Fitting COVID-19 case data with covidseir

The covidseir R package can be installed from https://github.com/seananderson/covidseir.

```
library(covidseir)
library(dplyr)
library(ggplot2)
ymd <- lubridate::ymd
```

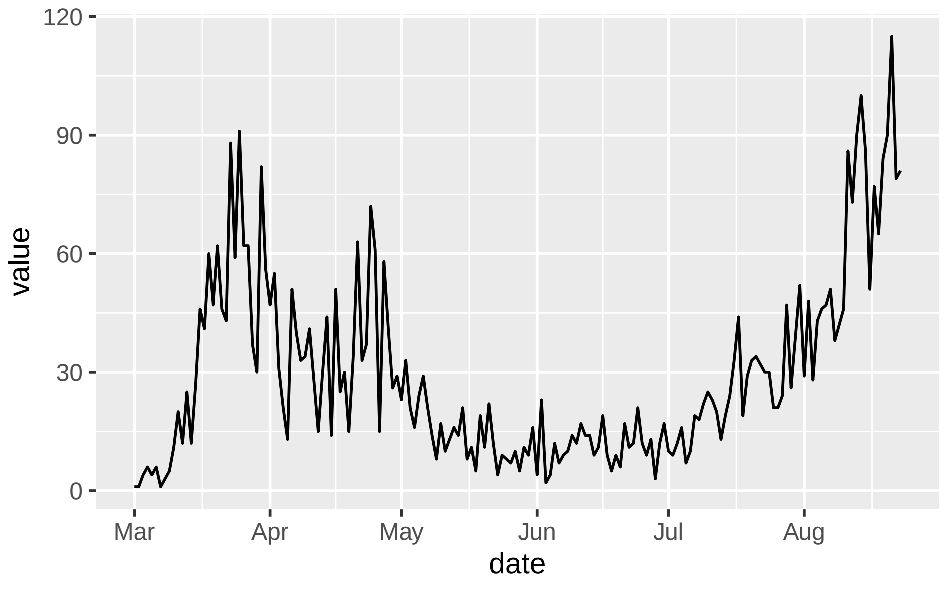
Data

First, we will read in the British Columbia, Canada COVID-19 reported-case data. This comes from http://www.bccdc.ca/Health-Info-Site/Documents/BCCDC_COVID19_Dashboard_Case_Details.csv.

```
dat <- structure(list(value = c(1, 1, 4, 6, 4, 6, 1, 3, 5, 11, 20, 12,
25, 12, 27, 46, 41, 60, 47, 62, 46, 43, 88, 59, 91, 62, 62, 37,
30, 82, 56, 47, 55, 31, 21, 13, 51, 40, 33, 34, 41, 28, 15, 30,
44, 14, 51, 25, 30, 15, 34, 63, 33, 37, 72, 61, 15, 58, 41, 26,
29, 23, 33, 21, 16, 24, 29, 21, 14, 8, 17, 10, 13, 16, 14, 21,
8, 11, 5, 19, 11, 22, 12, 4, 9, 8, 7, 10, 5, 11, 9, 16, 4, 23,
2, 4, 12, 7, 9, 10, 14, 12, 17, 14, 14, 9, 11, 19, 9, 5, 9, 6,
17, 11, 12, 21, 12, 9, 13, 3, 12, 17, 10, 9, 12, 16, 7, 10, 19,
18, 22, 25, 23, 20, 13, 19, 24, 33, 44, 19, 29, 33, 34, 32, 30,
30, 21, 21, 24, 47, 26, 39, 52, 29, 48, 28, 43, 46, 47, 51, 38,
42, 46, 86, 73, 90, 100, 86, 51, 77, 65, 84, 90, 115, 79, 81),
    day = 1:176), row.names = c(NA, -176L),
  class = "data.frame")
dat <- dplyr::as_tibble(dat)
dat$date <- ymd("2020-03-01") + dat$day - 1
```

```
dat
#> # A tibble: 176 x 3
#>    value   day date
#>    <dbl> <int> <date>
#>  1     1     1 2020-03-01
#>  2     1     2 2020-03-02
#>  3     4     3 2020-03-03
#>  4     6     4 2020-03-04
#>  5     4     5 2020-03-05
#>  6     6     6 2020-03-06
#>  7     1     7 2020-03-07
#>  8     3     8 2020-03-08
#>  9     5     9 2020-03-09
#> 10    11    10 2020-03-10
#> # ... with 166 more rows
```

```
ggplot(dat, aes(date, value)) + geom_line()
```



Model setup

Sampling fraction

We need an estimate of the fraction of positive cases that are sampled/detected. This could be based on serology testing and known changes in testing policy. In this case, we are basing the sampled fractions on known dates of changes in testing policy combined with numbers of hospitalizations and an assumed hospitalization rate from a model fit elsewhere.

We will set up a vector of assumed sampling fractions with one value per day and changes on the appropriate days:

```
# Based on estimation with hospital data in other model:
samp_frac <- c(rep(0.14, 13), rep(0.21, 38))
samp_frac <- c(samp_frac, rep(0.37, nrow(dat) - length(samp_frac)))
```

Contact rate breakpoints

We then need to set up a vector of ‘f’ (contact rate fraction) segment IDs. These start at 0 (before any social distancing) and increment by 1 every time we want to estimate a new contact rate, say because of known social distancing policy changes. Here we will estimate new ‘f’ segments starting May 1 and June 1. Note that the vector needs to switch from 0 to 1 before the first estimated date of the social distancing ramp. The start and end of the ramp are estimated parameters. It is simplest to start the value 1 on the 2nd day.

```
f_seg <- c(0, rep(1, nrow(dat) - 1))
day_new_f <- which(dat$date == ymd("2020-05-01"))
f_seg[seq(day_new_f, length(f_seg))] <- 2
day_ch <- which(dat$date == ymd("2020-06-01"))
f_seg[seq(day_ch, length(f_seg))] <- 3
f_seg
#>   [1] 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
#>  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
#>  [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
#> [112] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
#> [149] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

Fitting the model

Now we can fit the SEIR model. This requires specifying a prior on 'i0' (people infected at initial point in time), the starting and ending of the ramp-in of social distancing (`start_decline_prior` and `end_decline_prior`), 'R0', and 'f'. The f prior by default is mean 0.4 and SD 0.2 from a beta distribution and applies to all f segments. You can extend this prior to be different for each 'f' segment by using a matrix, which likely makes sense if you think that social distancing has relaxed over time. See `?fit_seir` for details and default values.

In the following example, we will use `rstan::optimizing()` to find the MAP (maximum a posteriori probability) estimate combined with sampling from the covariance matrix to generate samples. Alternatives are variational Bayes (`"VB"`) and full Bayesian NUTS MCMC sampling (`"NUTS"`). We strongly recommend NUTS sampling for final inference; however, the MAP estimate can be useful for model experimentation and is what we will use here in this vignette for speed:

```
fit <- covidseir::fit_seir(
  daily_cases = dat$value,
  samp_frac_fixed = samp_frac,
  f_seg = f_seg,
  i0_prior = c(log(8), 1),
  e_prior = c(0.8, 0.05),
  start_decline_prior = c(log(15), 0.1),
  end_decline_prior = c(log(22), 0.1),
  f_prior = cbind(c(0.4, 0.5, 0.6), c(0.2, 0.2, 0.2)),
  R0_prior = c(log(2.6), 0.2),
  N_pop = 5.1e6, # BC population
  iter = 500, # number of posterior samples
  fit_type = "optimizing" # for speed only
)
#> Finding the MAP estimate.
```

```
print(fit)
#> MAP estimate:
#>            i0            R0 start_decline   end_decline        f_s[1]
#>          6.63          3.19         13.60         19.14          0.34
#>        f_s[2]        f_s[3]        phi[1]             e
#>          0.42          0.63          9.59          0.82
#> Mean in constrained space of MVN samples:
#>            i0            R0 start_decline   end_decline        f_s[1]
#>          7.34          3.20         13.62         19.13          0.35
#>        f_s[2]        f_s[3]        phi[1]             e
#>          0.42          0.63          9.72          0.82
#> SD in constrained space of MVN samples:
#>            i0            R0 start_decline   end_decline        f_s[1]
#>          3.85          0.24          1.18          1.50          0.05
#>        f_s[2]        f_s[3]        phi[1]             e
#>          0.05          0.04          1.61          0.05
```

Visualizing the model fit

If you would like, you can choose to use parallel processing for the projections:

```
future::plan(future::multisession)
```

Now we will take the fitted model and calculate the corresponding model predictions. This can be slow, especially with future projections, and so we only use the first 50 posterior samples for this quick vignette example:

```
proj <- covidseir::project_seir(fit, iter = 1:50)
proj
#> # A tibble: 8,800 x 7
#>      day data_type    mu y_rep   phi .iteration forecast
#>    <int>     <int> <dbl> <dbl> <dbl>      <int> <lgl>
#>  1     1         1  2.22     3  9.39          1 FALSE
#>  2     2         1  2.55     1  9.39          1 FALSE
#>  3     3         1  2.93     2  9.39          1 FALSE
#>  4     4         1  3.38     2  9.39          1 FALSE
#>  5     5         1  3.89     2  9.39          1 FALSE
#>  6     6         1  4.48     3  9.39          1 FALSE
#>  7     7         1  5.15     3  9.39          1 FALSE
#>  8     8         1  5.93    14  9.39          1 FALSE
#>  9     9         1  6.83     3  9.39          1 FALSE
#> 10    10         1  7.86     4  9.39          1 FALSE
#> # ... with 8,790 more rows
```

Then we will take the posterior samples, re-sample 20 times from the negative binomial observation model to generate smoother predictions, and transform the output into a tidy data frame for plotting:

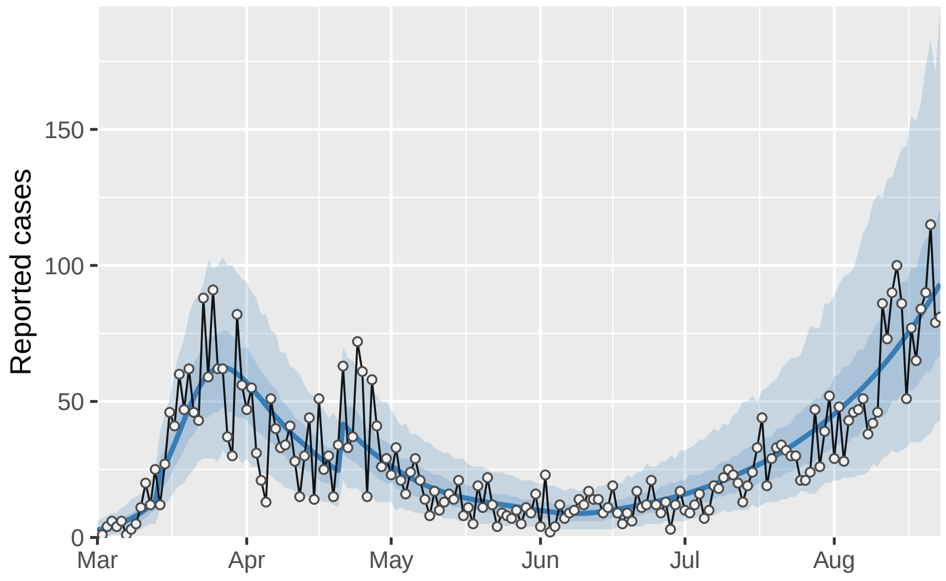
```
tidy_proj <- covidseir::tidy_seir(proj, resample_y_rep = 20)
tidy_proj
#> # A tibble: 176 x 15
#> # Groups:   data_type [1]
#>    data_type   day y_rep_0.05 y_rep_0.25 y_rep_mean y_rep_0.50 y_rep_0.75
#>        <int> <int>      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>
#>  1         1     1          0          1       2.68          2          4
#>  2         1     2          0          2       3.04          3          4
#>  3         1     3          0          2       3.58          3          5
#>  4         1     4          1          2       4.03          4          5
#>  5         1     5          1          3       4.60          4          6
#>  6         1     6          1          3       5.30          5          7
#>  7         1     7          2          4       6.17          6          8
#>  8         1     8          2          4       6.99          6          9
#>  9         1     9          2          5       7.93          7         10
#> 10         1    10          3          6       9.04          8         12
#> # ... with 166 more rows, and 8 more variables: y_rep_0.95 <dbl>,
#> #   mu_0.05 <dbl>, mu_0.25 <dbl>, mu_mean <dbl>, mu_0.50 <dbl>, mu_0.75 <dbl>,
#> #   mu_0.95 <dbl>, mu_0.5 <dbl>
```

For plotting, we need to join a date column back on, since our projections only have a column for a numeric day. Here, we do that by making a look-up-table (`lut`).

```
first_day <- min(dat$date)
last_day <- 300 # how many days to create dates for
lut <- dplyr::tibble(
  day = seq_len(last_day),
  date = seq(first_day, first_day + length(day) - 1, by = "1 day")
)
tidy_proj <- dplyr::left_join(tidy_proj, lut, by = "day")
dplyr::glimpse(tidy_proj)
#> Rows: 176
#> Columns: 16
#> Groups: data_type [1]
#> $ data_type  <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
#> $ day        <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1...
#> $ y_rep_0.05 <dbl> 0.00, 0.00, 0.00, 1.00, 1.00, 1.00, 2.00, 2.00, 2.00, 3....
#> $ y_rep_0.25 <dbl> 1.00, 2.00, 2.00, 2.00, 3.00, 3.00, 4.00, 4.00, 5.00, 6....
#> $ y_rep_mean <dbl> 2.683, 3.045, 3.582, 4.027, 4.603, 5.298, 6.170, 6.986, ...
#> $ y_rep_0.50 <dbl> 2, 3, 3, 4, 4, 5, 6, 6, 7, 8, 10, 11, 13, 22, 26, 29, 34...
#> $ y_rep_0.75 <dbl> 4, 4, 5, 5, 6, 7, 8, 9, 10, 12, 13, 15, 17, 29, 33, 38, ...
#> $ y_rep_0.95 <dbl> 6.00, 7.00, 8.00, 9.00, 9.00, 11.00, 12.00, 14.00, 15.00...
#> $ mu_0.05    <dbl> 1.936847, 2.258695, 2.634007, 3.071667, 3.582032, 4.1771...
#> $ mu_0.25    <dbl> 2.390450, 2.726700, 3.110217, 3.645367, 4.253651, 4.8830...
#> $ mu_mean    <dbl> 2.703382, 3.093344, 3.540019, 4.051735, 4.638050, 5.3099...
#> $ mu_0.50    <dbl> 2.733778, 3.129083, 3.579337, 4.071412, 4.658628, 5.3314...
#> $ mu_0.75    <dbl> 3.058550, 3.460075, 3.928361, 4.412826, 4.970541, 5.6329...
#> $ mu_0.95    <dbl> 3.305191, 3.765241, 4.280680, 4.886211, 5.566183, 6.3474...
#> $ mu_0.5     <dbl> 2.733778, 3.129083, 3.579337, 4.071412, 4.658628, 5.3314...
#> $ date       <date> 2020-03-01, 2020-03-02, 2020-03-03, 2020-03-04, 2020-03...
```

You can plot the output however you would like; however, the covidseir package includes a built-in basic plotting function. The `value_column` and `date_column` must both be columns present in the projection and observed data.

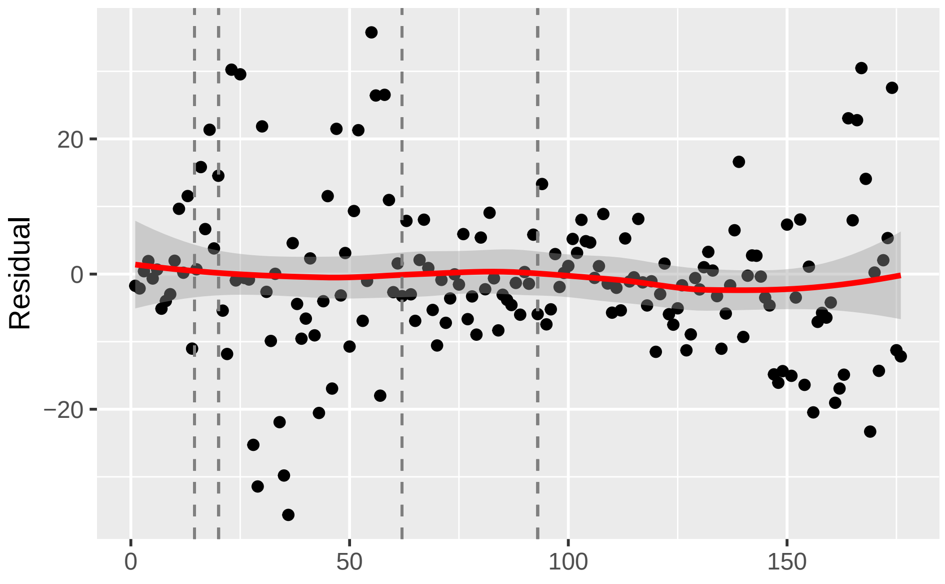
```
covidseir::plot_projection(tidy_proj, obs_dat = dat,
  value_column = "value", date_column = "date")
```



Checking model residuals

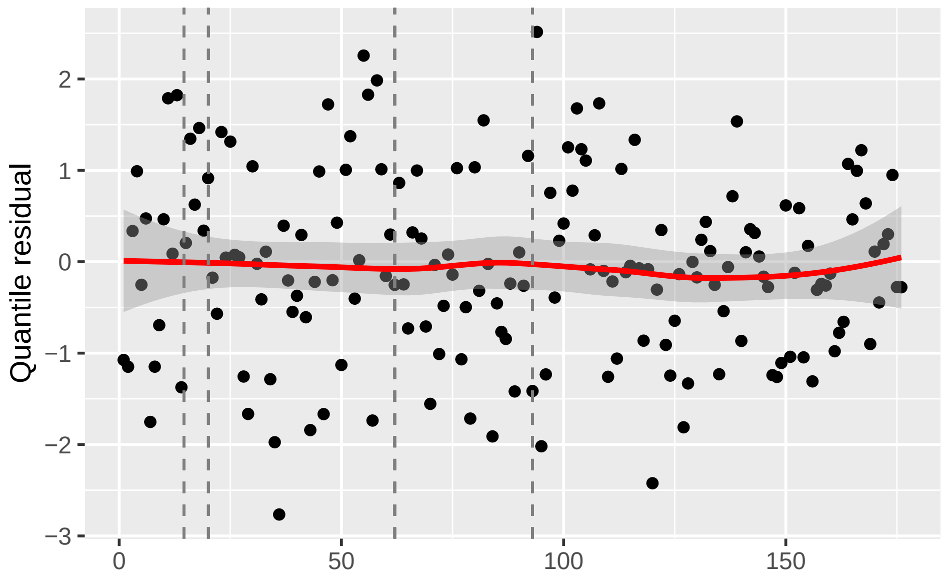
We can check the model fit residuals (here observed minus expected value):

```
plot_residuals(tidy_proj, obs_dat = dat, obj = fit)
```



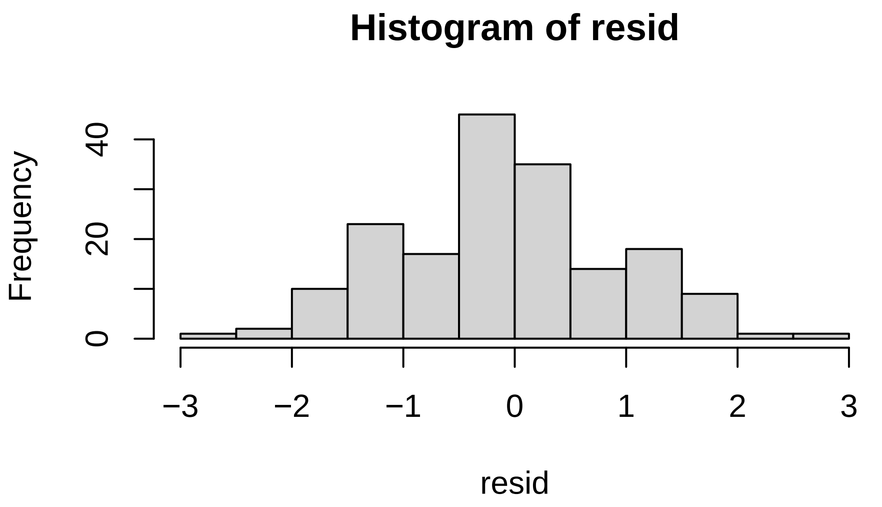
Or randomized quantile residuals, which should be approximately normally distributed (but introduce some randomness within each integer observed value):

```r
set.seed(1)
plot_residuals(tidy_proj, obs_dat = dat, obj = fit, type = "quantile")
```

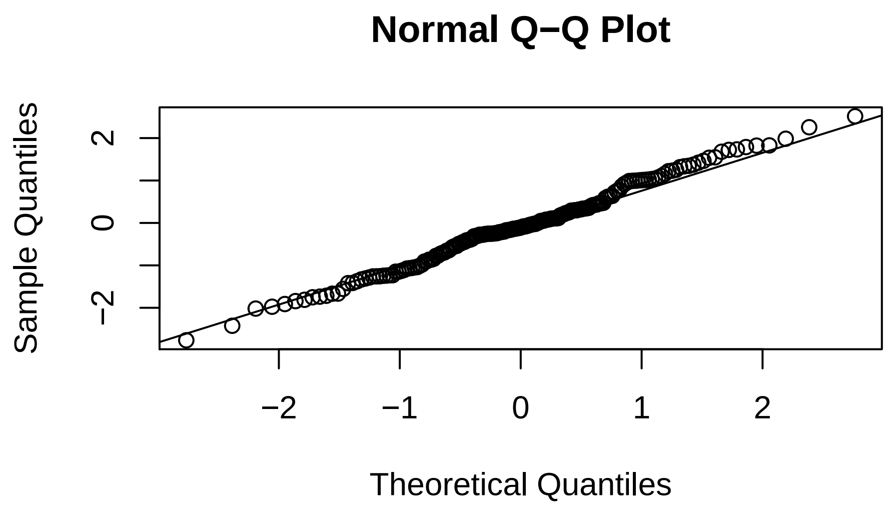


We can extract the (randomized quantile) residuals themselves to plot them however we would like or check their distribution:

```r
set.seed(1)
resid <- plot_residuals(tidy_proj, obs_dat = dat, obj = fit, type = "quantile",
  return_residuals = TRUE)
hist(resid)
```



```
qqnorm(resid)
qqline(resid)
```

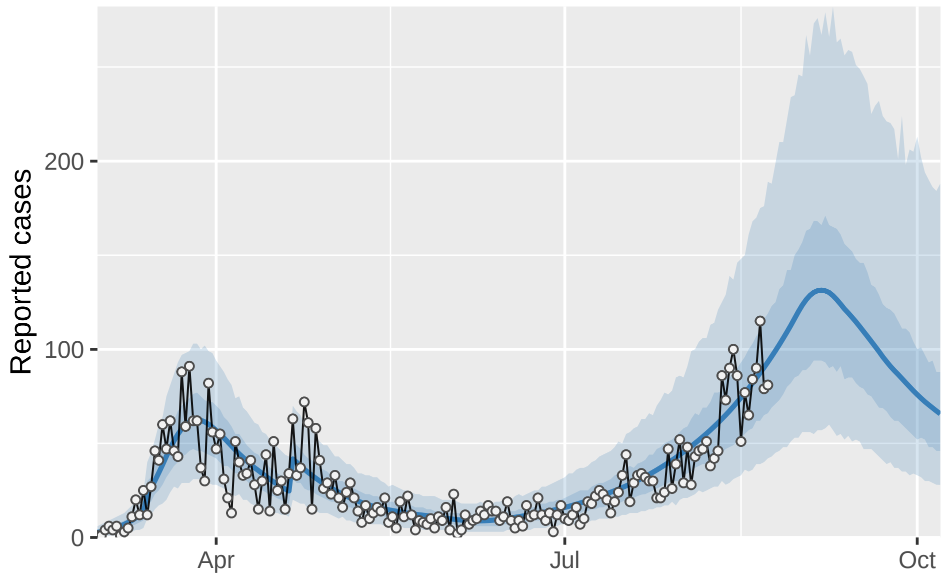


Projections

If we wanted to make a projection assuming that social distancing were to change in the future, we could do that as follows. For this example we will project 45 days into the future and change the contact ratio 'f' to be two-thirds of its final estimated value ('f' segment 3) starting after 5 days from the last observation. We will use only the first 50 posterior samples for speed of this example.

```
days_project <- 45
day_start_reduction <- 5
proj2 <- covidseir::project_seir(
  fit,
  iter = 1:50,
  forecast_days = days_project,
  f_fixed_start = max(fit$days) + day_start_reduction,
  f_multi = rep(0.67, days_project - day_start_reduction + 1),
  f_multi_seg = 3 # which f segment to use
)
tidy_proj2 <- covidseir::tidy_seir(proj2, resample_y_rep = 30)
tidy_proj2 <- dplyr::left_join(tidy_proj2, lut, by = "day")
```

```r
covidseir::plot_projection(tidy_proj2, obs_dat = dat,
  value_column = "value", date_column = "date")
```

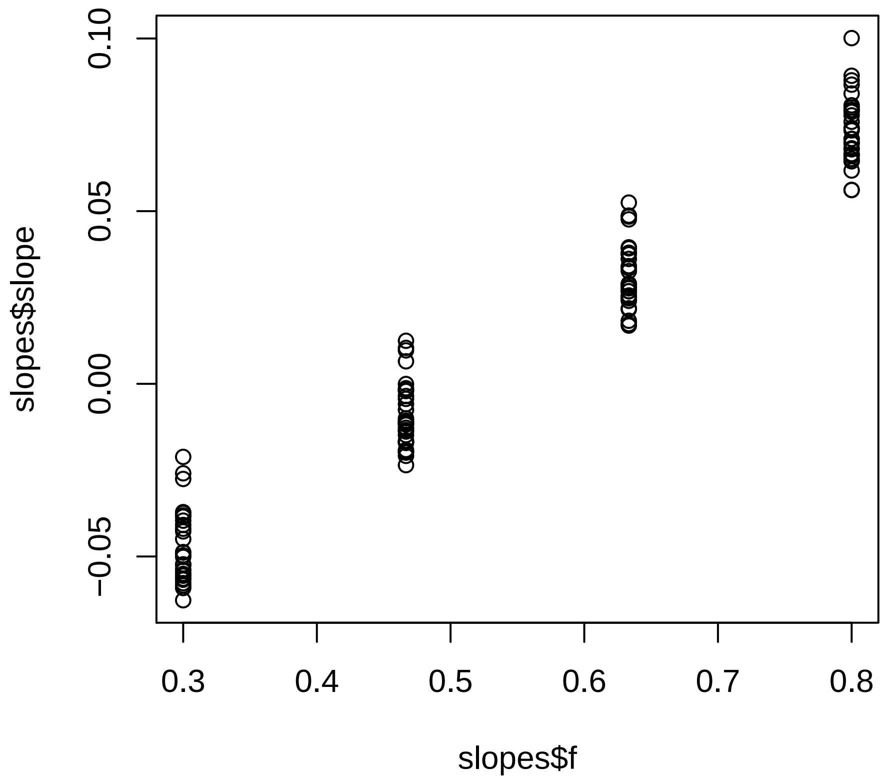


Calculating the threshold for increase

We can also calculate the threshold for expected increases in prevalence. We estimate the threshold by determining which 'f' value gives a zero rate of growth using the following procedure:

- For a sequence of fractions of normal contacts (f), project the model posterior for N days into the future.
- Fit a linear regression to determine the slope of log(prevalence) vs. time for the last X days of the projection period for each of the f values.
- Fit a linear regression to the slopes from the previous step against the fraction of normal contacts.
- Use this fitted regression line to determine what fractional normal contacts would result in an expected change in log prevalence of zero over time based on where the regression line crosses 0 on the y-axis.

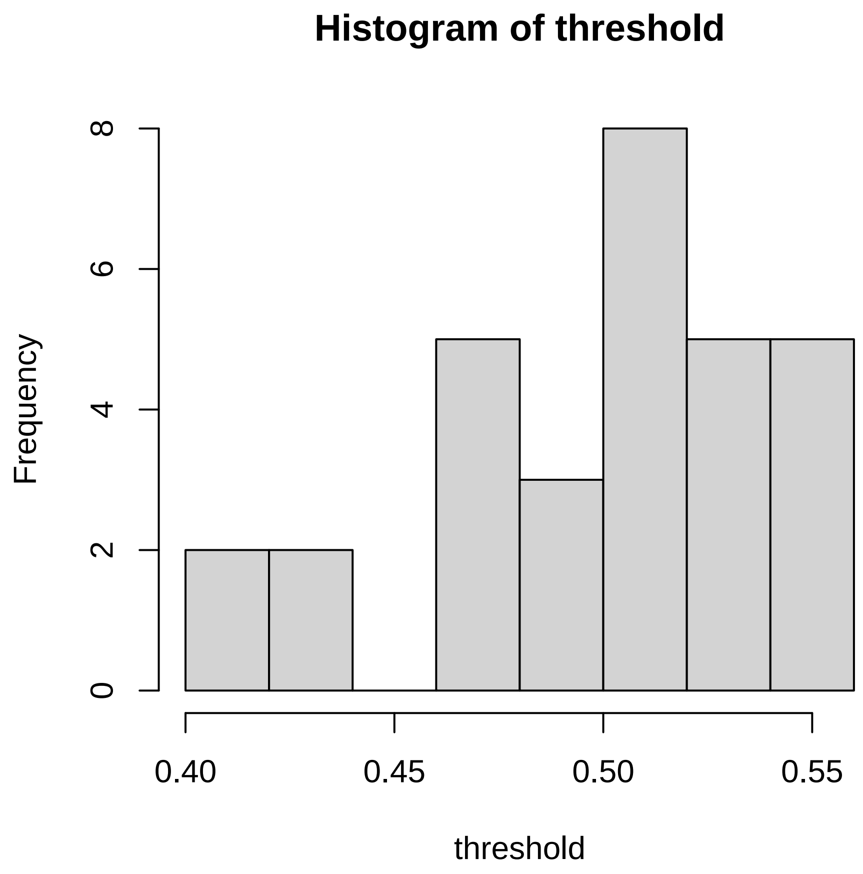
```r
threshold <- covidseir::get_threshold(fit, iter = 1:30,
  fs = seq(0.3, 0.8, length.out = 4))
#> Projecting 0.3
#> Projecting 0.47
#> Projecting 0.63
#> Projecting 0.8
```



We need to check that the above plot looks reasonable to fit a straight line through so we can extrapolate the point at which the slope values cross zero. It is possible the upper 'f' values (`fs`) will cause the population to reach heard immunity, resulting in prevalence decreasing over time and negative slopes. In that case, it wouldn't make sense to fit a straight line through the points and the upper limit of explored 'f' values or the number of forecasted days should be reduced. Here, it looks reasonable.

The output is a posterior distribution of the threshold:

```
hist(threshold)
```

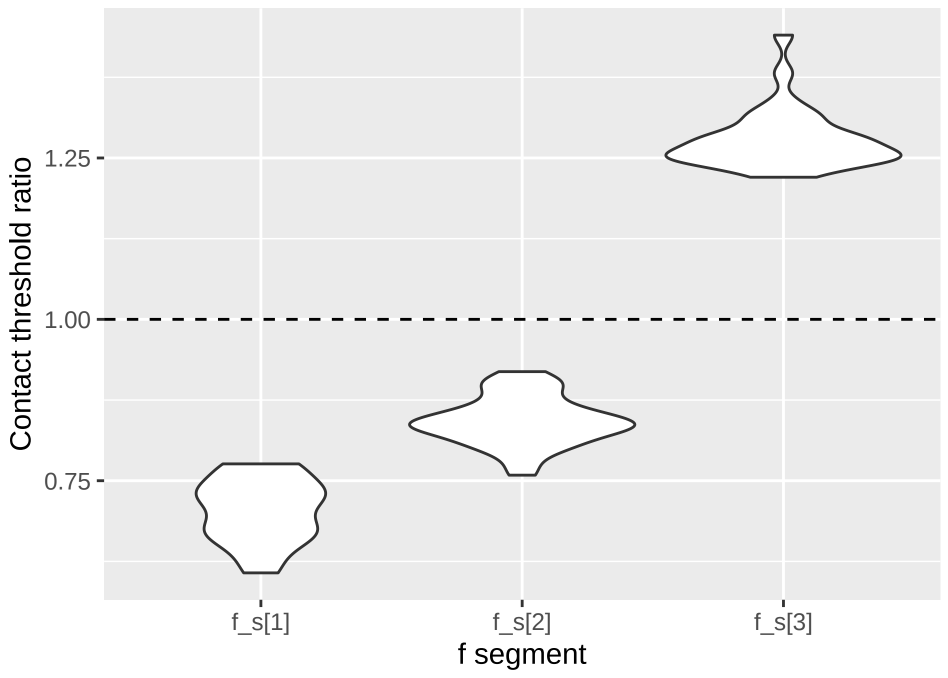


We can calculate the ratio between the posterior 'f' values for each segment and the threshold:

```
f_ratios <- fit$post$f_s[seq_along(threshold), ] / threshold
```

Here is an example of visualizing the output:

```
reshape2::melt(f_ratios) %>%
  ggplot(aes(Var2, value)) + geom_violin() +
  geom_hline(yintercept = 1, lty = 2) +
  xlab("f segment") + ylab("Contact threshold ratio")
```


Contact threshold ratio
1.25
1.00
0.75
f_s[1]
f_s[2]
f_s[3]
f segment