SEIDR Model with inference

Jamie Prentice

2026-01-09

Load libraries and source files

This pulls in all the necessary libraries and source files

```
suppressPackageStartupMessages({
    source("libraries.R")
    source("source_files.R")
})
```

Generating a params list

The first step is to generate a params list that contains the basic information, with some messages to note exactly what it's using. All elements in `make_parameters()` have default values, so these are just to see what you can add.

The params list might not fully do everything we want, but as it is just a basic R list, we can easily modify it afterwards.

```
params <- make_parameters(
    model_type = "SEIDR", # c("SIR", "SEIR", "SIDR", "SEIDR")
    dataset = "testing",
    name = "scen-1-1",
    setup = "fb_12_rpw", # "small", "fb_12", "fb_1", "fb_2", "single", etc.
    use_traits = "sit", # "all", "none", "sit", "si" etc.
    vars = c(sus = 1.0, inf = 1.5, tol = 0.5), # 0.5
    cors = c(si = 0.3, st = 0.2, it = 0.2), # 0.2
    group_layout = "fishboost", # "random", "family", "striped", "fishboost"
    trial_fe = "ildt", # ""
    donor_fe = "ildt", # ""
    txd_fe = "ildt", # ""
    weight_fe = "sildt", # ""
    weight_is_nested = TRUE,
    sim_new_data = "r" # c("bici", "r", "no", "etc_inf", "etc_ps")
)

# These are for doing a very quick local run of BICI, the actual values should
# be much higher if doing a real run on Eddie.
params$nchains <- 2L # 16L
params$nsample <- 1e2L # 2e6L
```

```
params$sample_states <- 100L # 1e3L

# We also set the time_step to 1, and censor the data (very useful as the last
# 20% of the simulation is typically very uneventful).
params$time_step <- 1
params$censor <- 0.8
```

This provides a short summary of the params

```
summarise_params(params)
```

```
## Simulating new SEIDR data via R

## - Demography is:
##     28 sires, 25 dams, 1750 progeny (1803 total), 70 groups (25 per group)
##     R0 = 10
## - FEs are
##     Trial = ildt, Donor = ildt, TxD = ildt, Weight = sildt
## - Running MCMC with:
##     100 updates / 10,000 samples / 0.2 burnin / 2 chains
## - Data will be saved to 'datasets/testing/data/scen-1-1.bici'

## Sigma_G =

##     sus inf lat det tol
## sus 1.0 0.3   0   0 0.2
## inf 0.3 1.5   0   0 0.2
## lat 0.0 0.0   0   0 0.0
## det 0.0 0.0   0   0 0.0
## tol 0.2 0.2   0   0 0.5
```

Generate pedigree and popn

Next we take the params file, set the groups and traits, and apply any fixed effects. Note that you these are piped into each other, but you can save the intermediate result at any point to examine what's happening (e.g. for bug fixing).

```
popn <- make_pedigree(params) |>
    set_groups(params) |>
    set_traits(params) |>
    apply_fixed_effects(params)
```

It's worth taking a look at the popn file to see what it includes. The columns are:

```
names(popn)
```

```
##  [1] "id"     "sire"   "dam"    "sdp"    "weight" "trial"  "group"  "donor"
##  [9] "GE"     "sus_g"  "inf_g"  "tol_g"  "sus_e"  "inf_e"  "tol_e"  "sus"
## [17] "inf"    "tol"    "lat"    "det"
```

Where:

- `id`: the unique ID of the individual. Sires come first, then dams, then progeny.
- `sire` and `dam` of the individual (together with `id` this makes a pedigree).
- `sdp`: whether the individual is a sire, dam, or progeny.
- The `weight` in kg at start of experiment.
- Which `trial` the individual was assigned to.
- Which `group` the individual was assigned to.
- `donor`: if the individual was inoculated (1) or not (0).
- `GE`: a small tank dependent group effect.
- `sus_g`, `inf_g`, `tol_g`, `sus_e`, `inf_e`, `tol_e`: genetic and environmental values. These are normally distributed with mean 0.
- `sus`, `inf`, `tol`, `lat`, `det`: phenotypic values. These are log-normally distributed and incorporate any fixed effects.

A typical progeny looks like:

```
popn[sdp == "progeny"][1]
```

```
## Key: <id>
##       id  sire   dam     sdp weight trial group donor    GE    sus_g    inf_g
##    <int> <int> <int>  <fctr>  <num> <int> <int> <int> <num>    <num>    <num>
## 1:    54    22    49 progeny   29.5     1     1     1     1 1.725466 1.768793
##        tol_g     sus_e    inf_e     tol_e      sus       inf       tol      lat
##        <num>     <num>    <num>     <num>    <num>     <num>     <num>    <num>
## 1: 0.2967071 0.2267179 -1.90113 -0.609229 2.658642 0.0695035 0.1365661 1.184444
##          det
##       <num>
## 1: 0.494508
```

Simulate and plot epidemic

We now pass the population file `popn` and the parameters `params` to `simulate_epidemic()`, which will run the appropriate model and return a *new* `popn` file with event times, final status, which generation infective they were, and the individual responsible for their infection. You can generate multiple realisations of the epidemic from the same inputs if you save the `popn` output each time.

```
tic(); popn <- simulate_epidemic(popn, params); toc()
```

```
## 15.628 sec elapsed
```

We can also use the generation value to calculate R_0, which is the mean across all groups of the ratio of secondary to primary infectives. Also it's useful to see how long the epidemic took until it completed (since this information is necessary for BICI to use).

```
params$estimated_R0 <- get_R0(popn)
```

```
## R0 estimate: 1.56
```

```
params$tmax <- get_tmax(popn, params)

message("Tmax = [", str_flatten_comma(round(params$tmax, 1)), "]")
```

```
## Tmax = [542, 1940]
```

The progeny we had before now looks like this, with the additional columns:

- `Tinf`: the infection time ($t : S \to E$). In actual data this is 0 for inoculated individuals, and missing for all others.
- `Tinc`: the incubation time ($t : E \to I$), missing in the data.
- `Tsym`: the time of symptoms ($t : I \to D$), present in data.
- `Tdeath`: the time of death ($t : D \to R$), present in data.
- `status`: which compartment the individual is in at the end of the epidemic (should only be `S` or `R`)
- `generation`: if an individual is a primary, secondary, tertiary etc. infective.
- `infected_by`: the id of the individual responsible for this infection (0 for inoculated individuals).
- `parasites` `TRUE` if an individual was infected. In actual data the sensitivity is not 100%.

```
popn[sdp == "progeny"][1]
```

```
##       id  sire   dam     sdp weight trial group donor    GE    sus_g    inf_g
##    <int> <int> <int>  <fctr>  <num> <int> <int> <int> <num>    <num>    <num>
## 1:    54    22    49 progeny   29.5     1     1     1     1 1.725466 1.768793
##        tol_g     sus_e    inf_e     tol_e      sus       inf       tol      lat
##        <num>     <num>    <num>     <num>    <num>     <num>     <num>    <num>
## 1: 0.2967071 0.2267179 -1.90113 -0.609229 2.658642 0.0695035 0.1365661 1.184444
##         det  Tinf     Tinc     Tsym  Tdeath status generation infected_by
##       <num> <num>    <num>    <num>   <num> <fctr>      <int>       <int>
## 1: 0.494508     0 4.738372 7.130005 7.43284      R          1           0
##    parasites
##       <lgcl>
## 1:      TRUE
```

Plot the epidemic

We can take a look at the time trajectory

```
plt <- plot_model(popn, params)
```

It's also important to look at the Kaplan-Meier survival plot.

```
plt2 <- basic_km(popn, params)
```

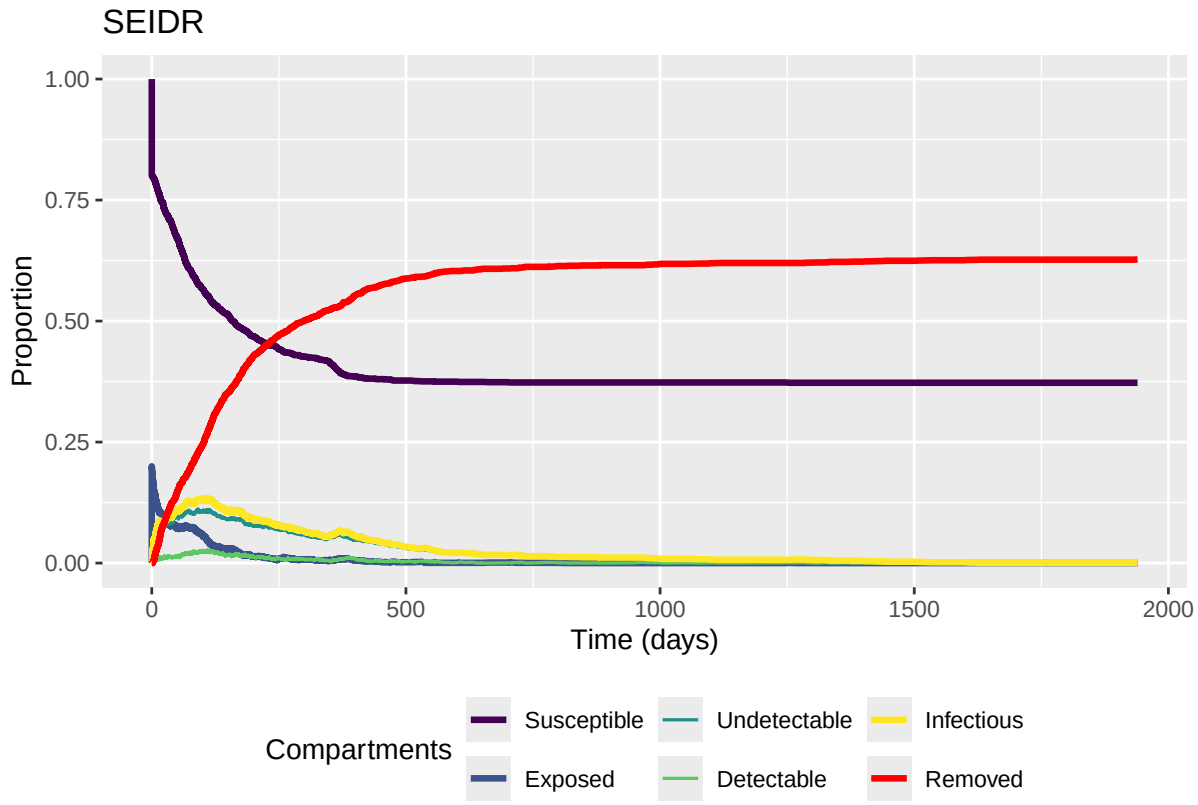


Figure 1: Time trajectory of SEIDR model

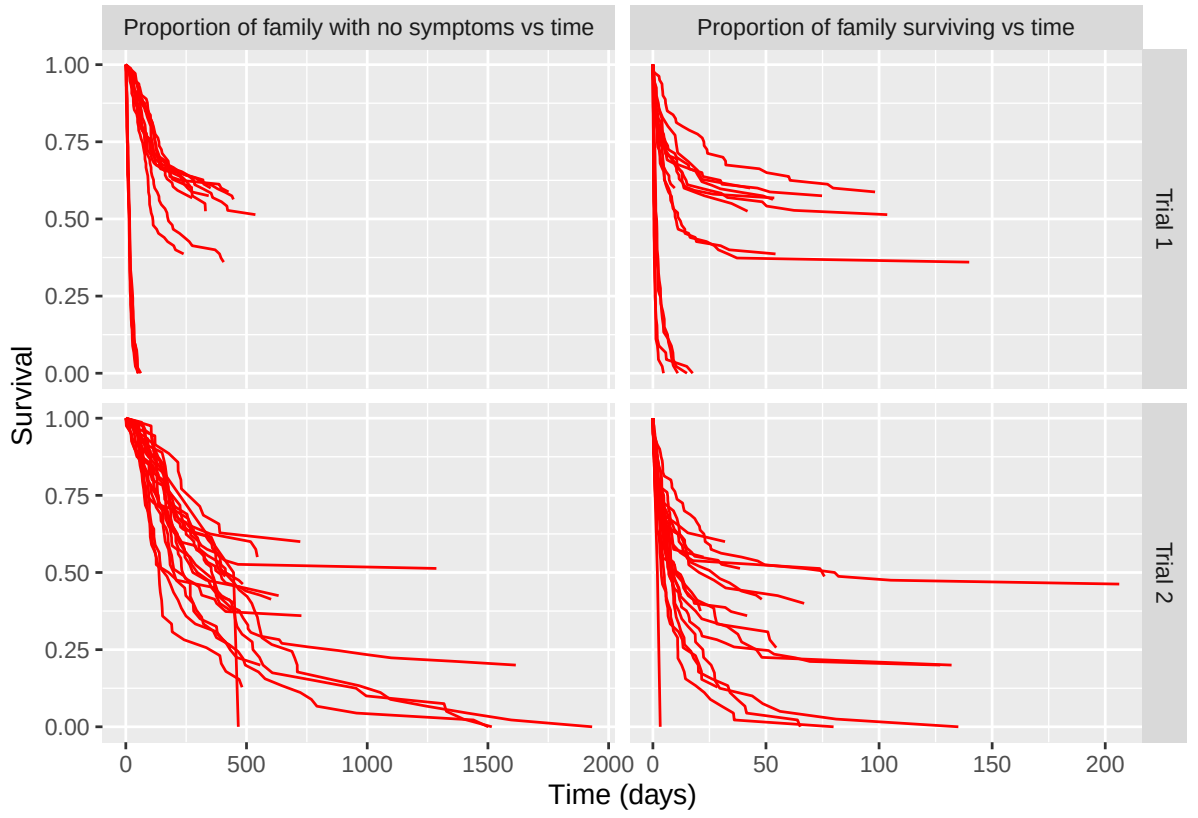


Figure 2: KM plot of SEIDR model, showing the proportion of each family surviving over time.

Generate directories and config files

In order for BICI to work, it needs the data and the model to fit to the data. We first generate the directories (all stored in the `params` file) and then create the BICI script file.

```
walk(params[str_ends(names(params), "_dir")], ~ {
    if (!dir.exists(.x)) dir.create(.x, recursive = TRUE)
})
# Remove any old files
cleanup_bici_files(params)
bici_txt <- generate_bici_script(popn, params)
```

Run BICI

Here we build `cmd`, the command that runs BICI, and run it with `system(cmd)`. There is some additional platform information in case the BICI directory contains an executable compiled for each platform ("bici-Linux", "bici-Darwin", or "bici-Windows"). If running the PAS algorithm, then BICI needs to be compiled with MPI support, and `--output :raw` is necessary for it to output a progress meter (otherwise it saves all output until it's finished running).

```
cmd <- with(params, str_glue(
    if (algorithm == "pas")
        "mpirun -n {nchains} --output :raw --oversubscribe " else "",
    "../BICI/bici-{platform} {data_dir}/{name}.bici {bici_cmd}",
    platform = Sys.info()[["sysname"]]
))
message(str_glue("Running:\n$ {cmd}"))

tic()
out <- system(cmd)
time_taken <- toc()

if (out != 0) {
    stop("BICI failed to finish")
}
```

Retrieve results and save

Finally we collect the output from BICI do some post-processing, and save the results in a format suitable for easily reading back in R. The results file contains the data, the model fitted to the data, summary statistics, prediction accuracies (if available), and the time taken to run (handy for determining how long to reserve on Eddie for future runs with possibly more samples).

`rebuild_bici_posteriors()` creates a `trace_combine.tsv` file that contains the output of all the chains combined into one, with the burn-in period removed, making it immediately useful for working with and sampling from the posterior. It also creates a summary file with each parameter's mean, median, 95% CI, 95% HPDI, ESS, and GR, which is very useful for convergence diagnostics.

```
{
    name <- params$name
    dataset <- params$dataset
    output_dir <- params$output_dir
    results_dir <- params$results_dir
    bici_cmd <- params$bici_cmd
}

if (bici_cmd == "inf") {
    parameter_estimates <- rebuild_bici_posteriors(dataset, name)

    ebvs_name     <- str_glue("{output_dir}/ebvs.csv")
    estimated_BVs <- if (file.exists(ebvs_name)) fread(ebvs_name)

    pa_name   <- str_glue("{output_dir}/pred_accs.csv")
    pred_accs <- if (file.exists(pa_name)) fread(pa_name)

    # msg_pars(parameter_estimates)
    print(parameter_estimates[!str_starts(parameter, "Group effect"),
                              .(parameter = rename_pars(parameter),
                                median = round(median, 3),
                                hdi95min = round(hdi95min, 3),
                                hdi95max = round(hdi95max, 3),
                                ESS,
                                GR = round(GR, 3))])

    time_end <- now()

    results <- c("params", "popn", "parameter_estimates", "estimated_BVs",
                 "ranks", "pred_accs", "time_taken", "time_start", "time_end") |>
        keep(exists)

    saveRDS(mget(results),
            file = str_glue("{results_dir}/{name}.rds"))
}
```

```
##                          parameter  median hdi95min hdi95max   ESS     GR
##                             <char>   <num>    <num>    <num> <num>  <num>
##  1:                     Var G (Sus)   1.334    1.117    1.685     5  4.300
##  2:                     Var G (Inf)   3.700    3.129    4.000     7  6.987
##  3:                     Var G (Tol)   0.055    0.036    0.100     6 10.363
##  4:                Cor G (Sus, Inf)   0.128    0.072    0.183    13  1.189
##  5:                Cor G (Sus, Tol)   0.629    0.497    0.736    11  6.951
##  6:                Cor G (Inf, Tol)  -0.119   -0.386    0.259     5 19.502
##  7:                     Var E (Sus)   0.614    0.424    0.765     8  5.500
##  8:                     Var E (Inf)   3.773    3.292    3.994    10  1.984
##  9:                     Var E (Tol)   0.950    0.812    1.122     6  3.539
## 10:                Cor E (Sus, Inf)   0.399    0.213    0.543     2 23.856
## 11:                Cor E (Sus, Tol)  -0.135   -0.260    0.116     2 12.915
## 12:                Cor E (Inf, Tol)  -0.189   -0.276   -0.121    19  1.070
## 13:                        beta Tr1   1.130    0.993    1.436     6  1.059
```

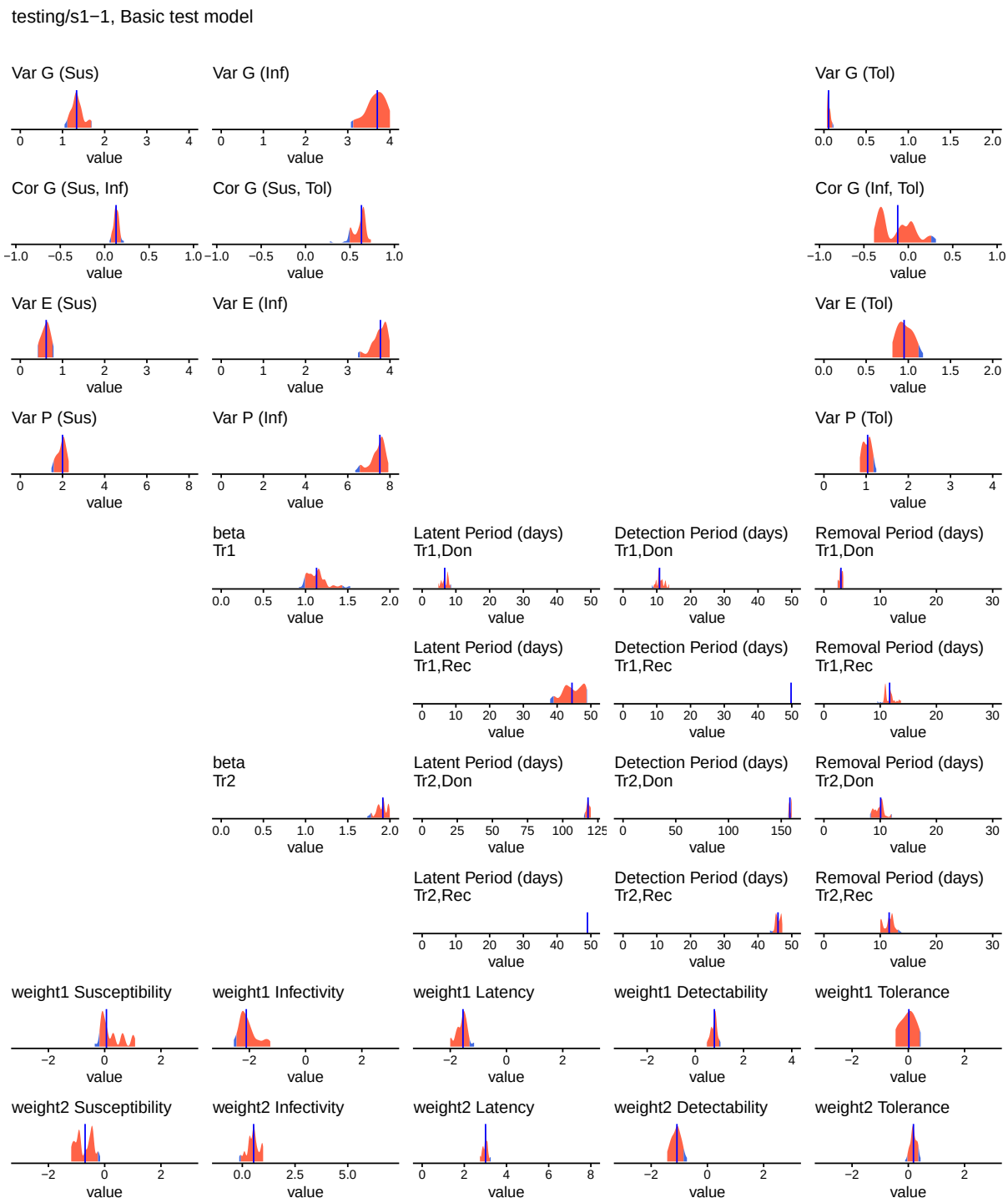
```
## 14:                          beta Tr2   1.916    1.777    1.998    10  1.083
## 15:    Latent Period (days) Tr1,Don   6.695    4.812    8.417     5  4.959
## 16:    Latent Period (days) Tr1,Rec  44.411   39.005   48.718     7  4.907
## 17:    Latent Period (days) Tr2,Don 117.972  115.875  119.939     5  3.303
## 18:    Latent Period (days) Tr2,Rec  48.992   48.830   49.159    10  1.186
## 19: Detection Period (days) Tr1,Don  10.813    9.045   13.646    16  1.000
## 20: Detection Period (days) Tr1,Rec  49.794   49.662   49.975    10  1.974
## 21: Detection Period (days) Tr2,Don 158.239  156.950  159.955     3  9.834
## 22: Detection Period (days) Tr2,Rec  45.916   44.340   47.201     5  7.548
## 23:   Removal Period (days) Tr1,Don   3.021    2.463    3.438     9  2.597
## 24:   Removal Period (days) Tr1,Rec  11.676   10.531   13.699     8  1.703
## 25:   Removal Period (days) Tr2,Don  10.078    8.364   11.983     8  4.230
## 26:   Removal Period (days) Tr2,Rec  11.595   10.016   13.203     7  4.859
## 27:          weight1 Susceptibility   0.062   -0.182    1.081     3  4.587
## 28:             weight1 Infectivity  -2.109   -2.457   -1.256     6  3.616
## 29:                 weight1 Latency  -1.547   -2.001   -1.313    11  1.108
## 30:           weight1 Detectability   0.788    0.482    0.993     9  3.352
## 31:               weight1 Tolerance   0.019   -0.453    0.409     7  1.156
## 32:          weight2 Susceptibility  -0.696   -1.189   -0.239     5 10.858
## 33:             weight2 Infectivity   0.536   -0.070    0.987     7  1.186
## 34:                 weight2 Latency   3.007    2.741    3.210    13  1.007
## 35:           weight2 Detectability  -1.083   -1.428   -0.846     8  4.811
## 36:               weight2 Tolerance   0.183   -0.030    0.357    16  2.981
##                         parameter  median hdi95min hdi95max   ESS     GR
```

Finally `flatten_bici_states()` creates an `etc-inf.rds` or `etc-ps.rds` summary file that contains snapshots of the state and the corresponding parameters.

```
flatten_bici_states(dataset, name, bici_cmd)
```

Now we can examine the output of BICI more closely. First generate a posterior plot

```
source("scripts/posterior.R")
get_posterior(dataset, scen_rep = str_remove(params$name, "scen-"))
```



The output is pretty bad as we using only 10^4 samples, but it's enough to get the idea.