as the error magnitude is proportional to the true number of reported individuals at each time step. As with the normal distribution, we generate 500 simulations.

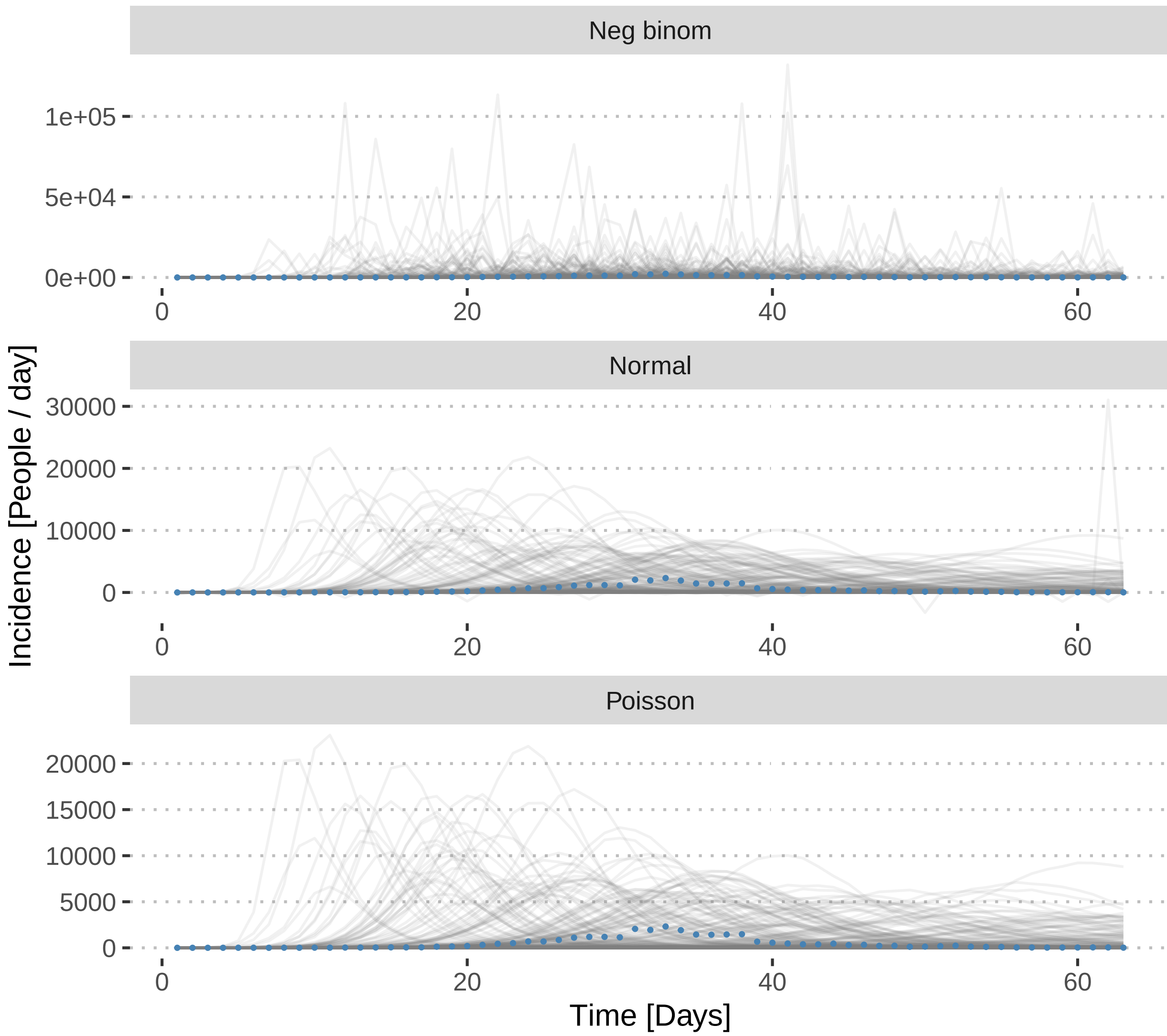
Finally, to count data, the Negative Binomial distribution is an alternative to the Poisson should the latter fail to capture overdispersion in the data. We model such overdispersion in the Negative Binomial's scale parameter ($\phi \sim$ Half-normal$(0, 1)$). We can see that the trajectories generated by the Negative Binomial are more dispersed than the ones produced by the Poisson distribution. Nevertheless, in this case, they do not provide a more accurate representation of the data. Consequently, we adopt the Poisson distribution as the measurement model.

In addition, in the "The reproduction number for influenza" article, the following is stated:

"The stochastic version of the model is formulated as usual by taking the rates on the right-hand side of the population equations to determine the mean change λ over the time τ of the several population classes, which is in practice extracted from a probability distribution $P[\lambda]$ with average λ. In the estimation procedure described below, P is taken to be a Poisson distribution, which is the maximal entropy distribution for a discrete process for which only the average is known. If information is also available about the statistics of fluctuations, a more general distribution, such as a Negative Binomial, can be employed instead."

```
set.seed(102667)

pois_meas <- sens_inc %>% mutate(dist = "Poisson")
```

```r
# Normal distributed measurements
norm_meas   <- predictive_checks(n_sims, sens_o, "norm") %>%
  mutate(dist = "Normal")

# Negative binomial measurements
nbinom_meas <- predictive_checks(n_sims, sens_o, "nbinom") %>%
  mutate(dist = "Neg binom")

meas_df <- bind_rows(pois_meas, norm_meas, nbinom_meas)

n1 <- ggplot(meas_df, aes(x = time, y = y)) +
  geom_line(aes(group = iter), alpha = 0.1, colour = "grey50") +
  geom_point(data = flu_data, aes(x = time, y = y), size = 0.5,
             colour = "steelblue") +
  facet_wrap(~dist, ncol = 1, scales = "free") +
  theme_pubclean() +
  labs(y        = "Incidence [People / day]",
       x        = "Time [Days]",
       subtitle = "")

print(n1)
```



3.3 Fitting

For calibrating SD models in Stan, it is necessary to create a file written in Stan's own language. This kind of file structures the code in blocks. For this example, five blocks are necessary: *Functions*, *data*, *parameters*, *transformed parameters*, and *model*. Here model refers to prior distributions and the likelihood (measurement model). The SD model is considered a function, which can be constructed from the model specification. This translation is possible thanks to the function *stan_ode_function_2* (a modification of the *stan_ode_function* from the *readsdr* package). Similarly, *readsdr* supports the creation of the data block. The other blocks must be built manually.

```
pars_hat  <- c("I0", "E0", "beta", "q", "rho", "omega", "alpha")  # For next steps

consts    <- sd_constants(model)
ODE_fn    <- "SEIR"
stan_fun  <- stan_ode_function_2(model_structure,
                                 ODE_fn,
                                 pars = consts$name[c(2, 3, 5:7)])  # "beta" "q" "rho" "omega" "alpha"

fun_exe_line <- str_glue("  o = ode_rk45({ODE_fn}, x0, t0, ts, params);")
```

```
stan_data <- stan_data("y", type = "int", inits = FALSE)

stan_params <- paste(
  "parameters {",
  "  real<lower = 0>            beta;",
  "  real<lower = 0, upper = 1> q;",
  "  real<lower = 0, upper = 1> rho;",
  "  real<lower = 0, upper = 1> omega;",
  "  real<lower = 0, upper = 1> alpha;",
  "  real<lower = 0, upper = 55000> I0;",
  "  real<lower = 0, upper = 55000> E0;",
"}", sep = "\n")

stan_tp <- paste(
  "transformed parameters{",
  "  vector[n_difeq] o[n_obs]; // Output from the ODE solver",
  "  real x[n_obs];",
  "  vector[n_difeq] x0;",
  "  real params[n_params];",
  "  x0[1] = 550000 - E0 - I0;",
  "  x0[2] = E0;",
  "  x0[3] = 0;",
  "  x0[4] = I0;",
  "  x0[5] = 0;",
  "  x0[6] = 0;",
  "  x0[7] = 0;",
  "  x0[8] = I0;",
  "  params[1] = beta;",
  "  params[2] = q;",
  "  params[3] = rho;",
  "  params[4] = omega;",
  "  params[5] = alpha;",
  fun_exe_line,
  "  x[1] =  o[1, 8]  - x0[8];",
  "  for (i in 1:n_obs-1) {",
  "    x[i + 1] = o[i + 1, 8] - o[i, 8] + 1e-5;",
  "  }",
  "}", sep = "\n")

stan_model <- paste(
  "model {",
  "  omega   ~ uniform(0.01, 0.99);",
  "  beta    ~ lognormal(0.5, 1);",
  "  q       ~ beta(0.75, 30);",
  "  rho     ~ beta(10, 40);",
  "  alpha   ~ lognormal(-0.7, 0.2);",
  "  I0      ~ lognormal(1, 1);",
  "  E0      ~ lognormal(1, 1);",
  "  y       ~ poisson(x);",
  "}",
  sep = "\n")

stan_gc <- paste(
```

```
  "generated quantities {",
  "  real log_lik;",
  "  log_lik = poisson_lpmf(y | x);",
  "}",
  sep = "\n")

stan_text   <- paste(stan_fun, stan_data, stan_params,
                       stan_tp, stan_model, stan_gc, sep = "\n")

stan_fldr     <- "./Stan_files/san_francisco_ext_202010"
dir.create(stan_fldr, showWarnings = FALSE, recursive = TRUE)
stan_filepath <- file.path(stan_fldr, "flu_poisson_new.stan")

create_stan_file(stan_text, stan_filepath)
```

We show below the code contained in the Stan file.

```
cat(stan_text)
```

```
## functions {
##   vector SEIR(real time, vector y, real[] params) {
##     vector[8] dydt;
##     real gamma_1;
##     real DR;
##     real gamma_2;
##     real RR2;
##     real RR12;
##     real RR11;
##     real ER;
##     real HR6;
##     real HR5;
##     real HR4;
##     real HR3;
##     real HR2;
##     real HR1;
##     real N;
##     real delta;
##     real MR;
##     real PR;
##     real AR;
##     real lambda;
##     real IR;
##     real BR;
##     gamma_1 = params[4]*params[5];
##     DR = y[4]*params[5];
##     gamma_2 = 1/((1/gamma_1)-(1/params[5]));
##     RR2 = y[5]*gamma_2;
##     RR12 = y[4]*gamma_1;
##     RR11 = y[3]*gamma_1;
##     ER = y[2]*0.5263157895;
##     HR6 = 4.56621e-05*y[6];
##     HR5 = 4.56621e-05*y[5];
##     HR4 = 4.56621e-05*y[4];
##     HR3 = 4.56621e-05*y[3];
```

```
##     HR2 = 4.56621e-05*y[2];
##     HR1 = 4.56621e-05*y[1];
##     N = y[1]+y[2]+y[4]+y[3]+y[5]+y[6];
##     delta = (0.067/(1-0.067))*(4.56621e-05+gamma_2);
##     MR = y[5]*delta;
##     PR = params[3]*ER;
##     AR = (1-params[3])*ER;
##     lambda = ((y[4]+y[5]+params[2]*y[3])*params[1])/N;
##     IR = lambda*y[1];
##     BR = 4.56621e-05*N;
##     dydt[1] = BR-IR-HR1;
##     dydt[2] = IR-ER-HR2;
##     dydt[3] = AR-RR11-HR3;
##     dydt[4] = PR-DR-RR12-HR4;
##     dydt[5] = DR-RR2-MR-HR5;
##     dydt[6] = RR11+RR12+RR2-HR6;
##     dydt[7] = MR;
##     dydt[8] = DR;
##     return dydt;
##   }
## }
## data {
##   int<lower = 1> n_obs;
##   int<lower = 1> n_params;
##   int<lower = 1> n_difeq;
##   int y[n_obs];
##   real t0;
##   real ts[n_obs];
## }
## parameters {
##   real<lower = 0>            beta;
##   real<lower = 0, upper = 1> q;
##   real<lower = 0, upper = 1> rho;
##   real<lower = 0, upper = 1> omega;
##   real<lower = 0, upper = 1> alpha;
##   real<lower = 0, upper = 55000> I0;
##   real<lower = 0, upper = 55000> E0;
## }
## transformed parameters{
##   vector[n_difeq] o[n_obs]; // Output from the ODE solver
##   real x[n_obs];
##   vector[n_difeq] x0;
##   real params[n_params];
##   x0[1] = 550000 - E0 - I0;
##   x0[2] = E0;
##   x0[3] = 0;
##   x0[4] = I0;
##   x0[5] = 0;
##   x0[6] = 0;
##   x0[7] = 0;
##   x0[8] = I0;
##   params[1] = beta;
##   params[2] = q;
##   params[3] = rho;
```

```
##   params[4] = omega;
##   params[5] = alpha;
##   o = ode_rk45(SEIR, x0, t0, ts, params);
##   x[1] =  o[1, 8]  - x0[8];
##   for (i in 1:n_obs-1) {
##     x[i + 1] = o[i + 1, 8] - o[i, 8] + 1e-5;
##   }
## }
## model {
##   omega   ~ uniform(0.01, 0.99);
##   beta    ~ lognormal(0.5, 1);
##   q       ~ beta(0.75, 30);
##   rho     ~ beta(10, 40);
##   alpha   ~ lognormal(-0.7, 0.2);
##   I0      ~ lognormal(1, 1);
##   E0      ~ lognormal(1, 1);
##   y       ~ poisson(x);
## }
## generated quantities {
##   real log_lik;
##   log_lik = poisson_lpmf(y | x);
## }
```

To perform the calibration via HMC, we must provide Stan with the calibration parameters. We specify 2,000 iterations (1,000 for warming-up and 1,000 for sampling) and four chains. We also supply the number of parameters to be fitted for the SD model, the number of stocks (*n_difeq*), the simulation time, and San Francisco's data.

```
# Path to cmdstan
set_cmdstan_path("/Users/redondo/cmdstan")

fldr      <- "./backup_objs/san_francisco_ext_202210"
file_path <- file.path(fldr, "fit.rds")

if(!file.exists(file_path)) {
  stan_d <- list(n_obs    = nrow(flu_data),
                 y        = flu_data$y,
                 n_params = 5,
                 n_difeq  = 8,
                 t0       = 0,
                 ts       = 1:length(flu_data$y))

mod <- cmdstan_model(stan_filepath)

fit <- mod$sample(data            = stan_d,
                  seed            = 553616,
                  chains          = 4,
                  parallel_chains = 4,
                  iter_warmup     = 1500,
                  iter_sampling   = 500,
                  refresh         = 5,
                  save_warmup     = TRUE,
                  output_dir      = fldr)
                  # adapt_delta     = 0.99
                  # step_size       = 0.1
```

```r
fit$save_object(file_path)
} else {
  fit <- readRDS(file_path)
}
```

We note that the execution ends with the following information:

\texttt{ All 4 chains finished successfully.

Mean chain execution time: 5432.4 seconds.

Total execution time: 5758.2 seconds. }

Although the model is complex, execution has taken too long.

3.3.1 Diagnostics

Before inspecting the samples, Stan returns global diagnostics to the user about the sampling process. It is expected that the result is free from divergent iterations or indications of pathological behaviour from the Bayesian Fraction of Missing Information metric. Also, iterations that saturate the maximum tree depth indicate a complex posterior surface. The Stan manual provides intuitive interpretations of these metrics.

```r
# We generated this file from $fit$cmdstan_diagnose()
fileName <- file.path(fldr, "diags.txt")

if(!file.exists(fileName)) {
  diagnosis <- fit$cmdstan_diagnose()
  writeLines(diagnosis$stdout, fileName)
} else {
  readChar(fileName, file.info(fileName)$size) %>% cat()
}
```

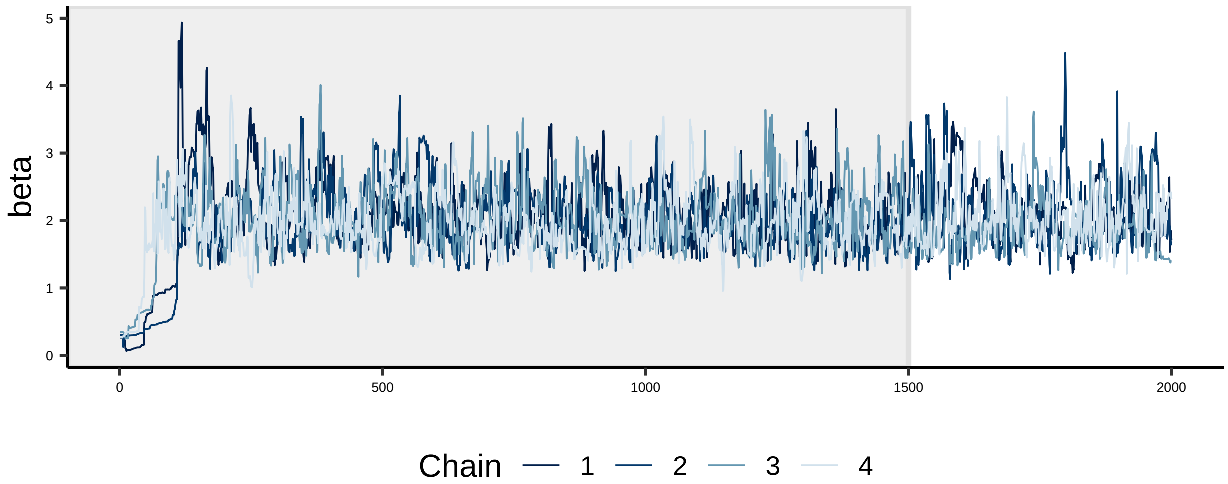
```
## Processing csv files: /Users/redondo/R/SDR_Bayes-master/backup_objs/san_francisco_ext_202209/flu_pois
## , /Users/redondo/R/SDR_Bayes-master/backup_objs/san_francisco_ext_202209/flu_poisson_new-202210140853
## , /Users/redondo/R/SDR_Bayes-master/backup_objs/san_francisco_ext_202209/flu_poisson_new-202210140853
## , /Users/redondo/R/SDR_Bayes-master/backup_objs/san_francisco_ext_202209/flu_poisson_new-202210140853
##
##
## Checking sampler transitions treedepth.
## 4 of 8000 (0.05%) transitions hit the maximum treedepth limit of 10, or 2^10 leapfrog steps.
## Trajectories that are prematurely terminated due to this limit will result in slow exploration.
## For optimal performance, increase this limit.
##
## Checking sampler transitions for divergences.
## 1107 of 8000 (13.84%) transitions ended with a divergence.
## These divergent transitions indicate that HMC is not fully able to explore the posterior distributio
## Try increasing adapt delta closer to 1.
## If this doesn't remove all divergences, try to reparameterize the model.
##
## Checking E-BFMI - sampler transitions HMC potential energy.
## E-BFMI satisfactory.
##
## Effective sample size satisfactory.
##
## Split R-hat values satisfactory all parameters.
##
## Processing complete.
```

As we can see, the diagnostics indicate that we must change either the priors, the model or some of its parameters if we want to achieve satisfactory results.

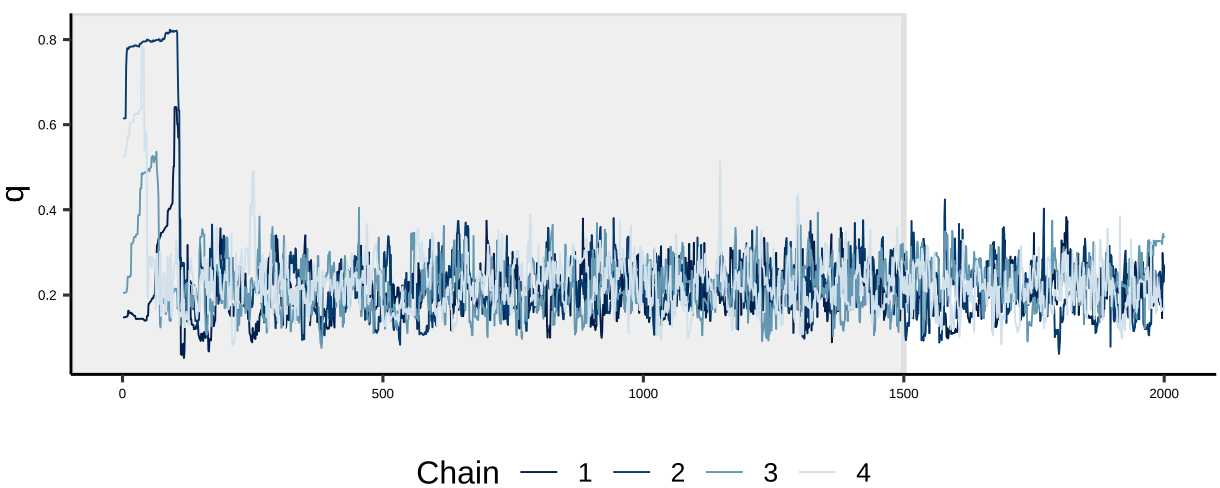
3.3.1.1 Trace plots

A common approach to inspect calibration results is to check for convergence in trace plots.

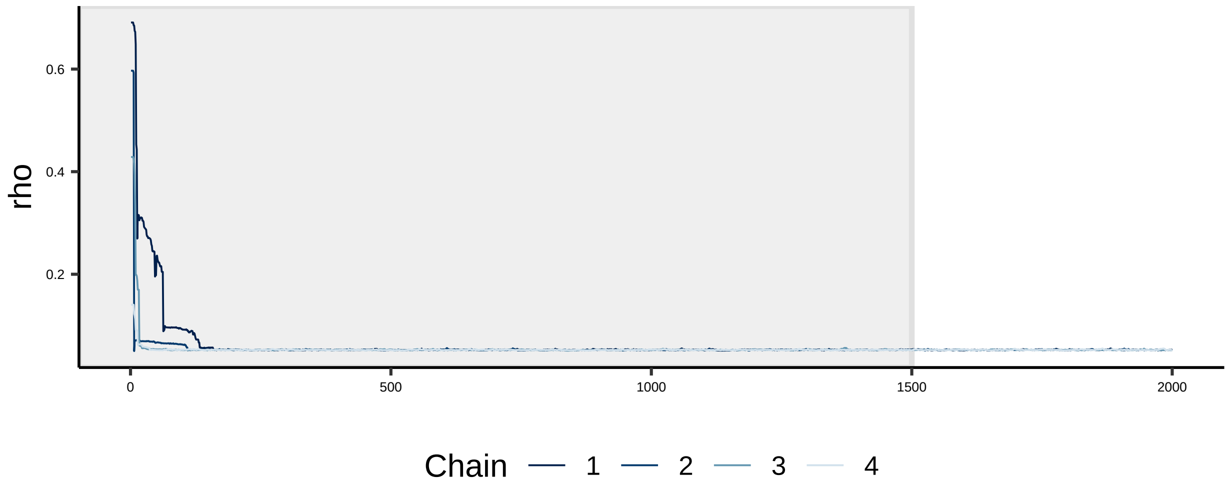
```
b1 <- trace_plot_2(fit, pars = c("beta"), n_samples = 2000, n_warmup = 1500) +
  theme(axis.text = element_text(size = 5))

print(b1)
```



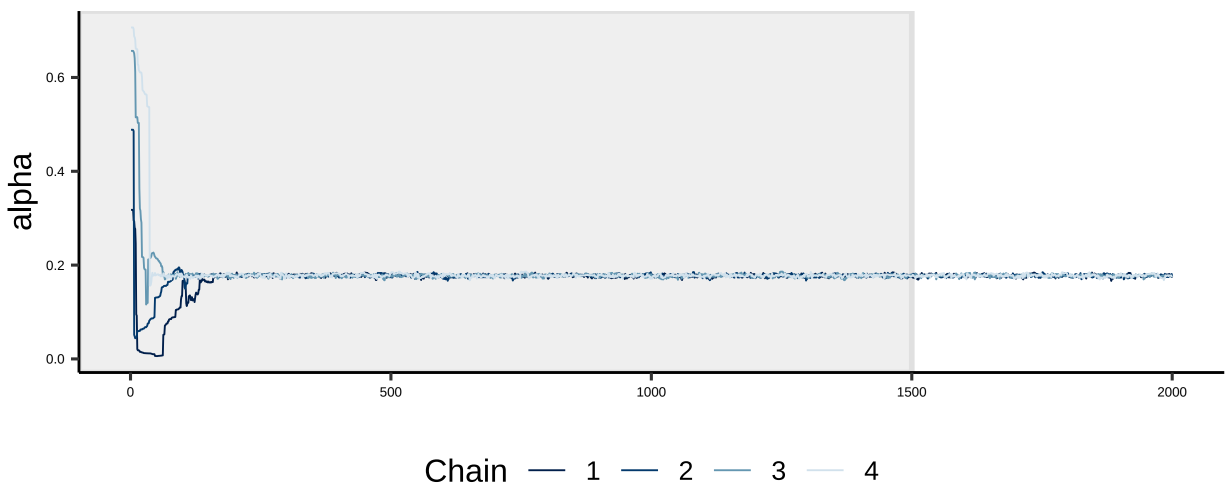

```
b2 <- trace_plot_2(fit, pars = c("q"), n_samples = 2000, n_warmup = 1500) +
  theme(axis.text = element_text(size = 5))

print(b2)
```



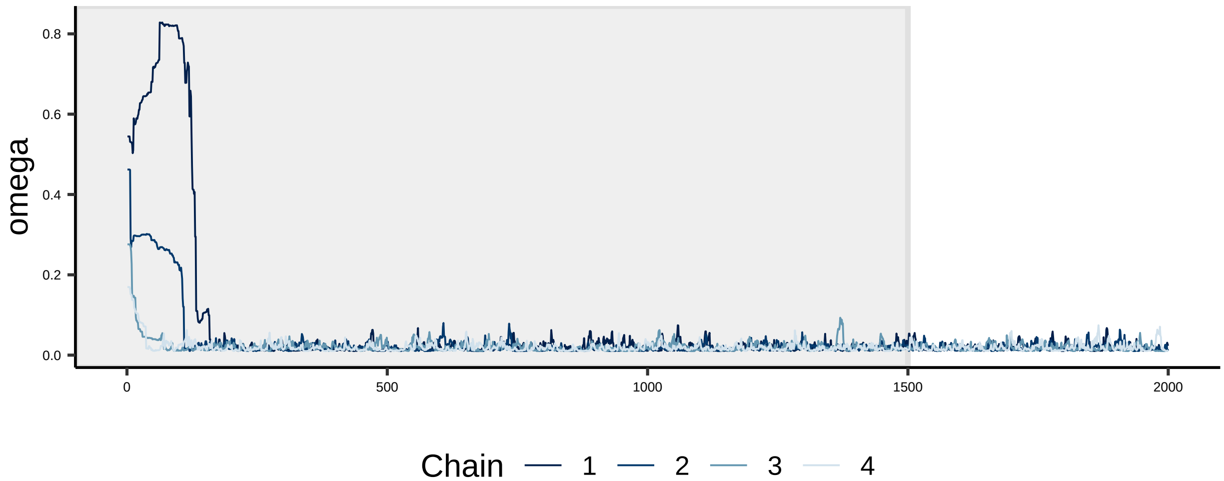

```
b3 <- trace_plot_2(fit, pars = c("rho"), n_samples = 2000, n_warmup = 1500) +
  theme(axis.text = element_text(size = 5))

print(b3)
```



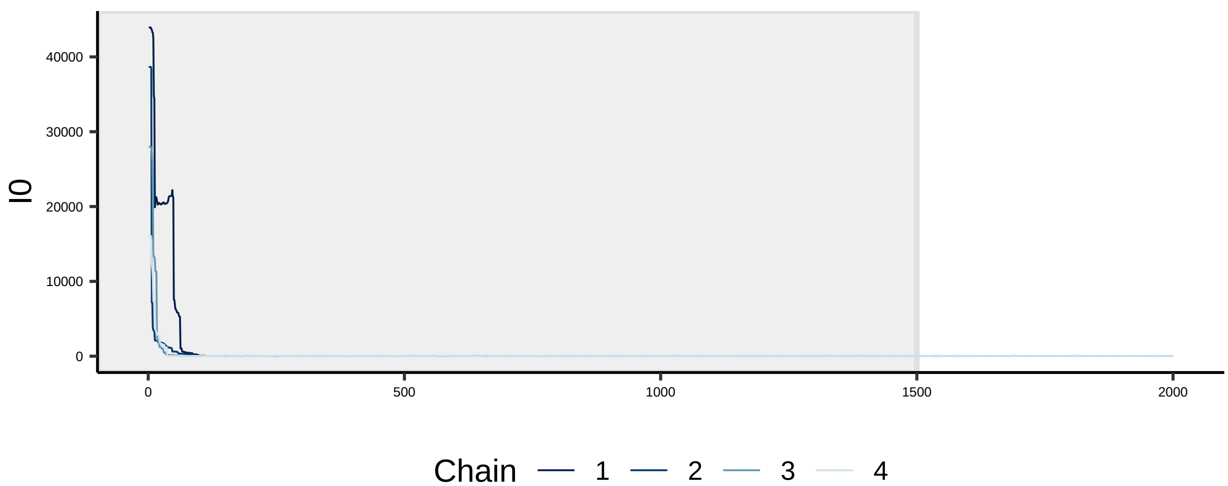

```
b5 <- trace_plot_2(fit, pars = c("alpha"), n_samples = 2000, n_warmup = 1500) +
  theme(axis.text = element_text(size = 5))

print(b5)
```




```
b4 <- trace_plot_2(fit, pars = c("omega"), n_samples = 2000, n_warmup = 1500) +
  theme(axis.text = element_text(size = 5))

print(b4)
```

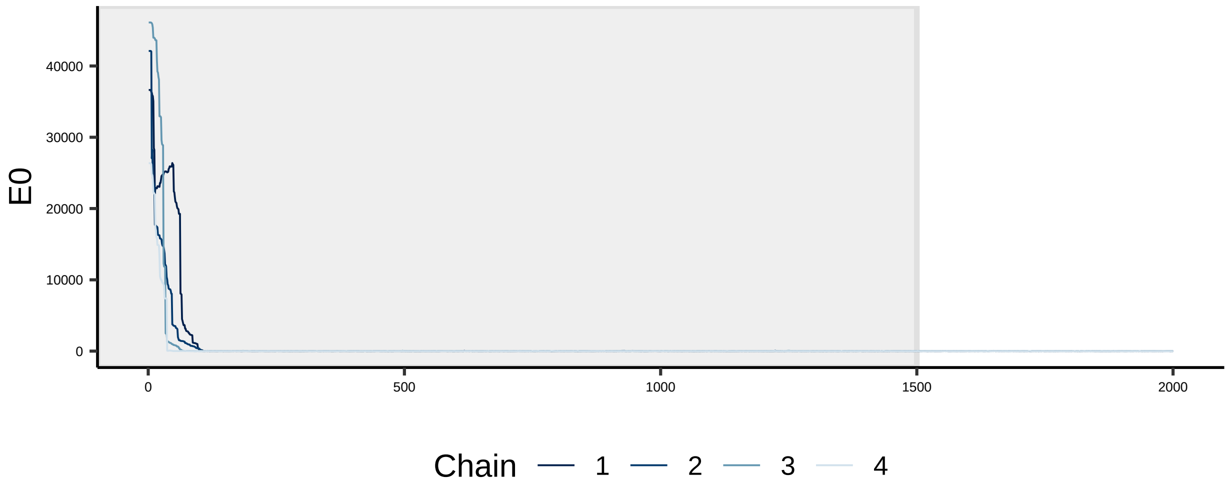



```r
b6 <- trace_plot_2(fit, pars = c("I0"), n_samples = 2000, n_warmup = 1500) +
  theme(axis.text = element_text(size = 5))

print(b6)
```




```r
# posterior  <- fit$draws(inc_warmup = TRUE) %>% as_draws_array()
# posterior  <- posterior[1:2000, ,]
# posterior[ , , variable = 'I0']
```

```r
b7 <- trace_plot_2(fit, pars = c("E0"), n_samples = 2000, n_warmup = 1500) +
  theme(axis.text = element_text(size = 5))

print(b7)
```



We can see...

3.3.1.2 Potential scale reduction factor ($\widehat{R}$) & Effective Sample Size ($\hat{n}_{eff}$)

$\widehat{R}$ is a convergence diagnostic, which compares the between- and within-chain estimates for model parameters and other univariate quantities of interest. If chains have not mixed well, R-hat is larger than 1. It is recommended to run at least *four chains* by default and only using the sample if R-hat is less than 1.01 [@Vehtari_2021]. **Stan reports R-hat, which is the maximum of rank normalized split-R-hat and rank normalized folded-split-R-hat, which works for thick-tailed distributions and is sensitive also to differences in scale**.

For each parameter θ, we split each chain from the *sampling phase* in two halves. That is, from **four** chains of 1000 draws each one, we obtain **eight** split chains of 500 draws each one. Then, we label the simulations as $\theta_{ij}(i = 1, ..., N; j = 1, ..., M)$, where N is the number of samples per split chain, M is the number of split chains, and $S = NM$ is the total number of draws from all chains. We subsequently transform these simulations to their corresponding rank normalized values z_{ij}. According to @Vehtari_2021, we replace each value θ_{ij} by its rank r_{ij} within the pooled draws from all chains. Second, we transform ranks to normal scores using the inverse normal transformation and a fractional offset via Equation 1:

$$z_{ij} = \Phi^{-1}\left(\frac{r_{ij} - 3/8}{S - 1/4}\right) \tag{1}$$

Using these normal scores, we calculate $\widehat{R}$ following the formulation proposed by @gelman2013bayesian. Initially, we compute B and W , the between- and within-sequence variances, respectively:

$$B = \frac{N}{M-1}\sum_{j=1}^{M}(\bar{z}_{.j} - \bar{z}_{..})^2,\ where\ \bar{z}_{.j} = \frac{1}{n}\sum_{i=1}^{n} z_{ij},\ \bar{z}_{..} = \frac{1}{M}\sum_{j=1}^{M}\bar{z}_{.j} \tag{2}$$

$$W = \frac{1}{M}\sum_{j=1}^{M} s_j^2,\ where\ s_j^2 = \frac{1}{N-1}\sum_{i=1}^{n}(z_{ij} - \bar{z}_{.j})^2 \tag{3}$$

Then, we can estimate $\widehat{var}^+(\theta|y)$, the marginal posterior variance of the parameter, by a weighted average of W and B:

$$\widehat{var}^+(\theta|y) = \frac{N-1}{N}W + \frac{1}{N}B \tag{4}$$

From (3) and (4), we obtain the rank normalized split $\widehat{R}$:

$$\widehat{R} = \sqrt{\frac{\widehat{var}^+(\theta|y)}{W}} \tag{5}$$

To obtain the rank normalized folded-split $\widehat{R}$, we simply transform the simulations (Equation 6) and then apply the procedure described above (Equations 1-5).

$$\zeta_{ij} = |\theta_{ij} - median(\theta)| \tag{6}$$

For MCMC draws, we define the estimated effective sample size as

$$\hat{n}_{eff} = \frac{MN}{1 + 2\sum_{t=1}^{T} \hat{\rho}_t} \tag{7}$$

This quantity requires an estimate of the sum of the correlations ρ up to lag T (the first odd positive integer for which $\hat{\rho}_{T+1} + \hat{\rho}_{T+2}$ is negative). The correlation at any specific lag t (Equation 8) depends upon the estimate $\widehat{var}^+$ and the *Variogram* at each t (Equation 9).

$$\hat{\rho}_t = 1 - \frac{V_t}{2\widehat{var}^+} \tag{8}$$

$$V_t = \frac{1}{M(N-t)} \sum_{j=1}^{M} \sum_{i=t+1}^{N} (z_{i,j} - z_{i-t,j})^2 \tag{9}$$

We use the term *bulk effective sample size* to refer to the effective sample size based on the rank normalized draws. To ensure reliable estimates of variances and autocorrelations needed for $\hat{R}$ and $\hat{n}_{eff}$, @Vehtari_2021 recommend that the rank-normalized effective sample size must be greater than 400 (100 per chain).

```
unk_pars <- c("beta", "q", "rho", "alpha", "omega", "I0", "E0")

par_matrices <- lapply(unk_pars, function(par) {
  extract_variable_matrix(fit$draws(), par)
})

ess_vals   <- map_dbl(par_matrices, ess_bulk)
r_hat_vals <- map_dbl(par_matrices, rhat)

smy_df  <- data.frame(par      = c("beta", "q", "rho", "gamma_1", "alpha", "I(0)", "E(0)"),
                      ess_bulk = ess_vals,
                      rhat     = r_hat_vals)
```

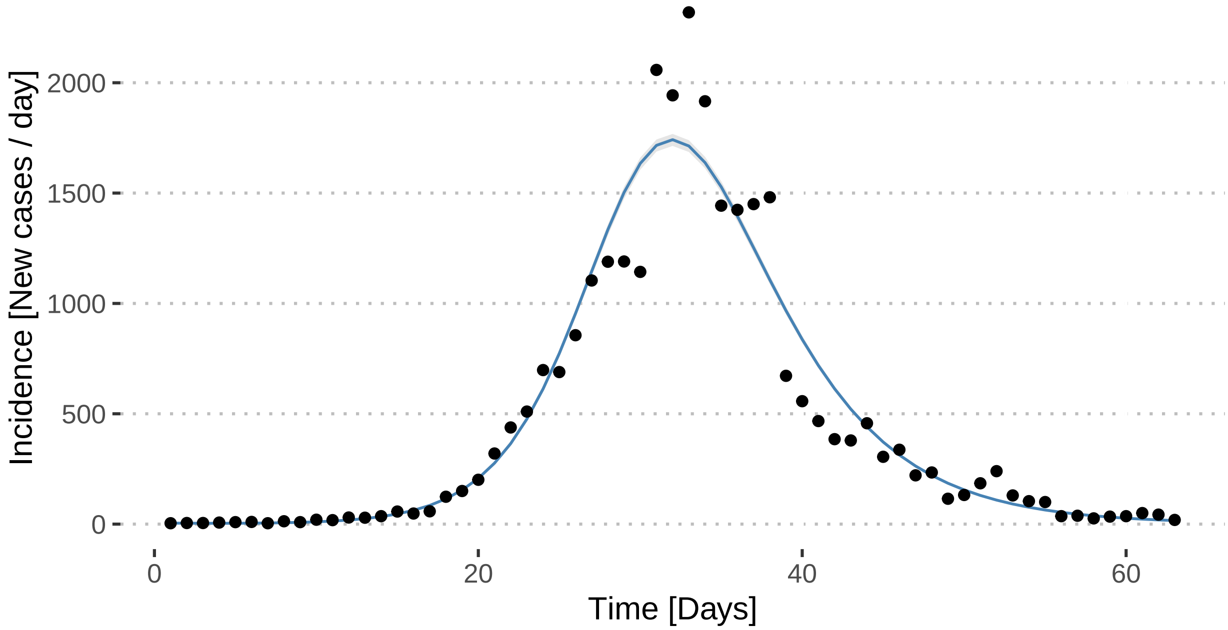
par	ess_bulk	rhat
beta	189.5047	1.033044
q	191.6735	1.033221
rho	299.5844	1.006239
gamma_1	398.9158	1.004438
alpha	263.1130	1.004162
I(0)	211.4549	1.026631
E(0)	516.4998	1.007880

As we can see, neither the values of the effective sample size nor the values of R meet the desired minimums to be able to use the estimations.

3.3.1.3 Check deterministic fit

Once the computation has been deemed satisfactory, we subsequently check that the model's expected value captures the underlying trajectory of the data. The reader should take into account that in each sampling iteration, Stan generates a trajectory from the SD model. We compare these simulated time series against the measured incidence (dots) to visually examine whether the trend is consistent with the data. The solid blue line denotes the mean trajectory, and the grey shaded area indicates the 95 % credible interval.

```r
posterior_df <- fit$draws() %>% as_draws_df()

# We add the fixed constants
samples_normal <- posterior_df[ , pars_hat] %>%
  mutate(mu = 1/(60*365), k = 1/1.9, cfp = 0.067) %>%
  mutate(
    gamma_1 = omega*alpha,
    gamma_2 = 1/((1/gamma_1)-(1/alpha)),
    delta = (cfp/(1-cfp))*(mu+gamma_2),
    R0 = (beta*k)/(k+mu) * (rho*((1/(gamma_1+alpha+mu))+(alpha/((gamma_1+alpha+mu)*(gamma_2+delta+mu)))
    O = alpha / (alpha+gamma_1+mu),
    ll = "Normal"
         )

y_hat_df_norm <- extract_timeseries_var("x", posterior_df)

summary_df <- y_hat_df_norm  %>% group_by(time) %>%
  summarise(lb = quantile(value, c(0.025, 0.975)[[1]]),
            ub = quantile(value, c(0.025, 0.975)[[2]]),
            y  = mean(value))

ggplot(summary_df, aes(x = time , y)) +
  geom_ribbon(aes(ymin = lb, ymax = ub), fill = "grey90") +
  geom_line(colour = "steelblue") +
  geom_point(data = flu_data) +
  theme_pubclean() +
  labs(y = "Incidence [New cases / day]", x = "Time [Days]")
```



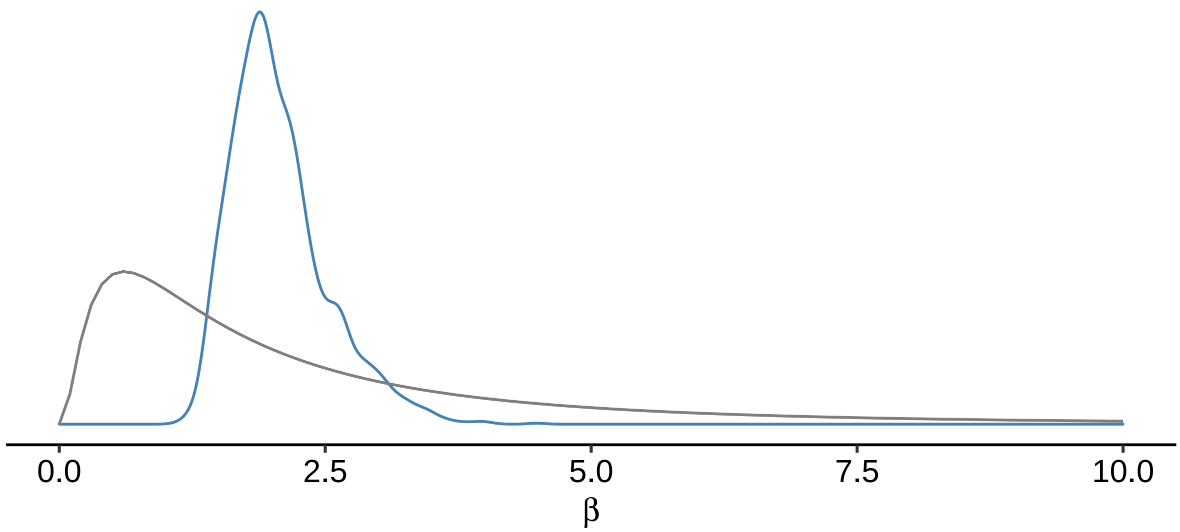
3.3.2 Posterior information

3.3.2.1 Prior & posterior comparison

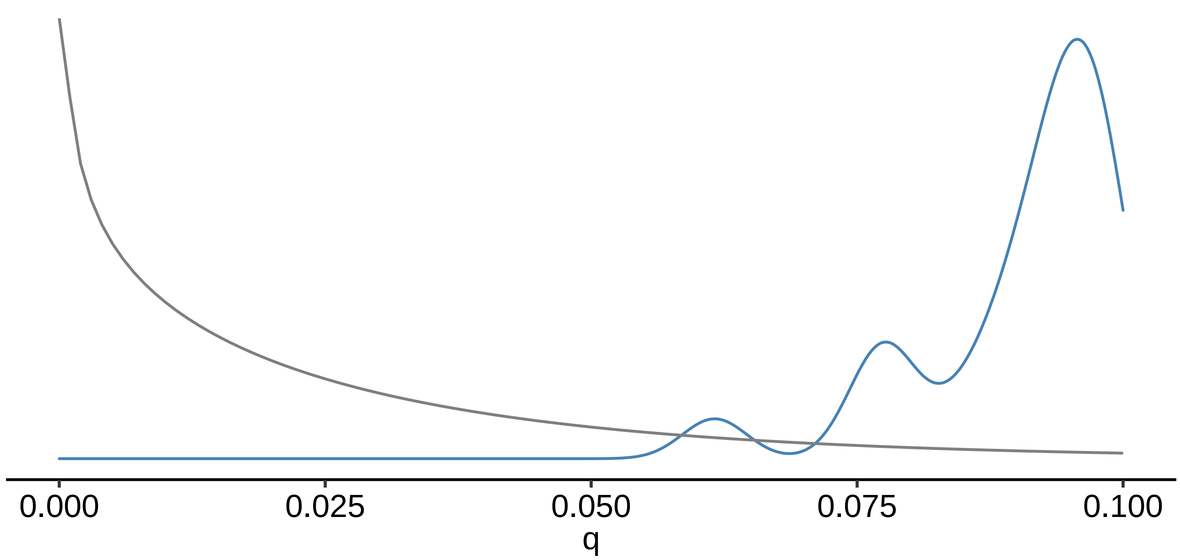
A critical goal of model calibration is to gain information from the data about the parameters of interest. One way to accomplish such a purpose is through the comparison of marginal prior and marginal posterior distributions. Namely, we evaluate the knowledge acquired from the process.

```
pars_df <- posterior_df[, c("beta", "q", "rho", "omega", "alpha", "I0", "E0")]
```

```
beta_df <- dplyr::select(pars_df, beta)

base <- ggplot(beta_df, aes(x = beta)) +
  geom_density(colour = "steelblue") +
  scale_x_continuous(limits = c(0, 10))

g1 <- base + geom_function(fun = dlnorm, args = list(meanlog = 0.5, sdlog = 1),
                           colour = "grey50") +
  theme_pubr() +
  labs(x = bquote(beta), y = "") +
  theme(axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.line.y = element_blank())

g1
```

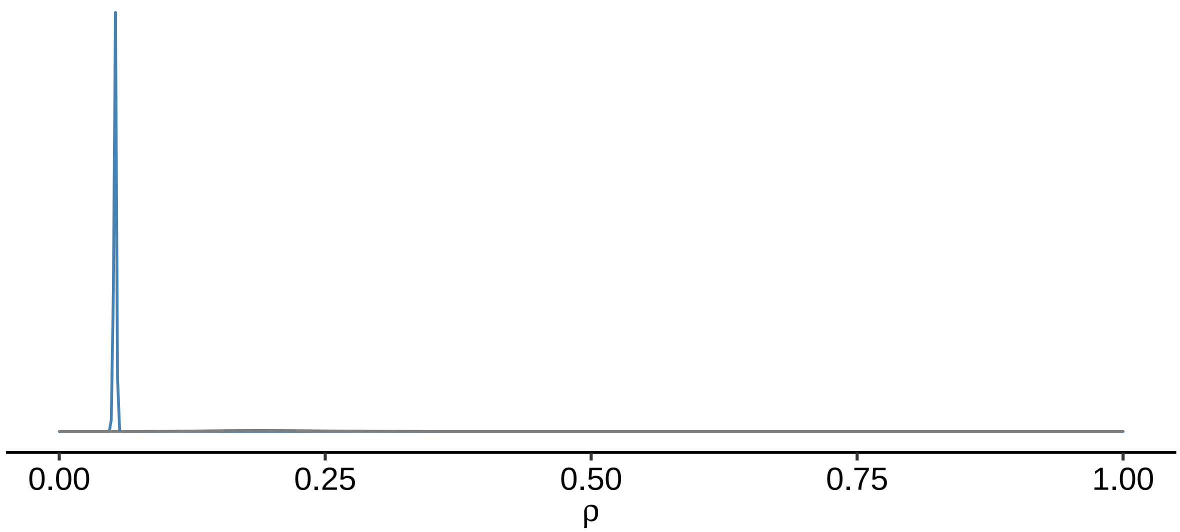



```
q_df <- dplyr::select(pars_df, q)

base <- ggplot(q_df, aes(x = q)) +
  geom_density(colour = "steelblue") +
  scale_x_continuous(limits = c(0, 0.1))

g2 <- base + geom_function(fun = dbeta, args = list(shape1 = 0.75, shape = 30),
                           colour = "grey50") +
  theme_pubr() +
  labs(x = bquote(q), y = "") +
  theme(axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.line.y = element_blank())

g2
```

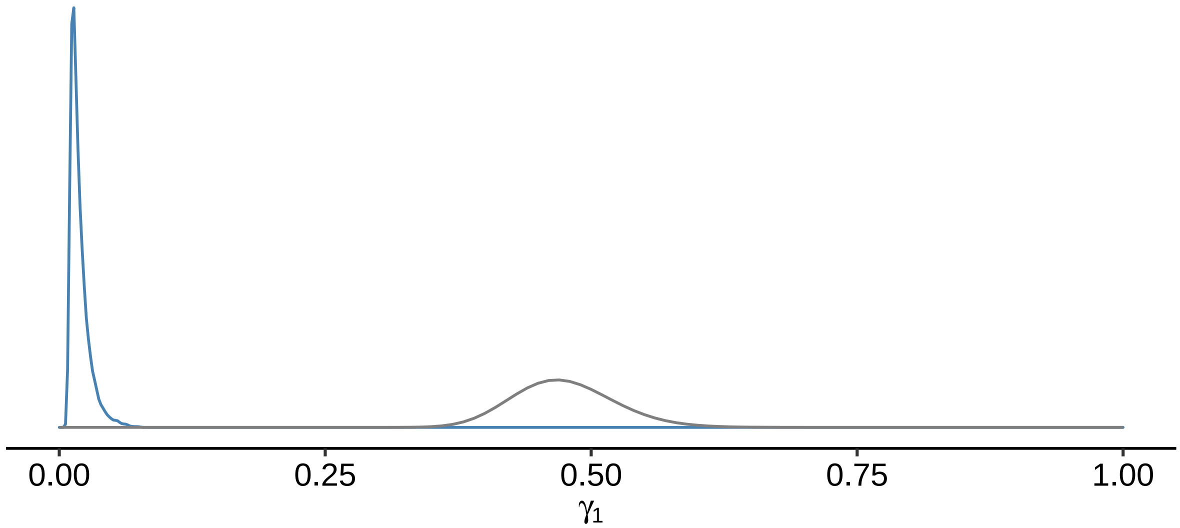



```
rho_df <- dplyr::select(pars_df, rho)

base <- ggplot(rho_df, aes(x = rho)) +
```

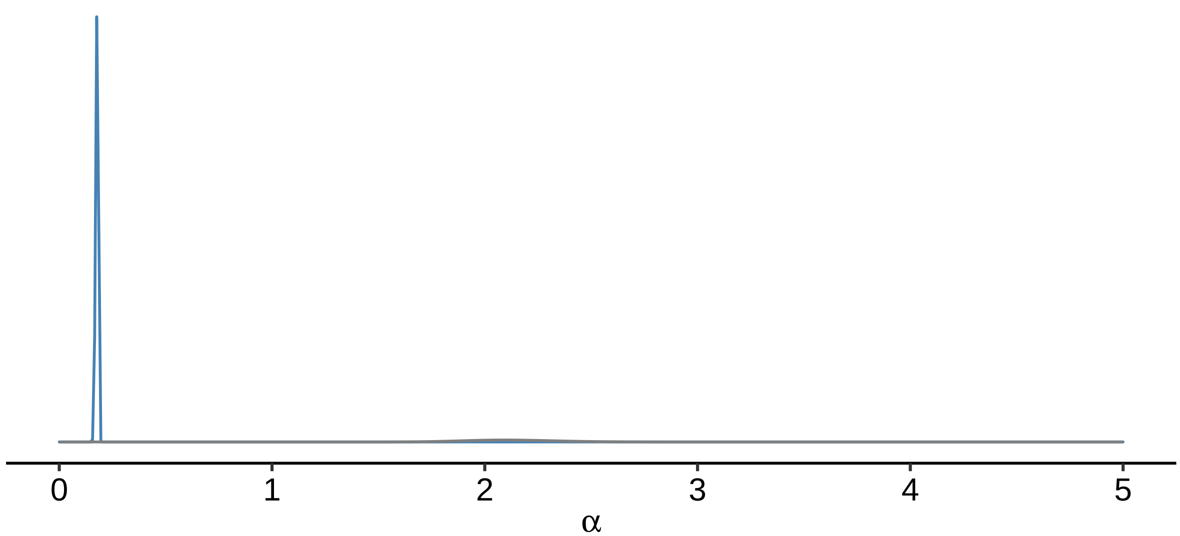
```
  geom_density(colour = "steelblue") +
  scale_x_continuous(limits = c(0, 1))

g3 <- base + geom_function(fun = dbeta, args = list(shape1 = 10, shape = 40),
                           colour = "grey50") +
  theme_pubr() +
  labs(x = bquote(rho), y = "") +
  theme(axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.line.y = element_blank())

g3
```




```
omega_df <- dplyr::select(pars_df, omega)

base <- ggplot(omega_df, aes(x = omega)) +
  geom_density(colour = "steelblue") +
  scale_x_continuous(limits = c(0, 1))

g4 <- base + geom_function(fun = dlnorm, args = list(meanlog = -0.75, sdlog = 0.1),
                           colour = "grey50") +
  theme_pubr() +
  labs(x = bquote(gamma[1]), y = "") +
  theme(axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.line.y = element_blank())

g4
```

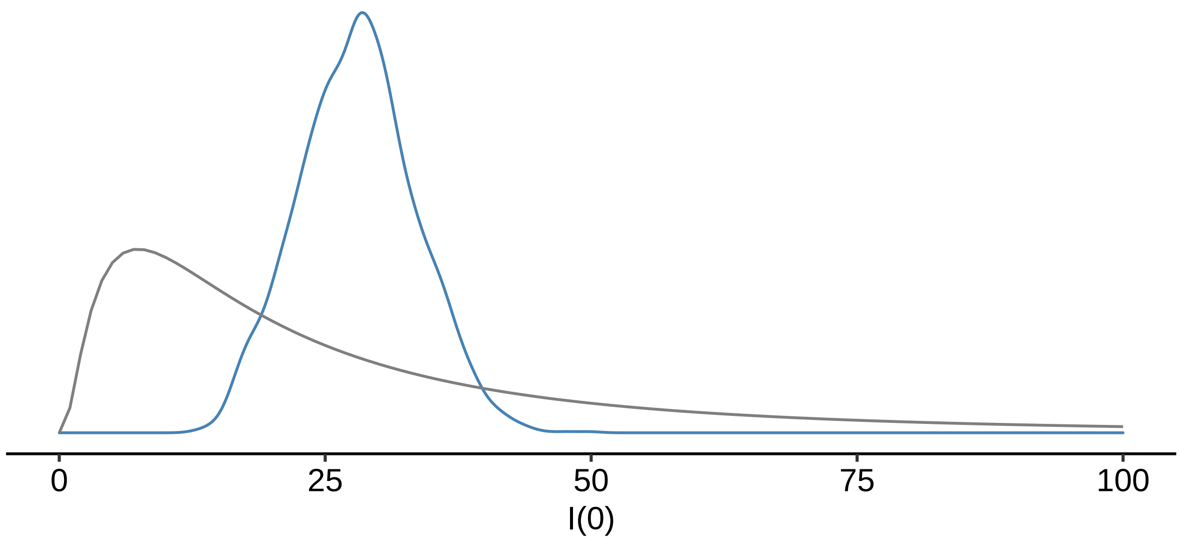



```r
alpha_df <- dplyr::select(pars_df, alpha)

base <- ggplot(alpha_df, aes(x = alpha)) +
  geom_density(colour = "steelblue") +
  scale_x_continuous(limits = c(0, 5))

g5 <- base + geom_function(fun = dlnorm, args = list(meanlog = 0.75, sdlog = 0.1),
                           colour = "grey50") +
  theme_pubr() +
  labs(x = bquote(alpha), y = "") +
  theme(axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.line.y = element_blank())

g5
```

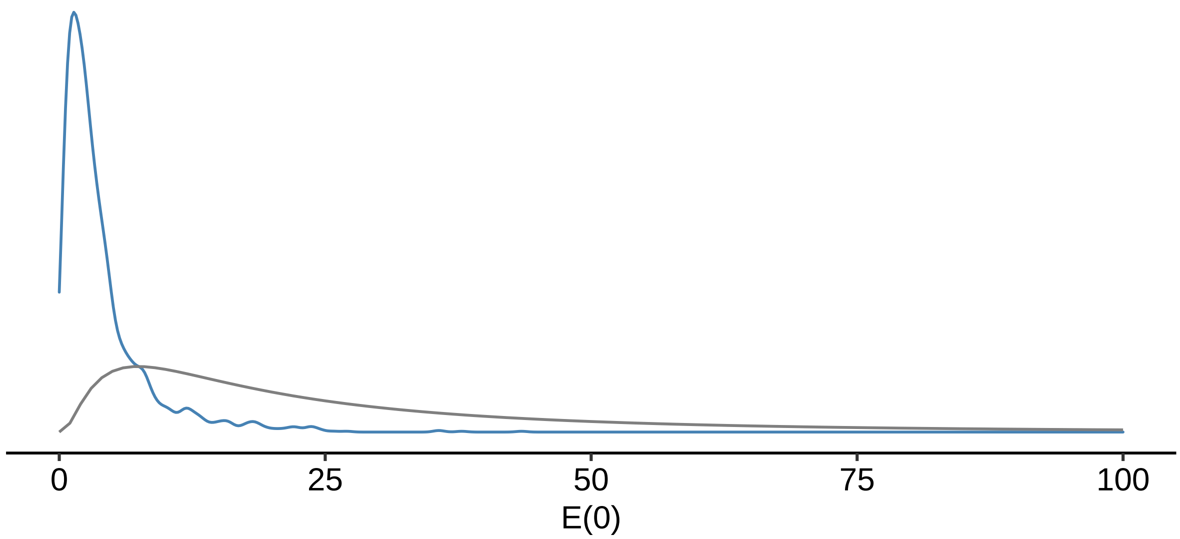



```r
I0_df <- dplyr::select(pars_df, I0)

base <- ggplot(I0_df, aes(x = I0)) +
  geom_density(colour = "steelblue") +
```

```
  scale_x_continuous(limits = c(0, 100))

g6 <- base + geom_function(fun = dlnorm, args = list(meanlog = 3, sdlog = 1),
                           colour = "grey50") +
  theme_pubr() +
  labs(x = "I(0)", y = "") +
  theme(axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.line.y = element_blank())

g6
```



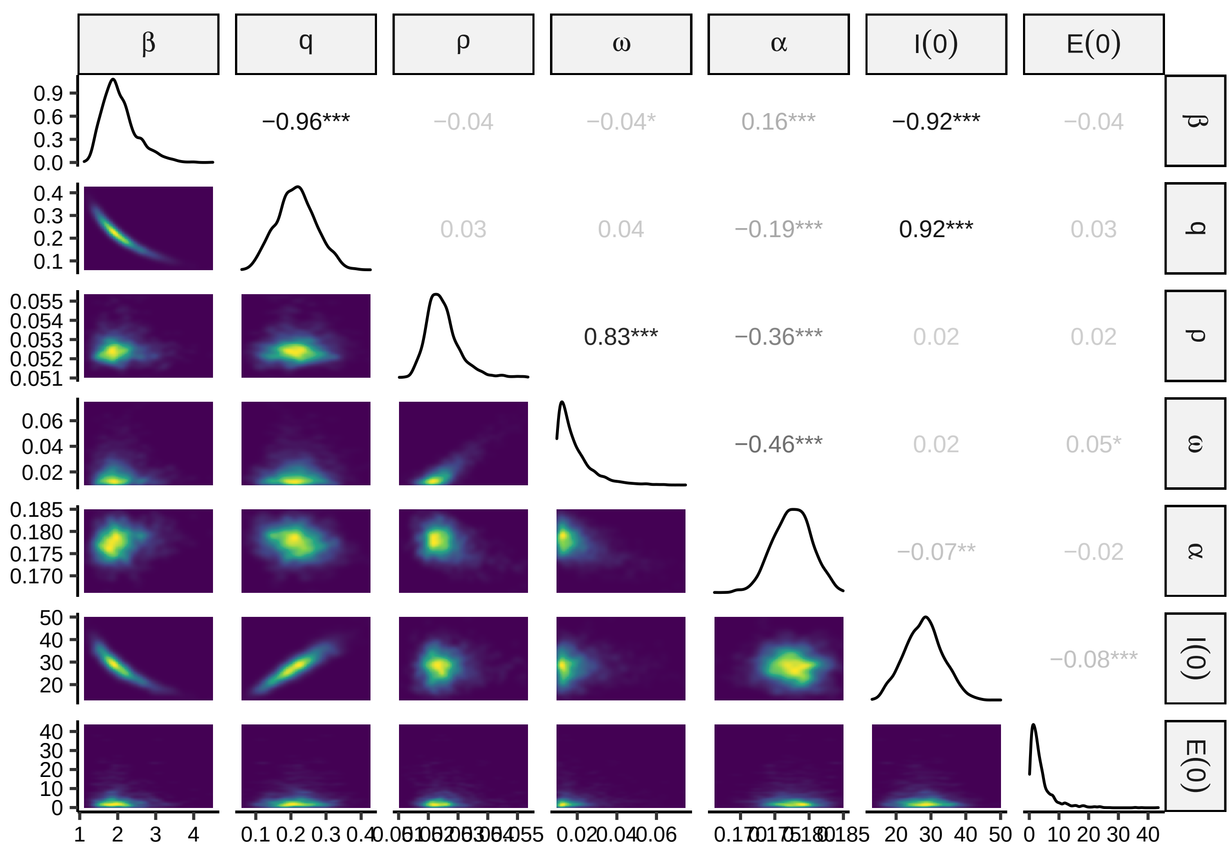

```
E0_df <- dplyr::select(pars_df, E0)

base <- ggplot(E0_df, aes(x = E0)) +
  geom_density(colour = "steelblue") +
  scale_x_continuous(limits = c(0, 100))

g7 <- base + geom_function(fun = dlnorm, args = list(meanlog = 3, sdlog = 1),
                           colour = "grey50") +
  theme_pubr() +
  labs(x = bquote("E(0)"), y = "") +
  theme(axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.line.y = element_blank())

g7
```



3.3.2.2 Pair plots

This plot helps us evaluate parameter interactions or joint distributions. The lower triangular shows, through heat maps, the concentration of values in the x-y plane among all possible parameter pairs. The upper triangular quantifies such interactions by the correlation coefficients. The diagonal displays marginal posterior distributions.

```
g <- pairs_posterior_2(pars_df, strip_text = 10, axis_text_size = 8)
print(g)
```



4 Original Computing Environment

```
sessionInfo()
```

```
## R version 4.1.2 (2021-11-01)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] viridisLite_0.4.1 tidyr_1.2.1       stringr_1.4.1     scales_1.2.1
##  [5] writexl_1.4.0     readxl_1.4.1      readsdr_0.2.0     readr_2.1.3
##  [9] purrr_0.3.5       posterior_1.3.1   patchwork_1.1.2   Metrics_0.1.4
## [13] lubridate_1.8.0   kableExtra_1.3.4  gridExtra_2.3     ggridges_0.5.4
## [17] ggpubr_0.4.0      reshape2_1.4.4    GGally_2.1.2      ggplot2_3.3.6
## [21] extraDistr_1.9.1  dplyr_1.0.10      cmdstanr_0.5.0    bayesplot_1.9.0
##
## loaded via a namespace (and not attached):
##  [1] httr_1.4.4           jsonlite_1.8.2       carData_3.0-5
##  [4] distributional_0.3.1 highr_0.9            tensorA_0.36.2
##  [7] cellranger_1.1.0     yaml_2.3.5           pillar_1.8.1
## [10] backports_1.4.1      glue_1.6.2           digest_0.6.29
## [13] RColorBrewer_1.1-3   ggsignif_0.6.4       checkmate_2.1.0
## [16] rvest_1.0.3          colorspace_2.0-3     htmltools_0.5.3
## [19] plyr_1.8.7           pkgconfig_2.0.3      broom_1.0.1
## [22] webshot_0.5.4        processx_3.7.0       svglite_2.1.0
## [25] tzdb_0.3.0           tibble_3.1.8         generics_0.1.3
## [28] farver_2.1.1         car_3.1-0            ellipsis_0.3.2
## [31] withr_2.5.0          cli_3.4.1            magrittr_2.0.3
## [34] ps_1.7.1             evaluate_0.17        fansi_1.0.3
## [37] MASS_7.3-58.1        rstatix_0.7.0        xml2_1.3.3
## [40] tools_4.1.2          hms_1.1.2            matrixStats_0.62.0
## [43] lifecycle_1.0.3      munsell_0.5.0        compiler_4.1.2
## [46] systemfonts_1.0.4    rlang_1.0.6          grid_4.1.2
## [49] rstudioapi_0.14      labeling_0.4.2       rmarkdown_2.17
## [52] gtable_0.3.1         abind_1.4-5          reshape_0.8.9
## [55] rematch_1.0.1        R6_2.5.1             knitr_1.40
## [58] fastmap_1.1.0        utf8_1.2.2           stringi_1.7.8
## [61] Rcpp_1.0.9           vctrs_0.4.2          tidyselect_1.2.0
## [64] xfun_0.33
```