Simulation-based calibration of capture-mark-recapture models in Stan

Matthijs Hollanders

This document contains the results of simulation-based calibration (SBC) for each model described in the manuscript. We assess calibration through ECDF plots and visually inspect parameter estimates alongside their true values. The simulation script is printed at the end of this document.

Table of contents

Introduction

Simulation-based calibration (SBC) was used to assess whether Stan programs correctly encoded the data-generating processes (DGPs). Briefly, SBC works by simulating prior predictive datasets, estimating the parameters for each of these datasets using the model, and then assessing the ranks of the simulated values within the posterior draws (Modrák et al. 2023). If the model reflects the DGP, then the ranks are uniformly distributed, which we assess with the `SBC::plot_ecdf_diff()` function. If the model is calibrated, then the black squiggly lines should fall within the blue ellipses. Parameter estimates were also plotted alongside their input to visually assess parameter recovery. SBC was only conducted for the log likelihood function signatures without individual-level parameters for speed.

In all models, constant state-specific mortality hazard rates and survey-varying state-specific detection probabilities were simulated. Survey intervals were simulated as unequal which are accounted for in the entry and ecological processes. In multistate models, survey intervals were scaled to have mean 1. In robust designs, primary-varying detection probabilities shared between secondaries were simulated, though Stan programs and functions accommodate secondary-level detection probabilities. In multistate models, state-to-state transition rates were simulated to be equal between surveys. In Jolly-Seber models, time-varying entry probabilities were simulated with an offset for survey interval as described in the manuscript. In multistate and multievent Jolly-Seber and multievent Cormack-Jolly-Seber models, the probabilities of entering or being first captured in each state are modeled as time-varying.

Cormack-Jolly-Seber models do not recover latent states because any derived quantities only pertain to captured individuals. In Jolly-Seber models, underlying latent states are recovered to derive the population size and number of entries and exits per survey/primary, as well as the super-population.

First we load packages and scripts and set the parameters for simulation and estimation. Models were fit with CmdStanR 0.9 (Gabry et al. 2025) with 500 simulated datasets per model.

```
library(SBC)
library(cmdstanr)
library(here)
library(tidyverse)
source(here("sbc/util.R"))
source(here("sbc/simulate-cmr.R"))
theme_set(my_theme(base_size = 6))
N_super <- 200 ; J <- 6 ; K <- 2 ; S <- 2 ; n_sims <- 500
chains <- 8 ; iter_warmup <- 200 ; iter_sampling <- 500
options(digits = 3, mc.cores = 8)
```

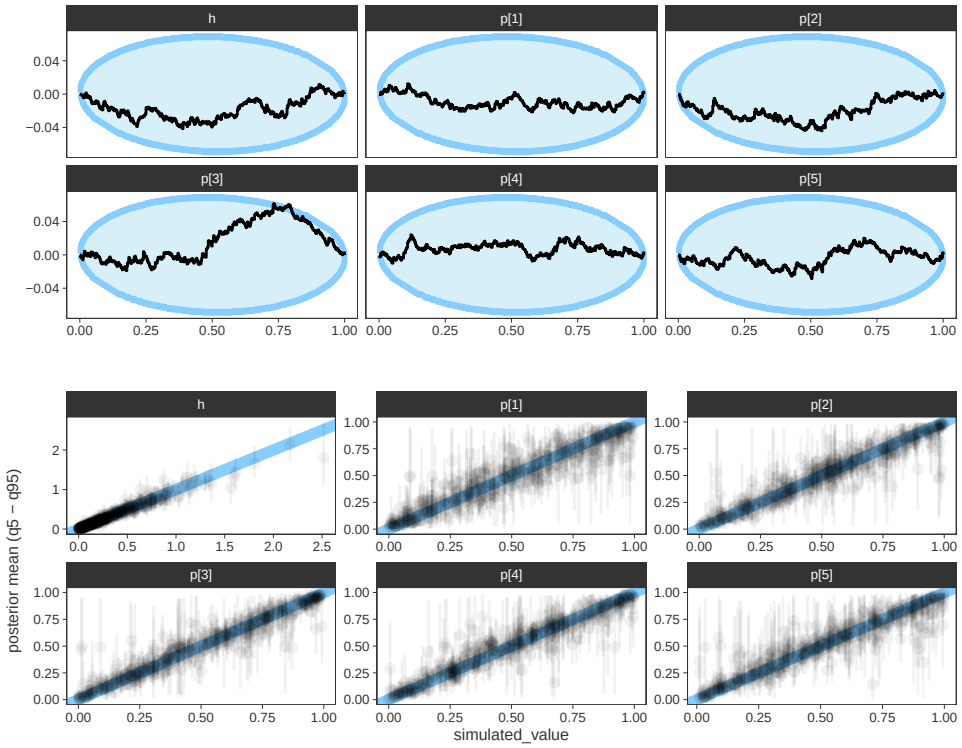
Cormack-Jolly-Seber

Single survey

```
datasets <- SBC_generator_function(simulate_cmr, N_super = N_super, J = J) |>
  generate_datasets(n_sims)
backend <- SBC_backend_cmdstan_sample(cmdstan_model(here("stan/cjs.stan")),
                                      init = 0.1, chains = chains,
                                      iter_warmup = iter_warmup,
                                      iter_sampling = iter_sampling)
sbc <- compute_SBC(datasets, backend, cores_per_fit = chains, keep_fits = F)
```

Mean time for the slowest of 8 chains with 700 iterations: 0.005 minutes.

```
plot_sbc(sbc, nrow = 2)
```

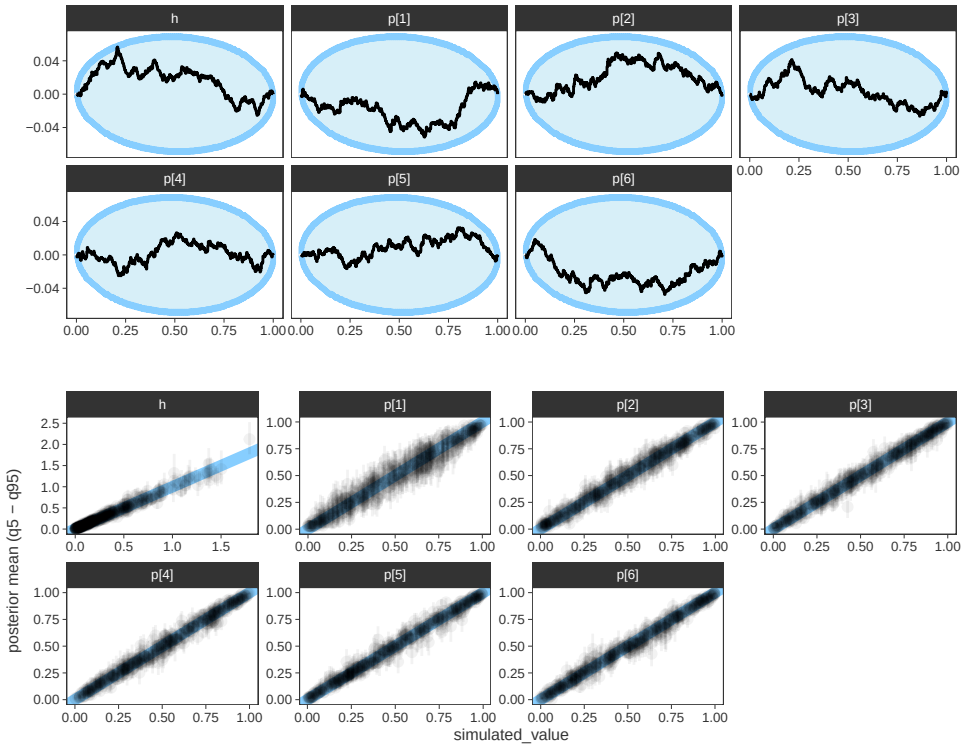


Robust design

```
datasets <- SBC_generator_function(simulate_cmr, N_super = N_super, J = J,
                                   K = K) |>
  generate_datasets(n_sims)
backend <- SBC_backend_cmdstan_sample(cmdstan_model(here("stan/cjs-rd.stan")),
                                      init = 0.1, chains = chains,
                                      iter_warmup = iter_warmup,
                                      iter_sampling = iter_sampling)
sbc <- compute_SBC(datasets, backend, cores_per_fit = chains, keep_fits = F)
```

Mean time for the slowest of 8 chains with 700 iterations: 0.01 minutes.

```
plot_sbc(sbc, nrow = 2)
```



Multistate Cormack-Jolly-Seber

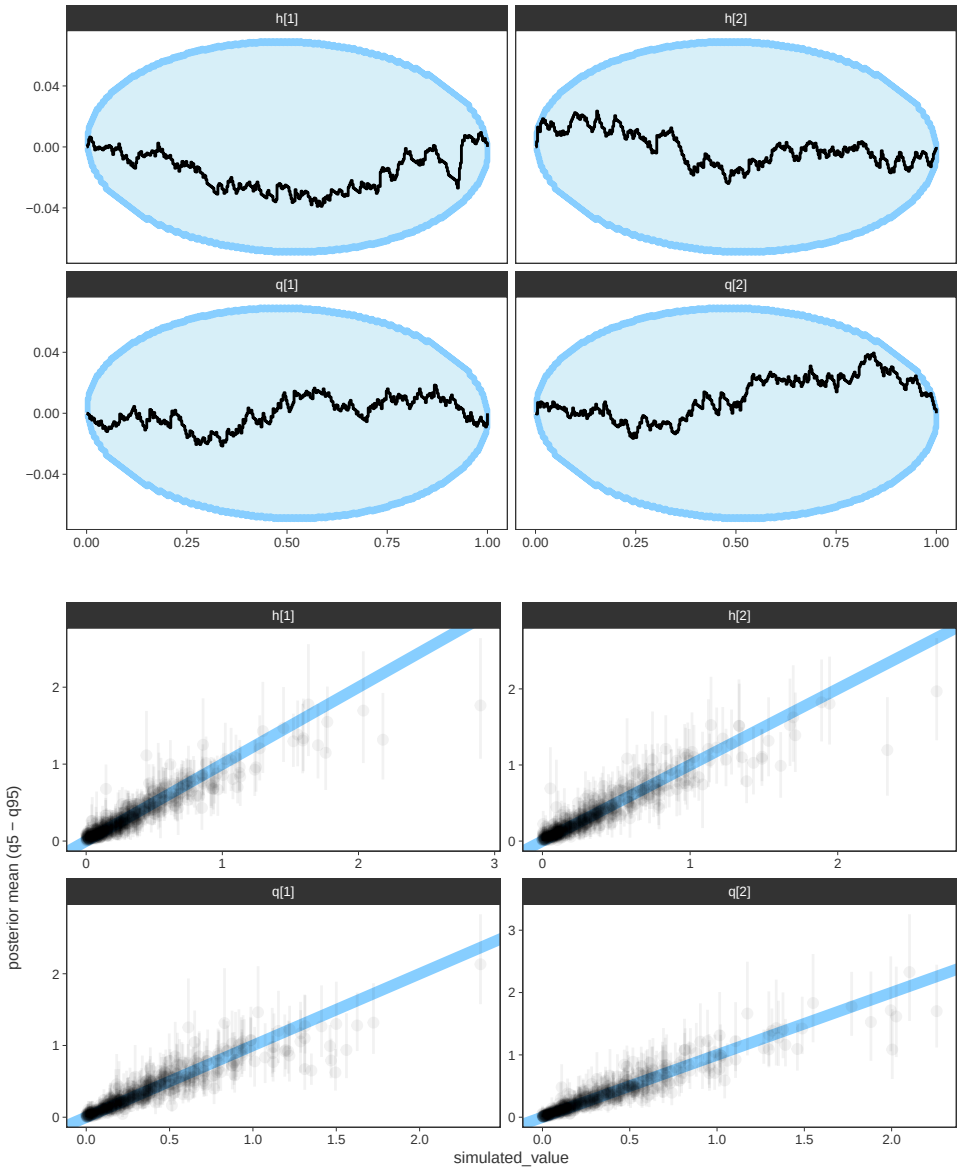
Single survey

```
datasets <- SBC_generator_function(simulate_cmr, N_super = N_super, J = J,
                                   S = S) |>
  generate_datasets(n_sims)
backend <- SBC_backend_cmdstan_sample(cmdstan_model(here("stan/cjs-ms.stan")),
                                      init = 0.1, chains = chains,
                                      iter_warmup = iter_warmup,
                                      iter_sampling = iter_sampling)
sbc <- compute_SBC(datasets, backend, cores_per_fit = chains, keep_fits = F)
```

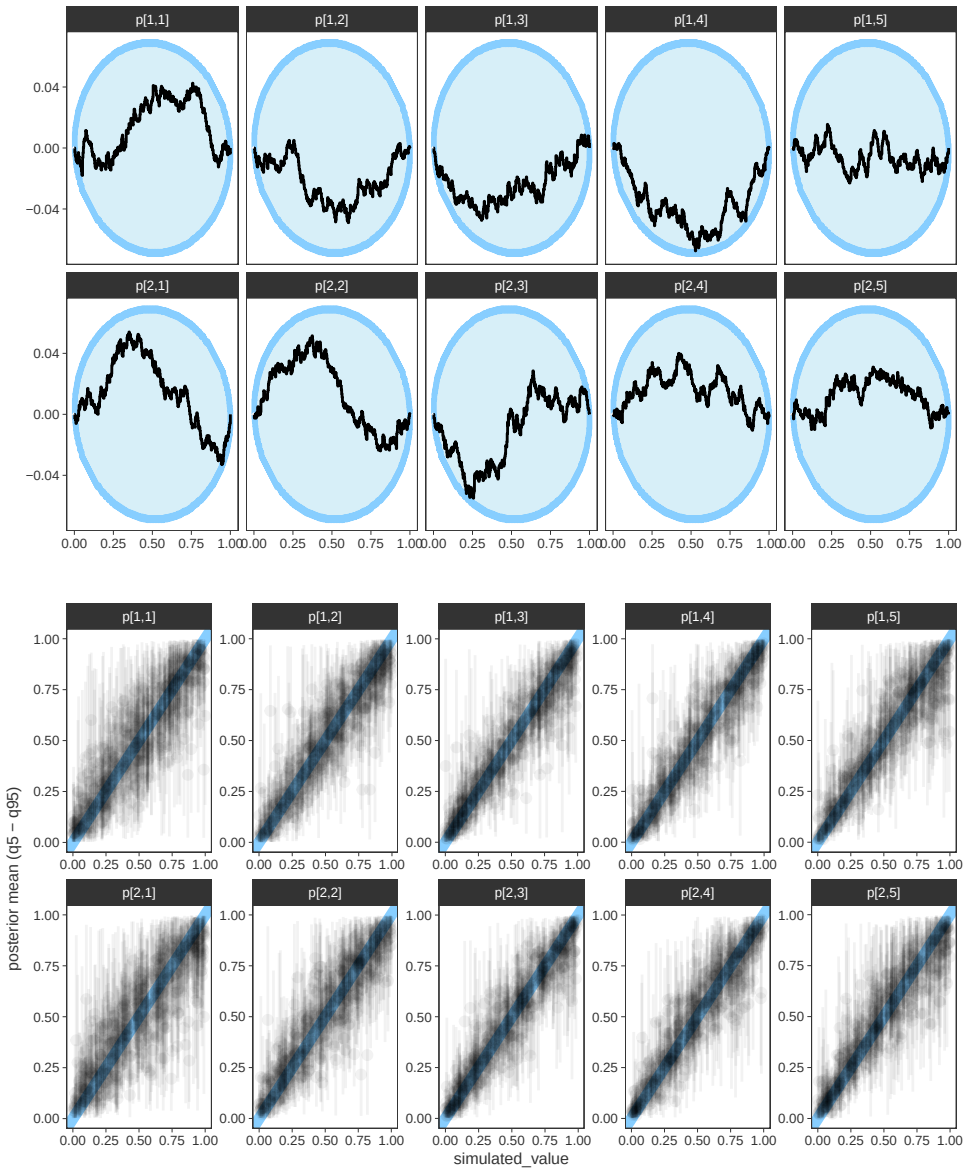
Mean time for the slowest of 8 chains with 700 iterations: 0.027 minutes.

> **Warning**
>
> Single survey multistate models with state-specific detection probabilities suffer from parameter identifiability issues in that they are confounded with the transition rates (Hollanders and Royle 2022). Notice the difference in parameter estimates with the robust design.

```
plot_sbc(sbc, c(str_c("h[", 1:S, "]"), str_c("q[", 1:(S * (S - 1)), "]")))
```



```
plot_sbc(sbc, flatten_chr(map(1:S, ~(str_c("p[", .x, ",", 1:(J - 1), "]")))),
         ncol = J - 1)
```

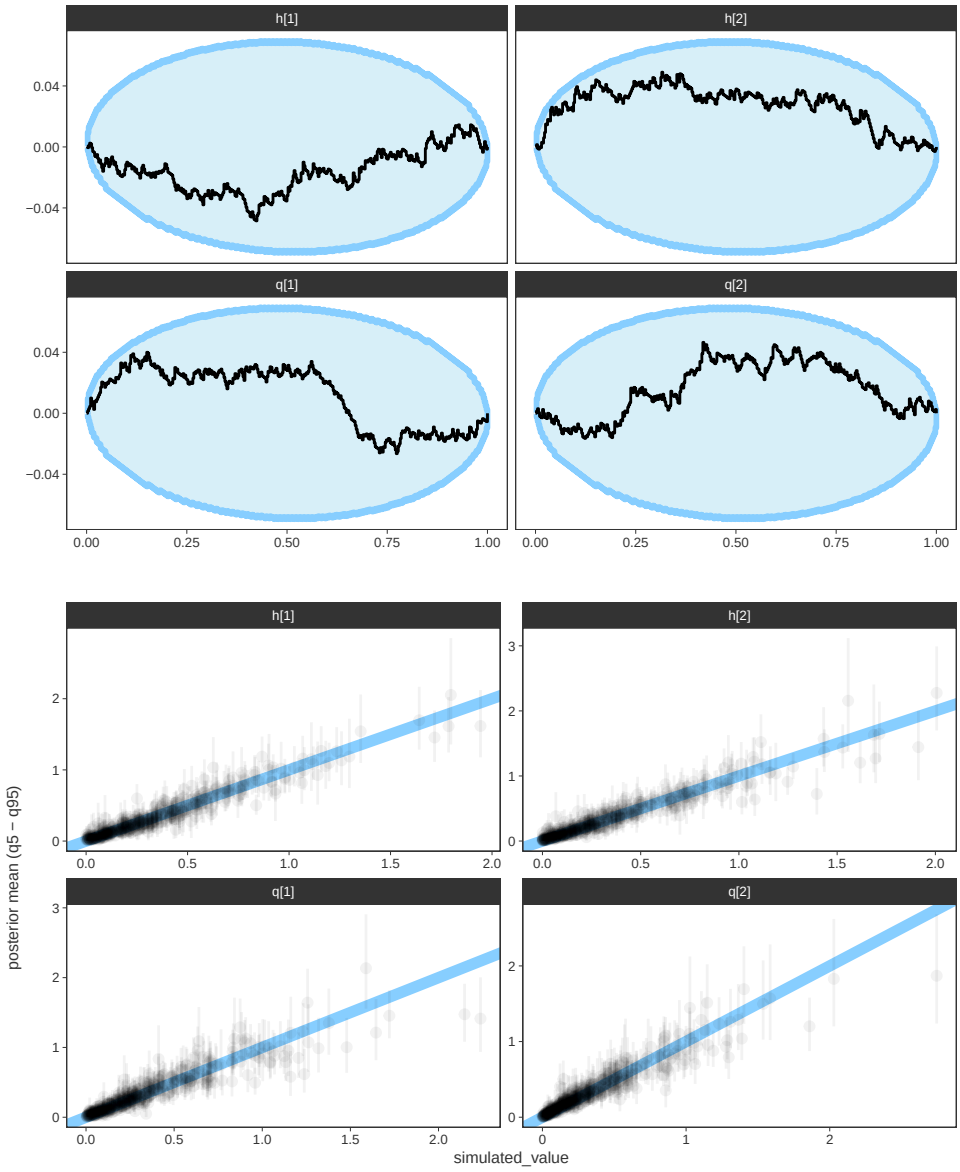


Robust design

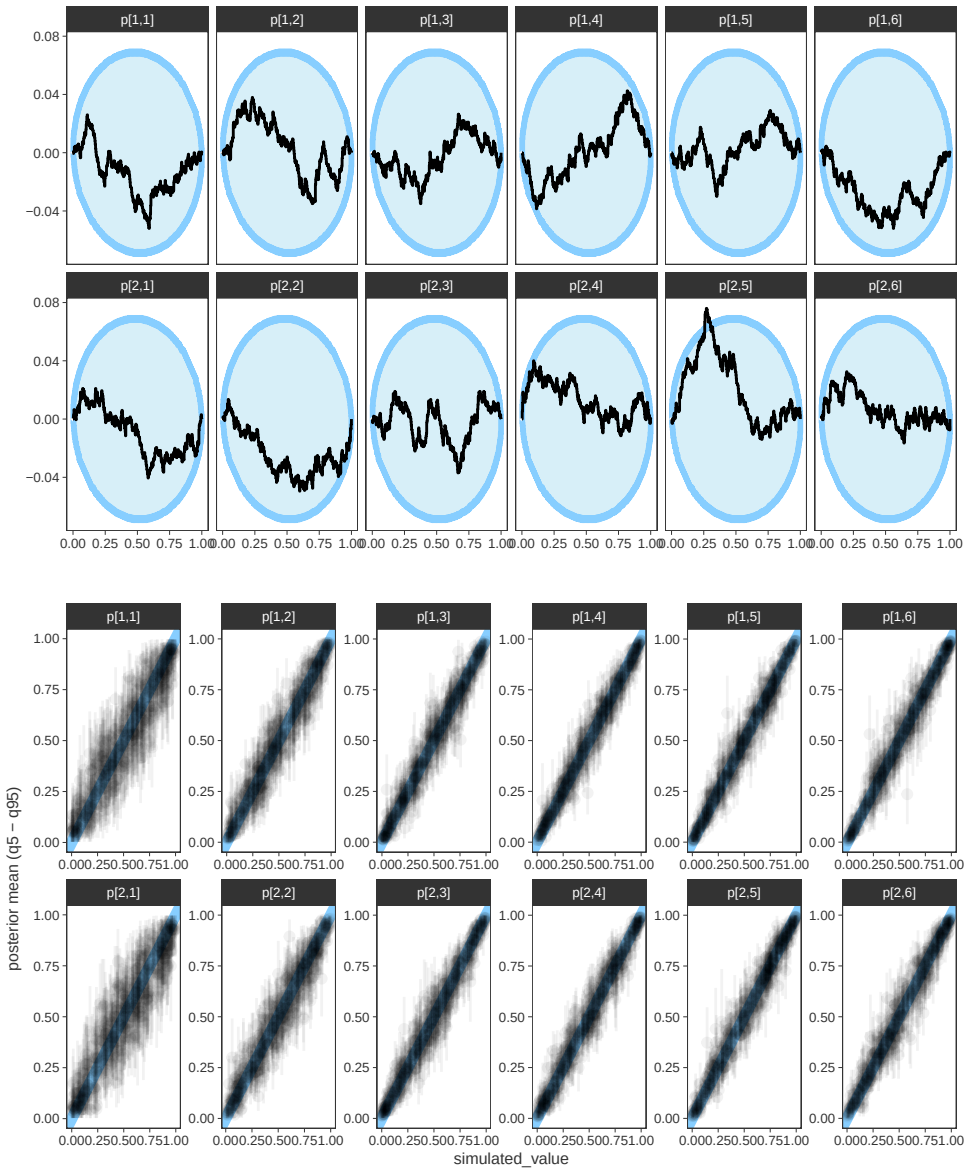
```
datasets <- SBC_generator_function(simulate_cmr, N_super = N_super, J = J,
                                   K = K, S = S) |>
  generate_datasets(n_sims)
backend <- SBC_backend_cmdstan_sample(cmdstan_model(here("stan/cjs-ms-rd.stan")),
                                      init = 0.1, chains = chains,
                                      iter_warmup = iter_warmup,
                                      iter_sampling = iter_sampling)
sbc <- compute_SBC(datasets, backend, cores_per_fit = chains, keep_fits = F)
```

Mean time for the slowest of 8 chains with 700 iterations: 0.026 minutes.

```
plot_sbc(sbc, c(str_c("h[", 1:S, "]"), str_c("q[", 1:(S * (S - 1)), "]")))
```



```
plot_sbc(sbc, flatten_chr(map(1:S, ~(str_c("p[", .x, ",", 1:J, "]")))),
         ncol = J)
```



Multievent Cormack-Jolly-Seber

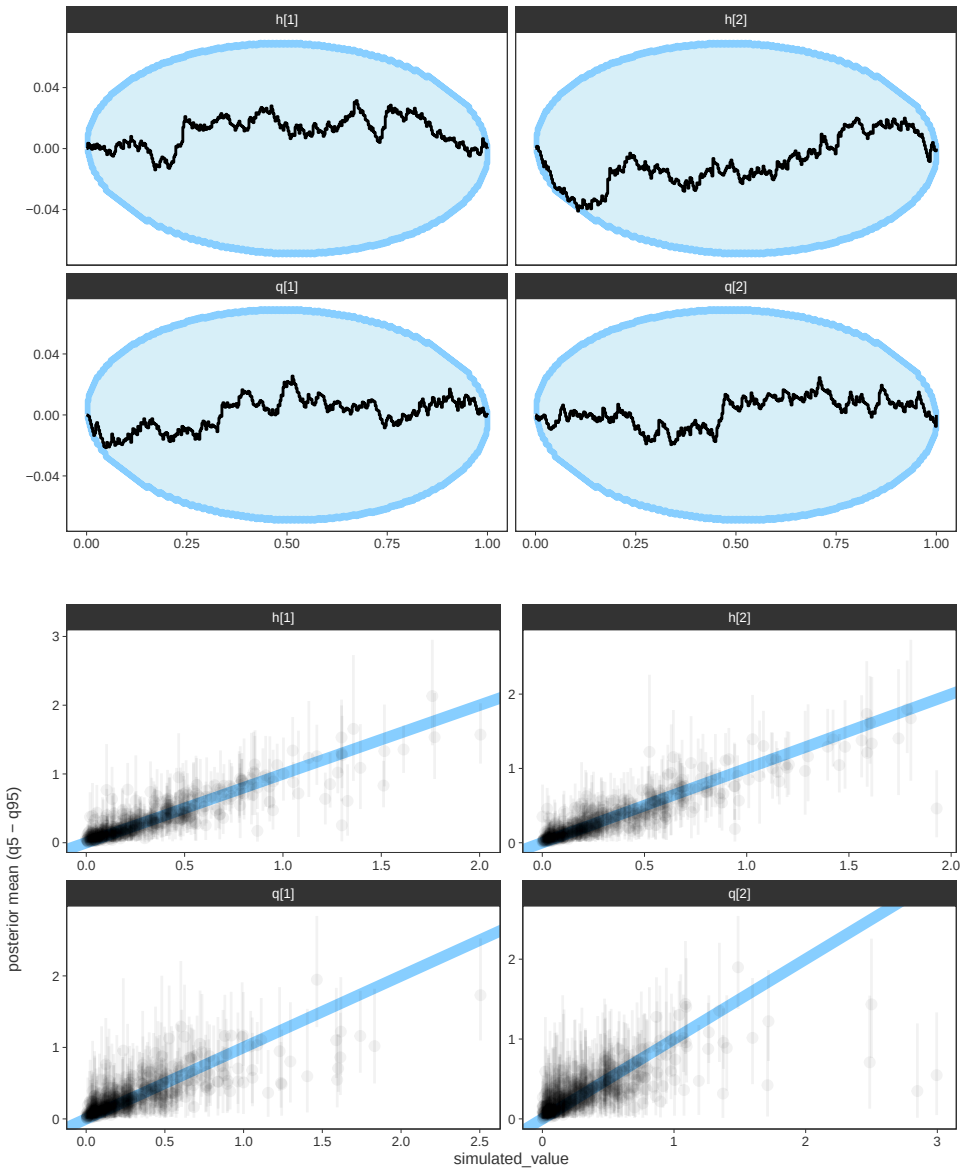
While running SBC for multievent models, lower bidiagonal stochastic matrices for the event matrix $\boldsymbol{E}$, with states-specific assignment probabilities $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_{S-1})$ for states $s \forall \{2, \ldots, S\}$, with state $s-1$ having state misclassification probability $1 - \delta_{s+1}$.

Single survey

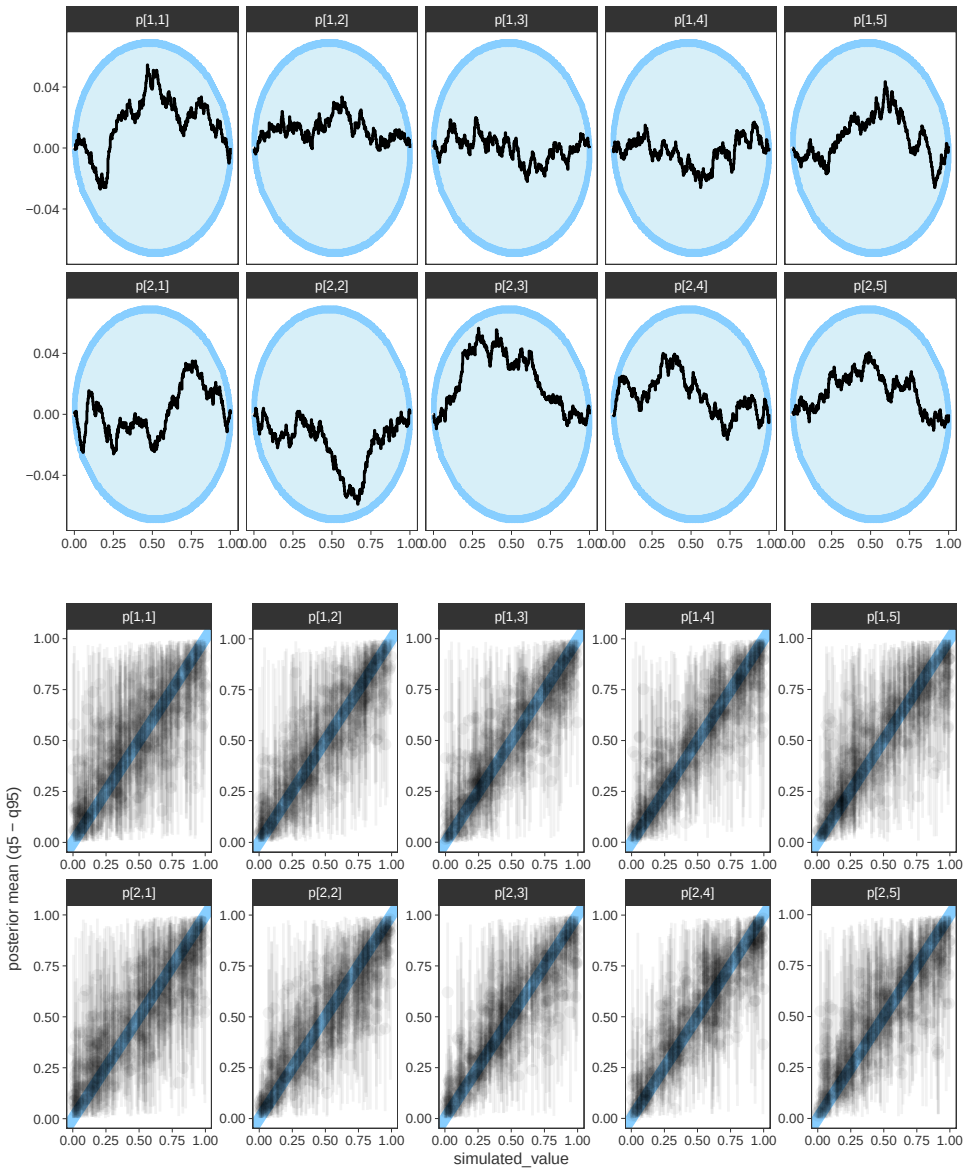
```
datasets <- SBC_generator_function(simulate_cmr, N_super = N_super, J = J,
                                   S = S, ME = T) |>
  generate_datasets(n_sims)
backend <- SBC_backend_cmdstan_sample(cmdstan_model(here("stan/cjs-me.stan")),
                                      init = 0.1, chains = chains,
                                      iter_warmup = iter_warmup,
                                      iter_sampling = iter_sampling)
sbc <- compute_SBC(datasets, backend, cores_per_fit = chains, keep_fits = F)
```

Mean time for the slowest of 8 chains with 700 iterations: 0.168 minutes.

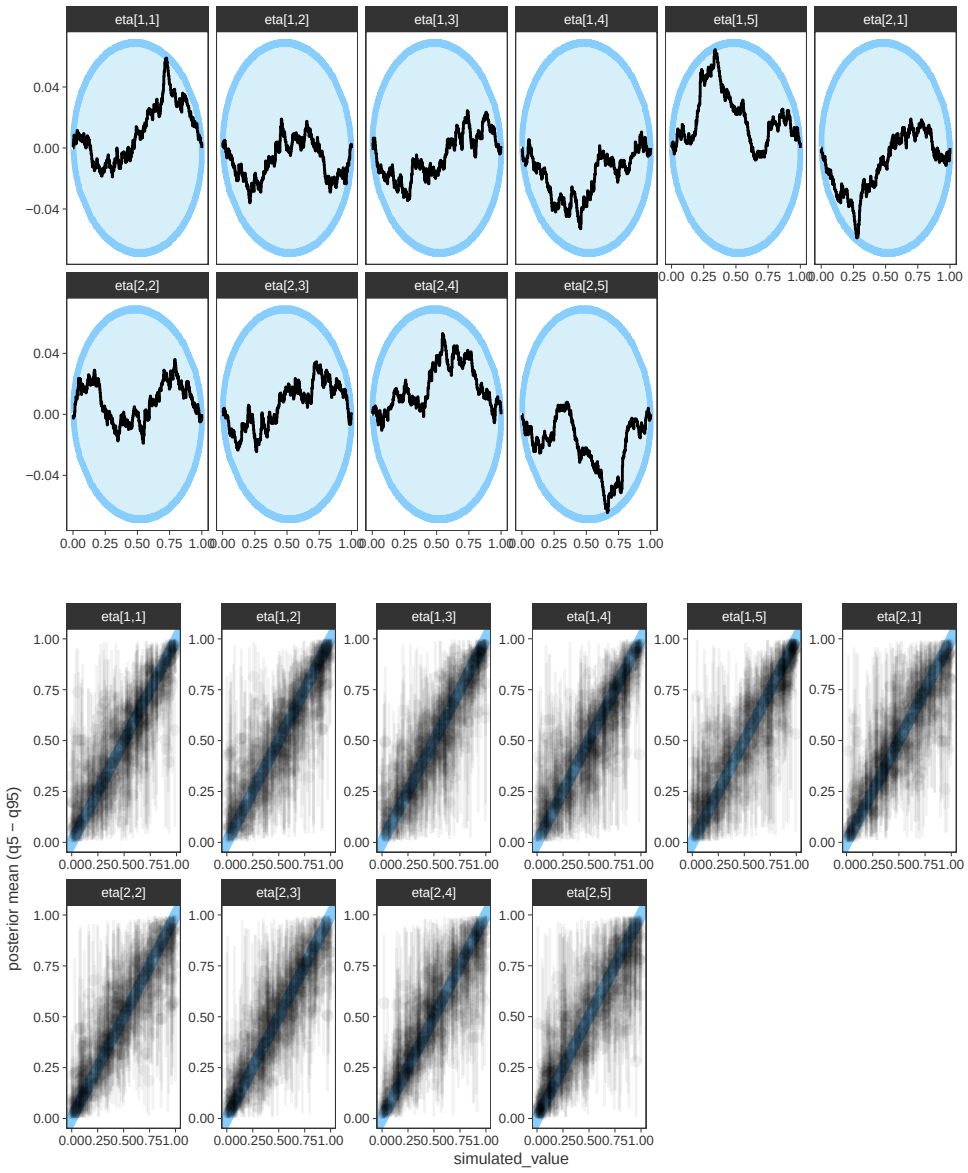
```
plot_sbc(sbc, c(str_c("h[", 1:S, "]"),
                str_c("q[", 1:(S * (S - 1)), "]"),
                str_c("delta[", 1:(S - 1), "]")))
```



```
plot_sbc(sbc, flatten_chr(map(1:S, ~(str_c("p[", .x, ",", 1:J, "]")))),
         ncol = J - 1)
```



```
plot_sbc(sbc, flatten_chr(map(1:S, ~(str_c("eta[", .x, ",", 1:(J - 1), "]")))),
         ncol = J)
```

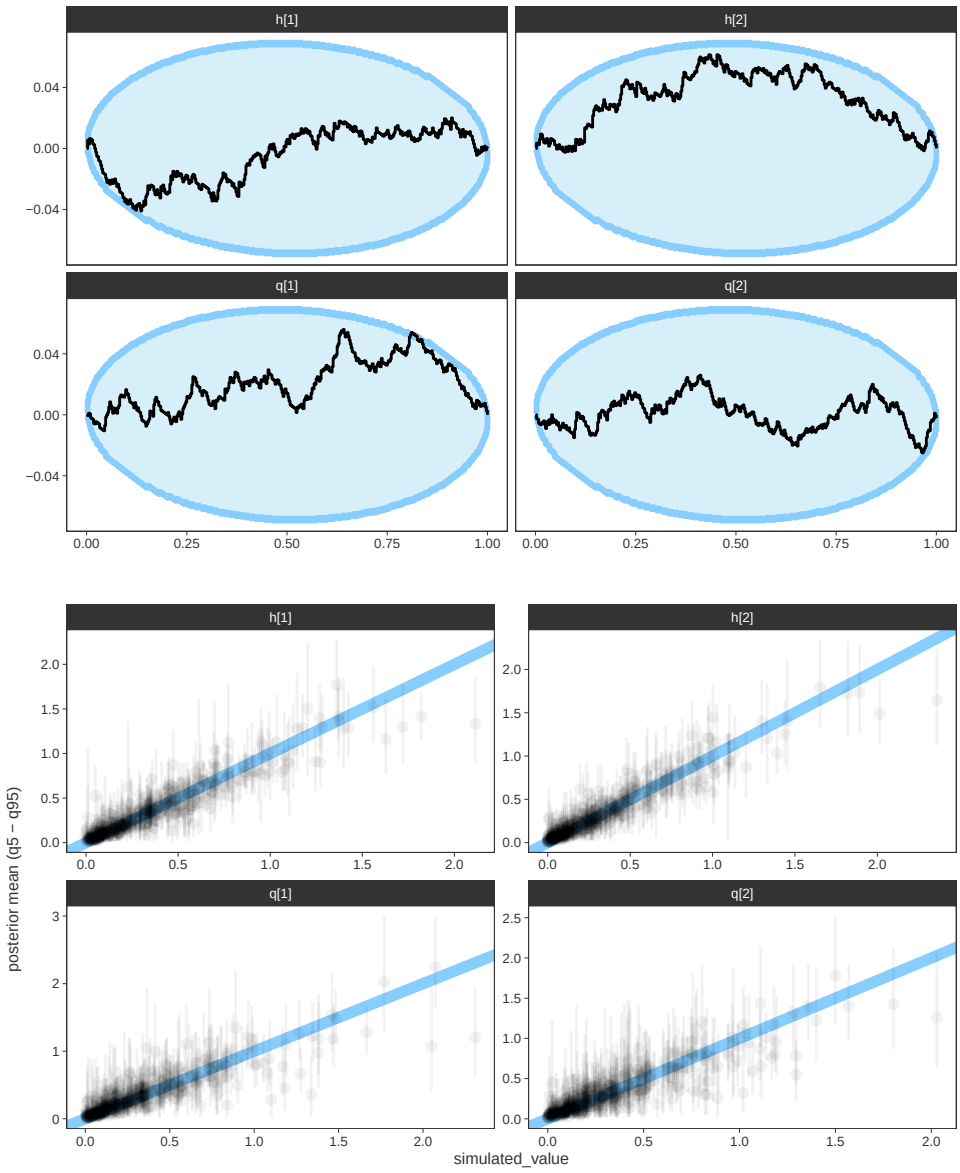


Robust design

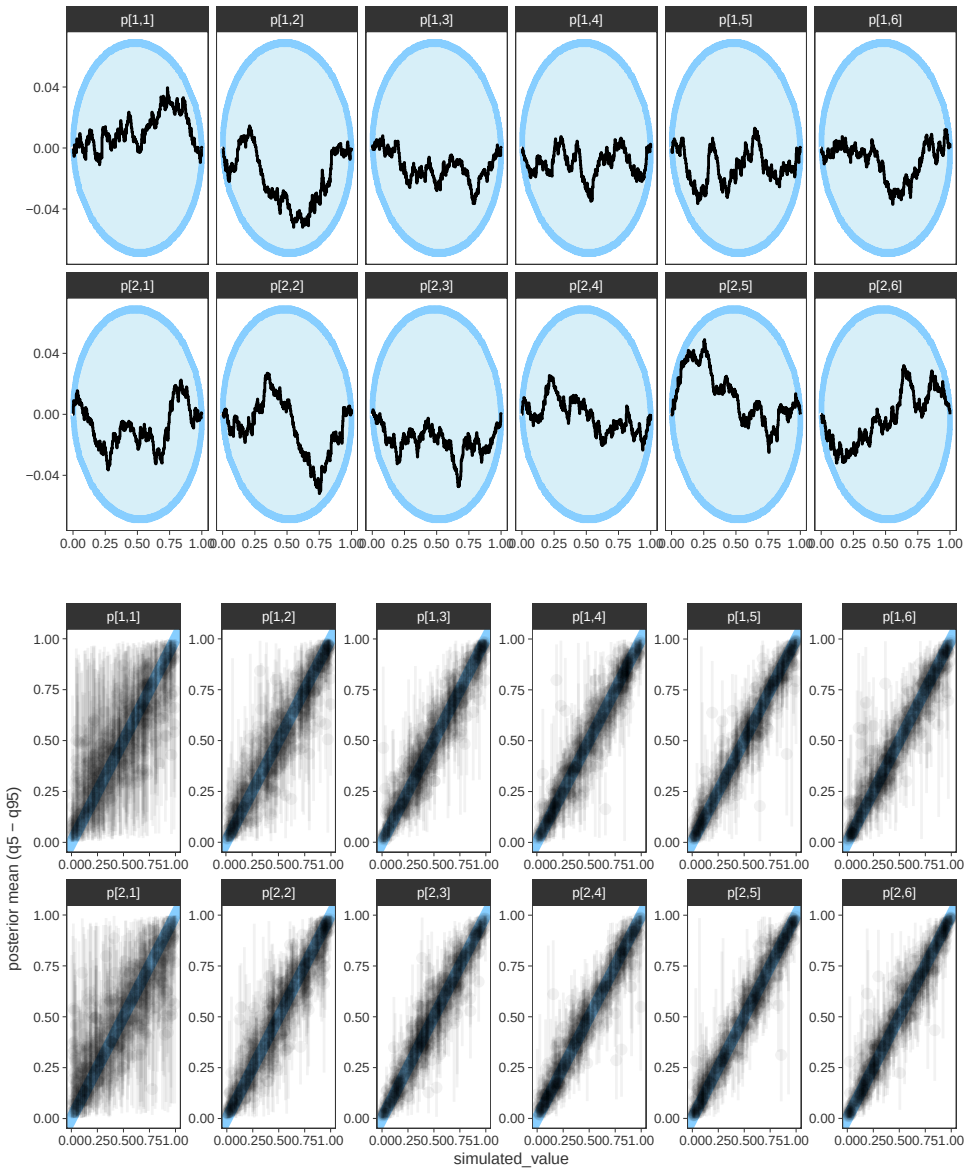
```
datasets <- SBC_generator_function(simulate_cmr, N_super = N_super, J = J,
                                   K = K, S = S, ME = T) |>
  generate_datasets(n_sims)
backend <- SBC_backend_cmdstan_sample(cmdstan_model(here("stan/cjs-me-rd.stan")),
                                      init = 0.1, chains = chains,
                                      iter_warmup = iter_warmup,
                                      iter_sampling = iter_sampling)
sbc <- compute_SBC(datasets, backend, cores_per_fit = chains, keep_fits = F)
```

Mean time for the slowest of 8 chains with 700 iterations: 0.14 minutes.

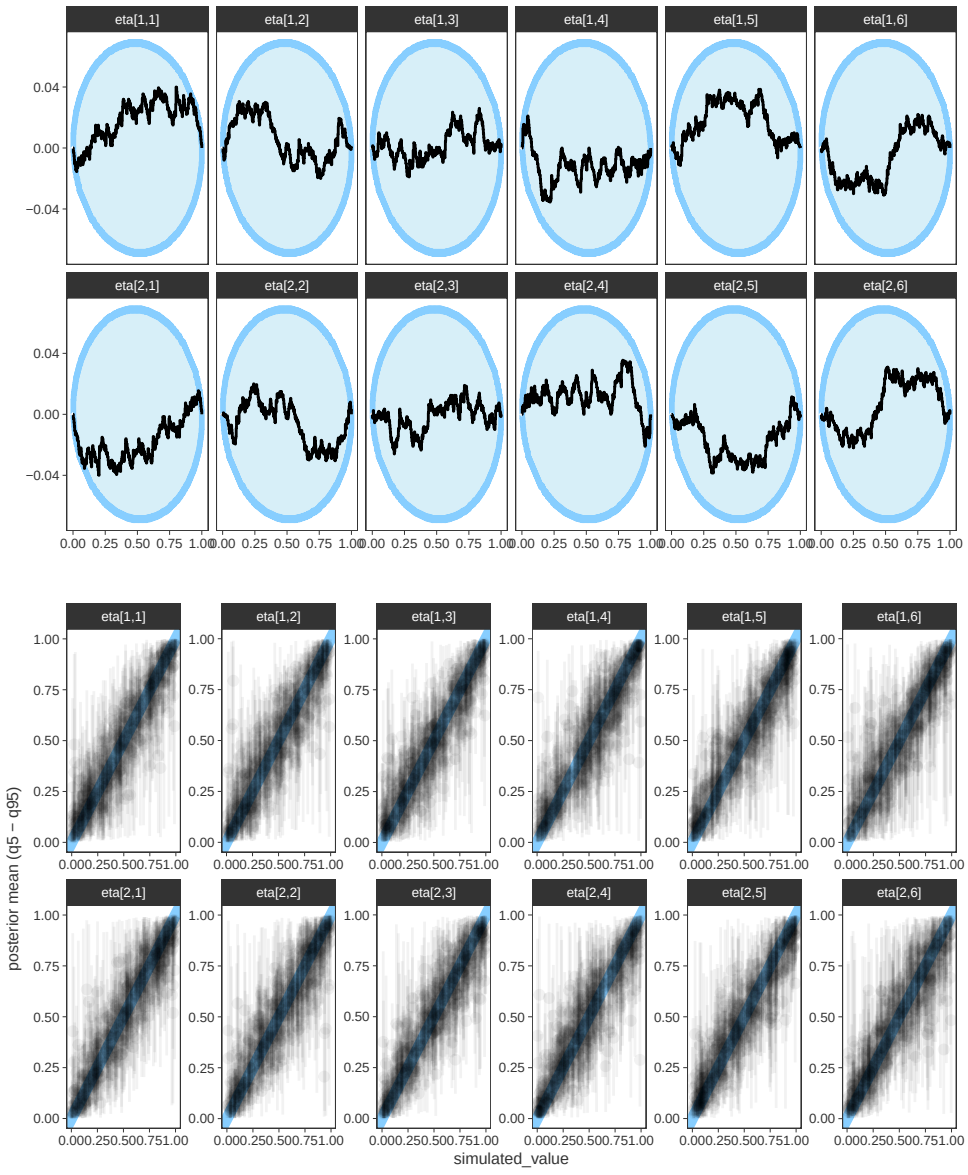
```
plot_sbc(sbc, c(str_c("h[", 1:S, "]"), str_c("q[", 1:(S * (S - 1)), "]"),
                str_c("delta[", 1:(S - 1), "]")))
```



```
plot_sbc(sbc, flatten_chr(map(1:S, ~(str_c("p[", .x, ",", 1:J, "]")))),
         ncol = J)
```



```
plot_sbc(sbc, flatten_chr(map(1:S, ~(str_c("eta[", .x, ",", 1:J, "]")))),
         ncol = J)
```



Jolly-Seber

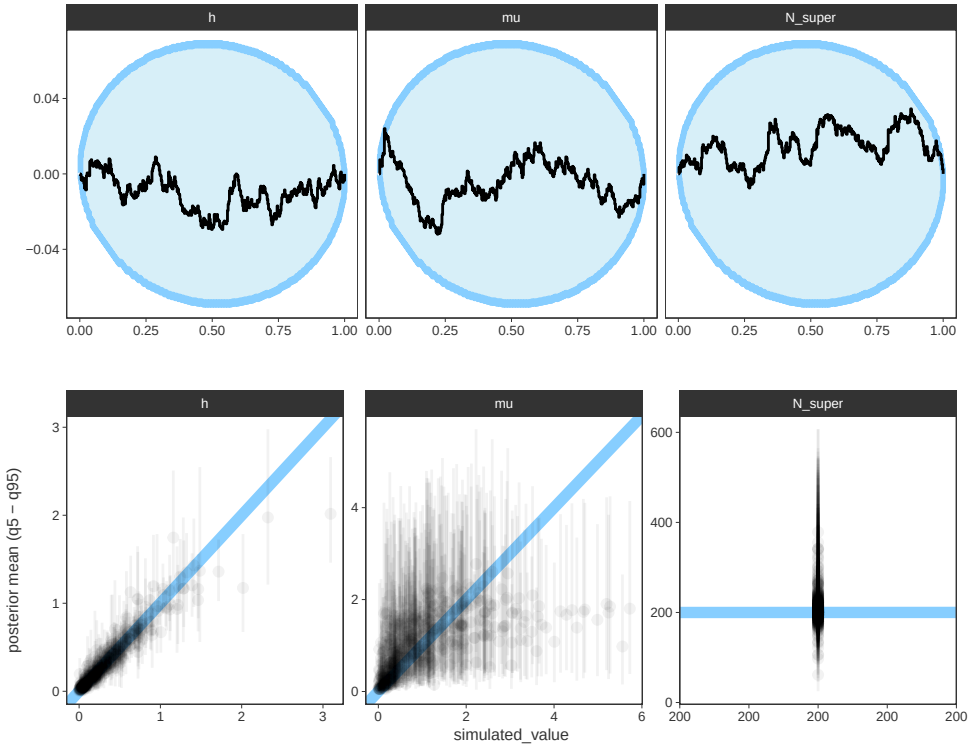
Single survey

```
datasets <- SBC_generator_function(simulate_cmr, N_super = N_super, J = J,
                                   JS = T) |>
  generate_datasets(n_sims)
backend <- SBC_backend_cmdstan_sample(cmdstan_model(here("stan/js.stan")),
                                      init = 0.1, chains = chains,
                                      iter_warmup = iter_warmup,
                                      iter_sampling = iter_sampling)
sbc <- compute_SBC(datasets, backend, cores_per_fit = chains, keep_fits = F)
```

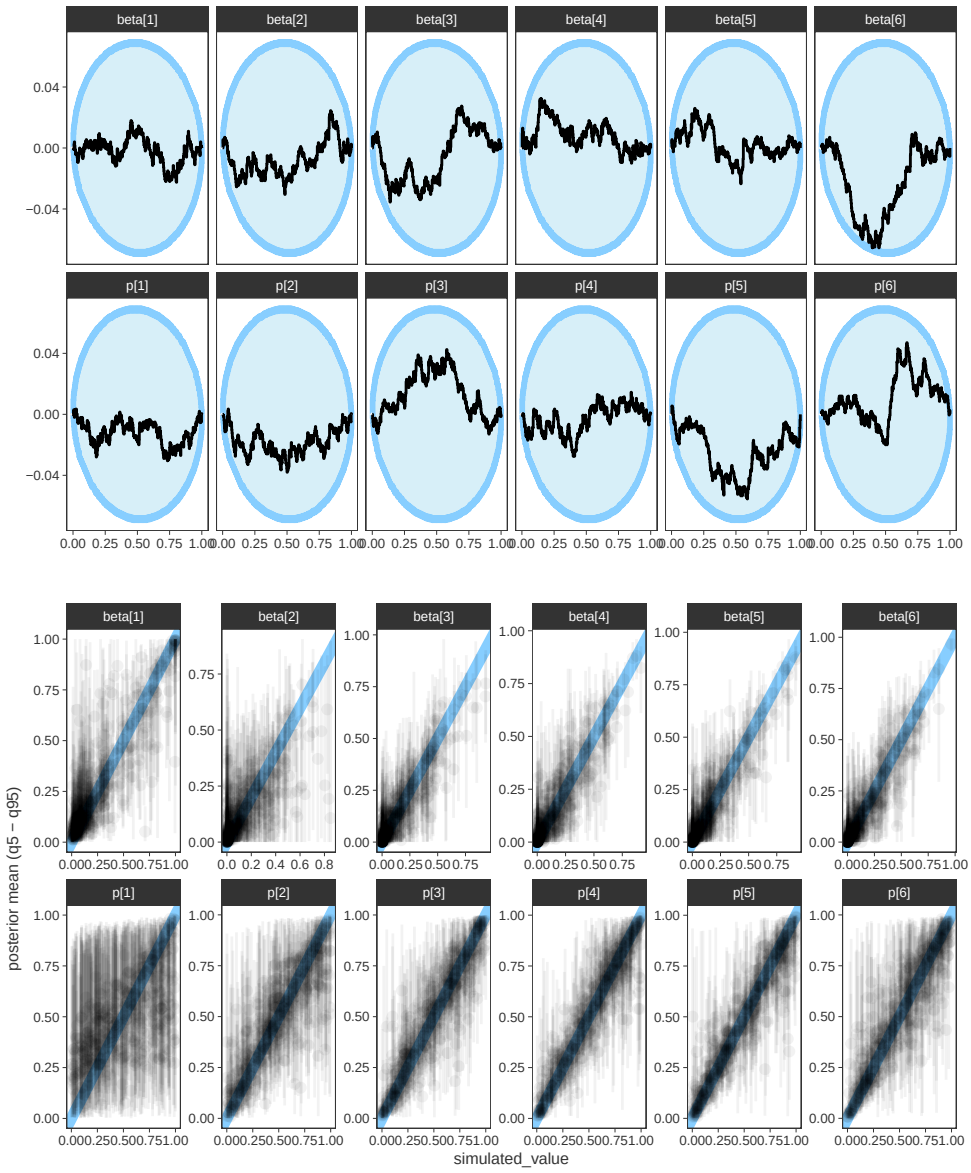
Mean time for the slowest of 8 chains with 700 iterations: 0.122 minutes.

> **Warning**
>
> Single-survey Jolly-Seber models with time-varying entry and detection probabilities suffer from parameter identifiability issues (Schwarz and Arnason 2008). Notice the difference in these parameter estimates with the robust design version.

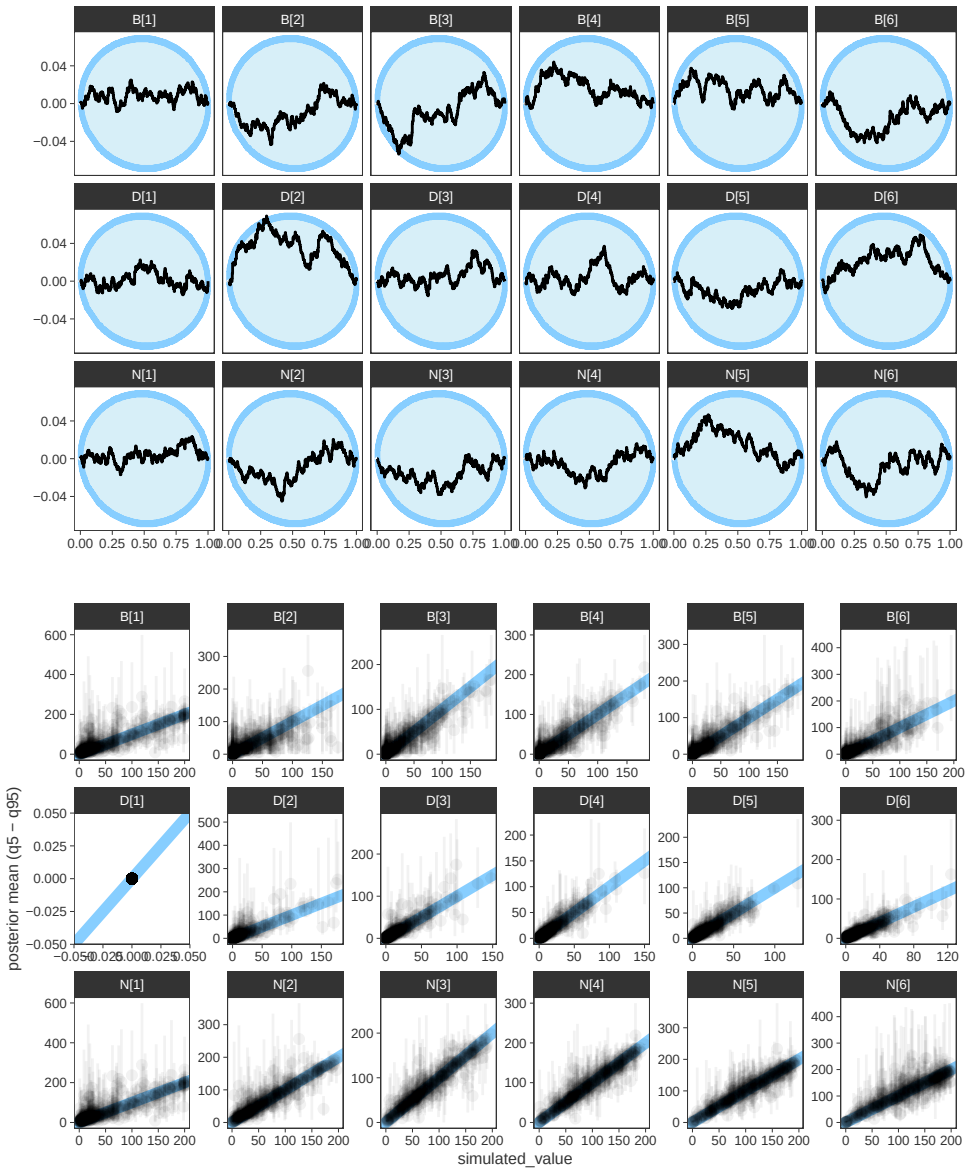
```
plot_sbc(sbc, c("h", "mu", "N_super"))
```



```
plot_sbc(sbc, flatten_chr(map(c("beta", "p"), ~(str_c(., "[", 1:J, "]")))),
         ncol = J)
```



```
plot_sbc(sbc, flatten_chr(map(c("N", "B", "D"), ~(str_c(., "[", 1:J, "]")))),
         ncol = J)
```

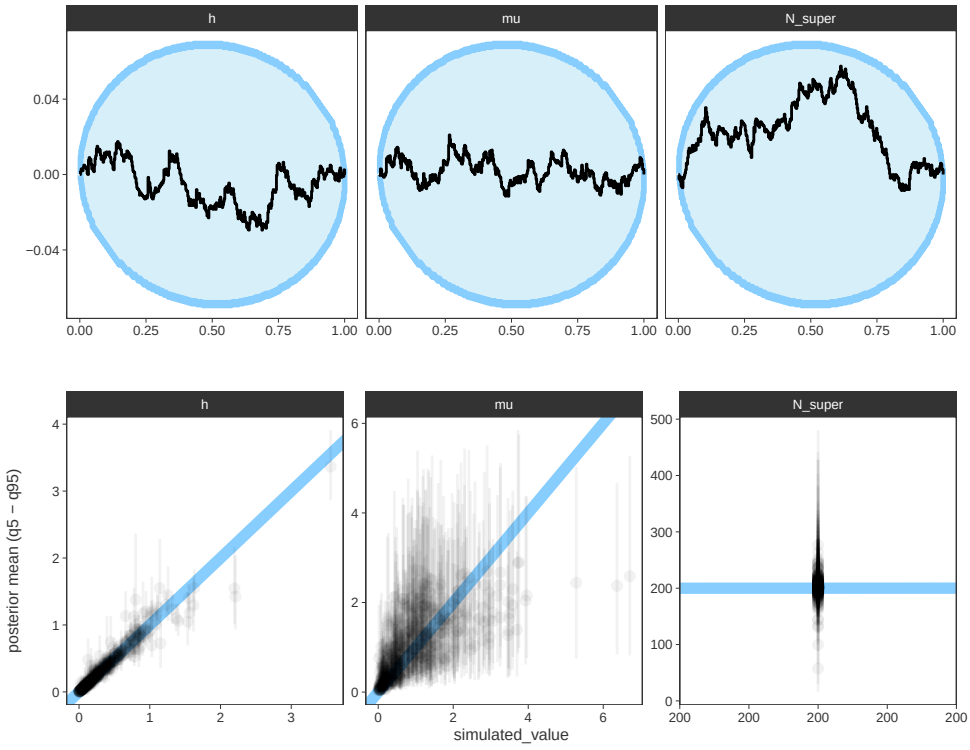


Robust design

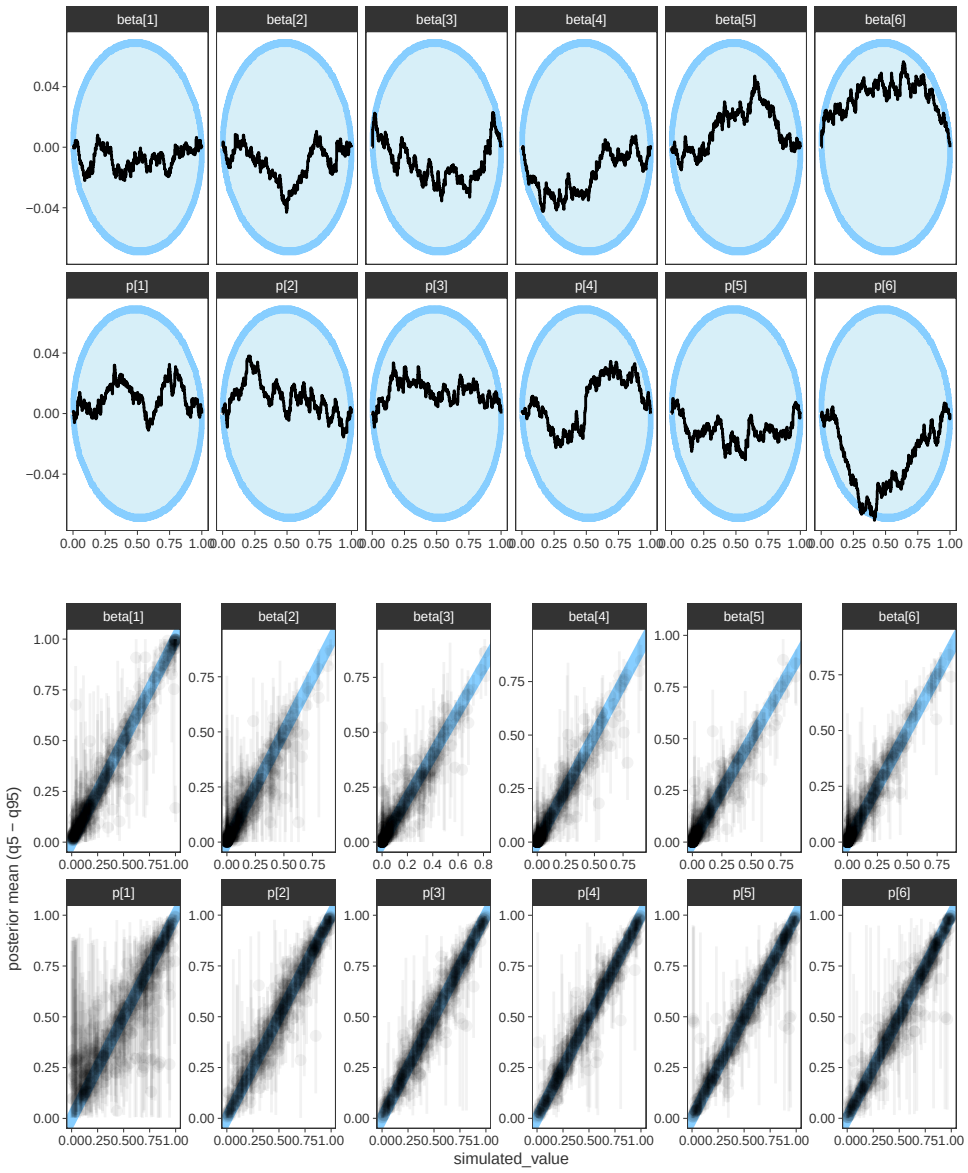
```
datasets <- SBC_generator_function(simulate_cmr, N_super = N_super, J = J,
                                   K = K, JS = T) |>
  generate_datasets(n_sims)
backend <- SBC_backend_cmdstan_sample(cmdstan_model(here("stan/js-rd.stan")),
                                      init = 0.1, chains = chains,
                                      iter_warmup = iter_warmup,
                                      iter_sampling = iter_sampling)
sbc <- compute_SBC(datasets, backend, cores_per_fit = chains, keep_fits = F)
```

Mean time for the slowest of 8 chains with 700 iterations: 0.254 minutes.

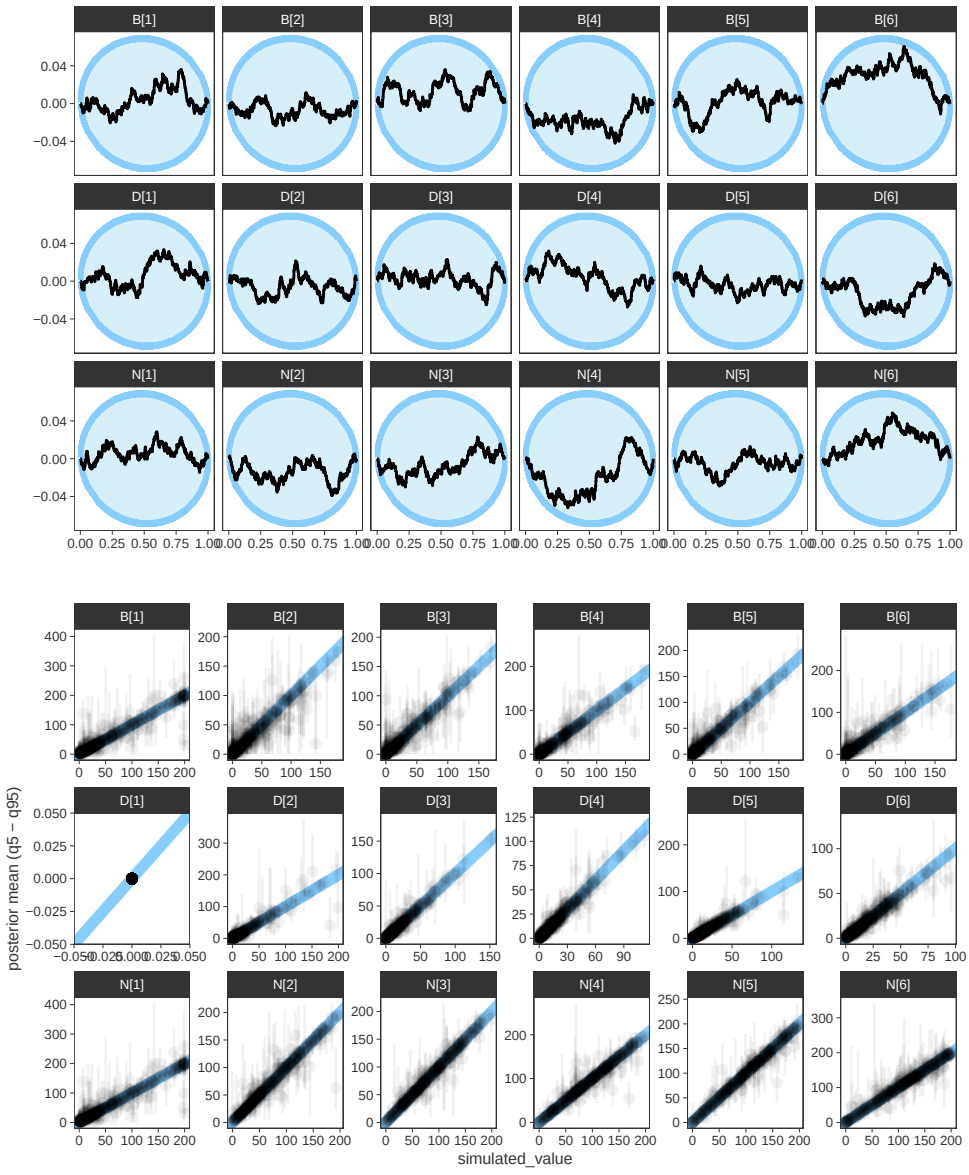
```
plot_sbc(sbc, c("h", "mu", "N_super"))
```



```
plot_sbc(sbc, flatten_chr(map(c("beta", "p"), ~(str_c(., "[", 1:J, "]")))),
         ncol = J)
```



```
plot_sbc(sbc, flatten_chr(map(c("N", "B", "D"), ~(str_c(., "[", 1:J, "]")))),
         ncol = J)
```



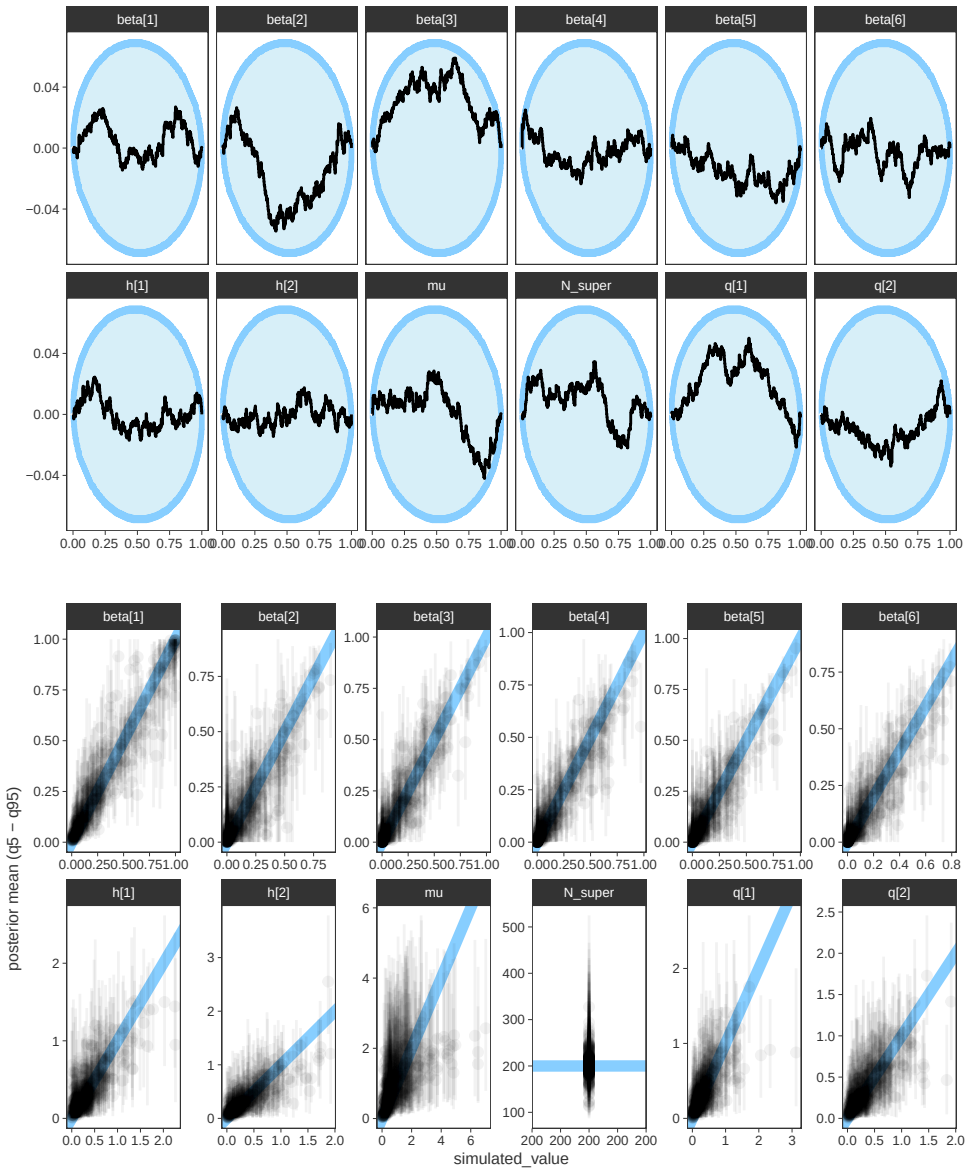
Multistate Jolly-Seber

Single survey

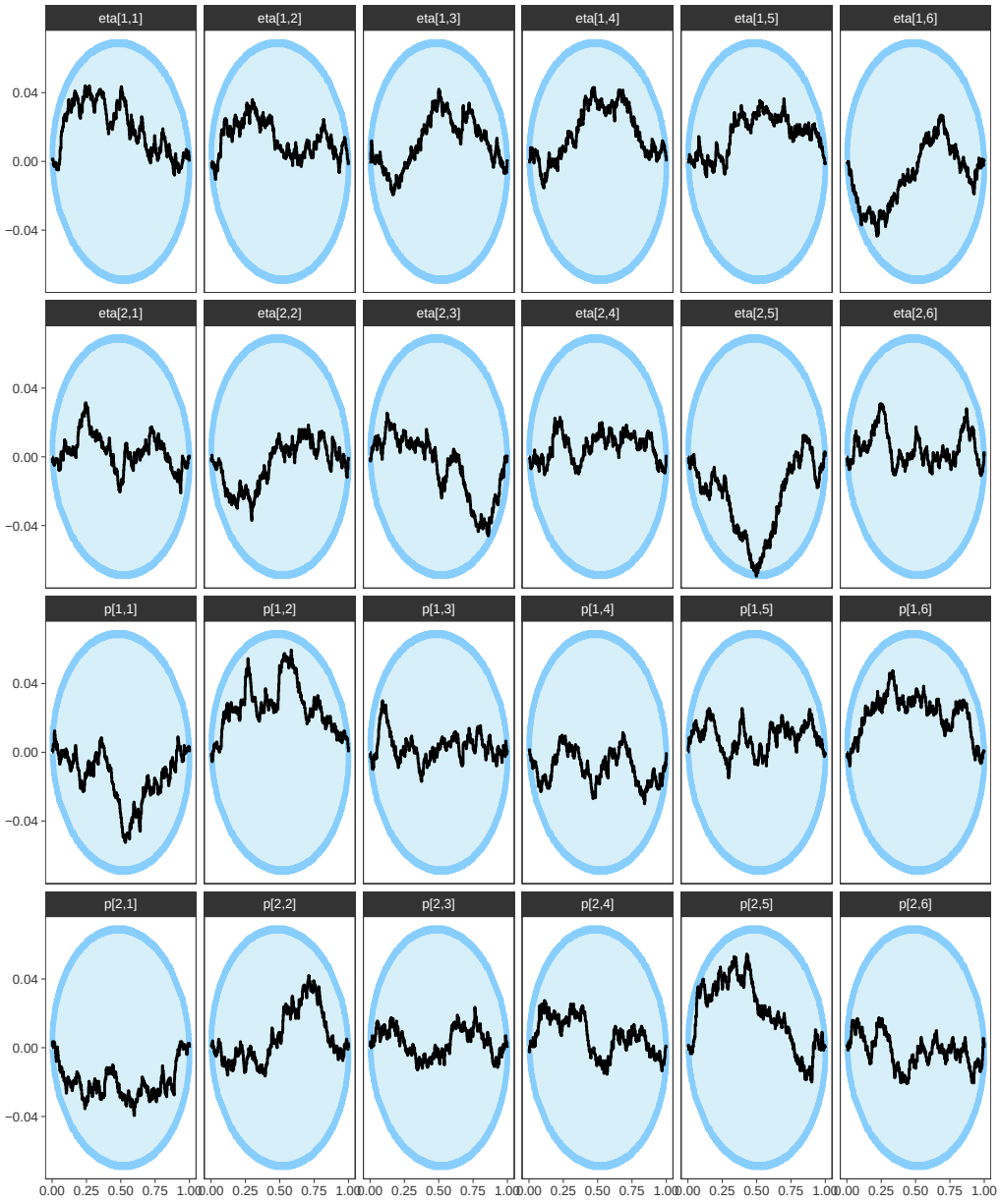
```
datasets <- SBC_generator_function(simulate_cmr, N_super = N_super, J = J,
                                   S = 3, JS = T) |>
  generate_datasets(n_sims)
backend <- SBC_backend_cmdstan_sample(cmdstan_model(here("stan/js-ms.stan")),
                                      init = 0.1, chains = chains,
                                      iter_warmup = iter_warmup,
                                      iter_sampling = iter_sampling)
sbc <- compute_SBC(datasets, backend, cores_per_fit = chains, keep_fits = F)
```

Mean time for the slowest of 8 chains with 700 iterations: 0.908 minutes.

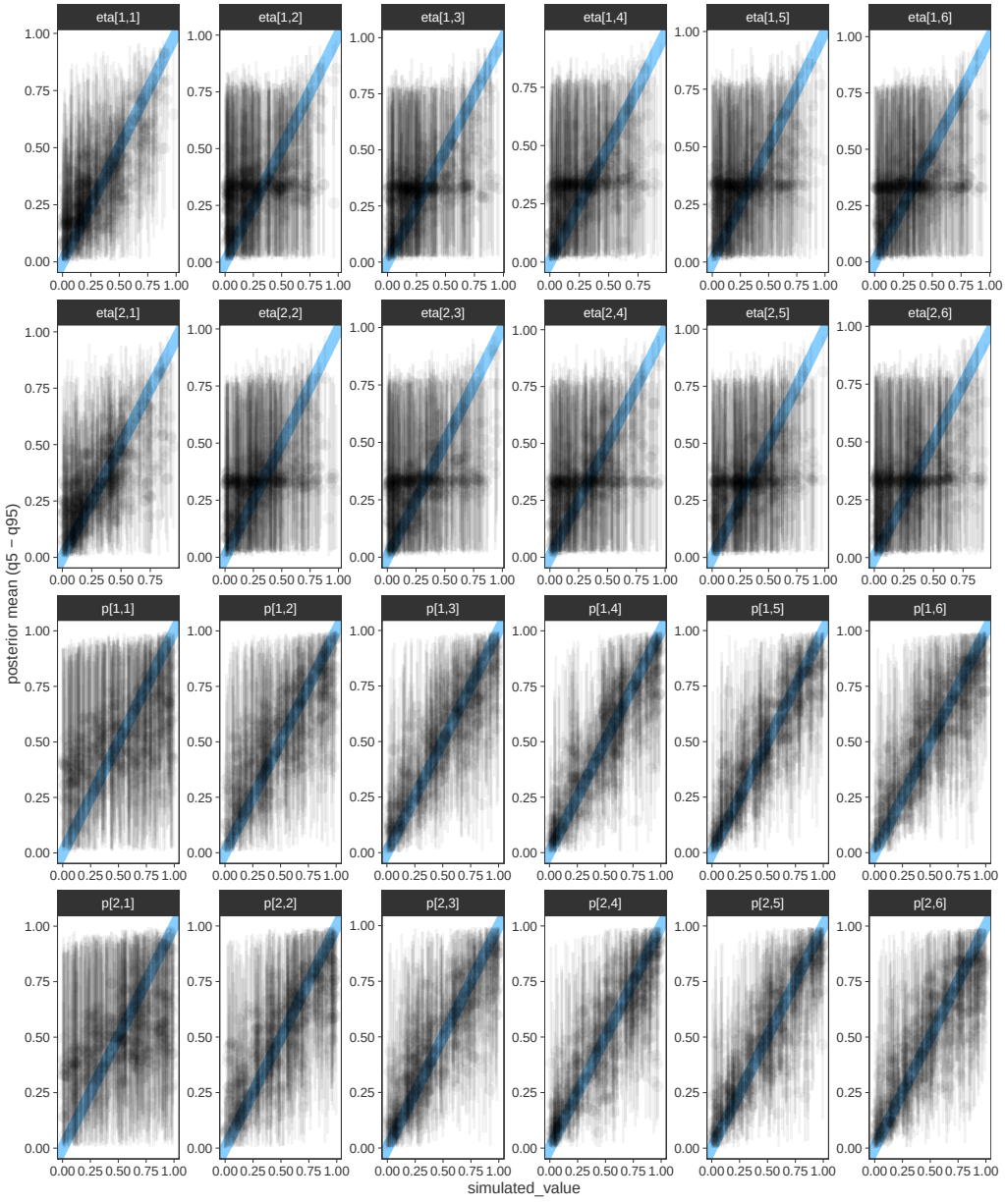
```
plot_sbc(sbc, c(str_c("h[", 1:S, "]"), str_c("q[", 1:(S * (S - 1)), "]"),
                str_c("beta[", 1:J, "]"), "mu", "N_super"), ncol = J)
```



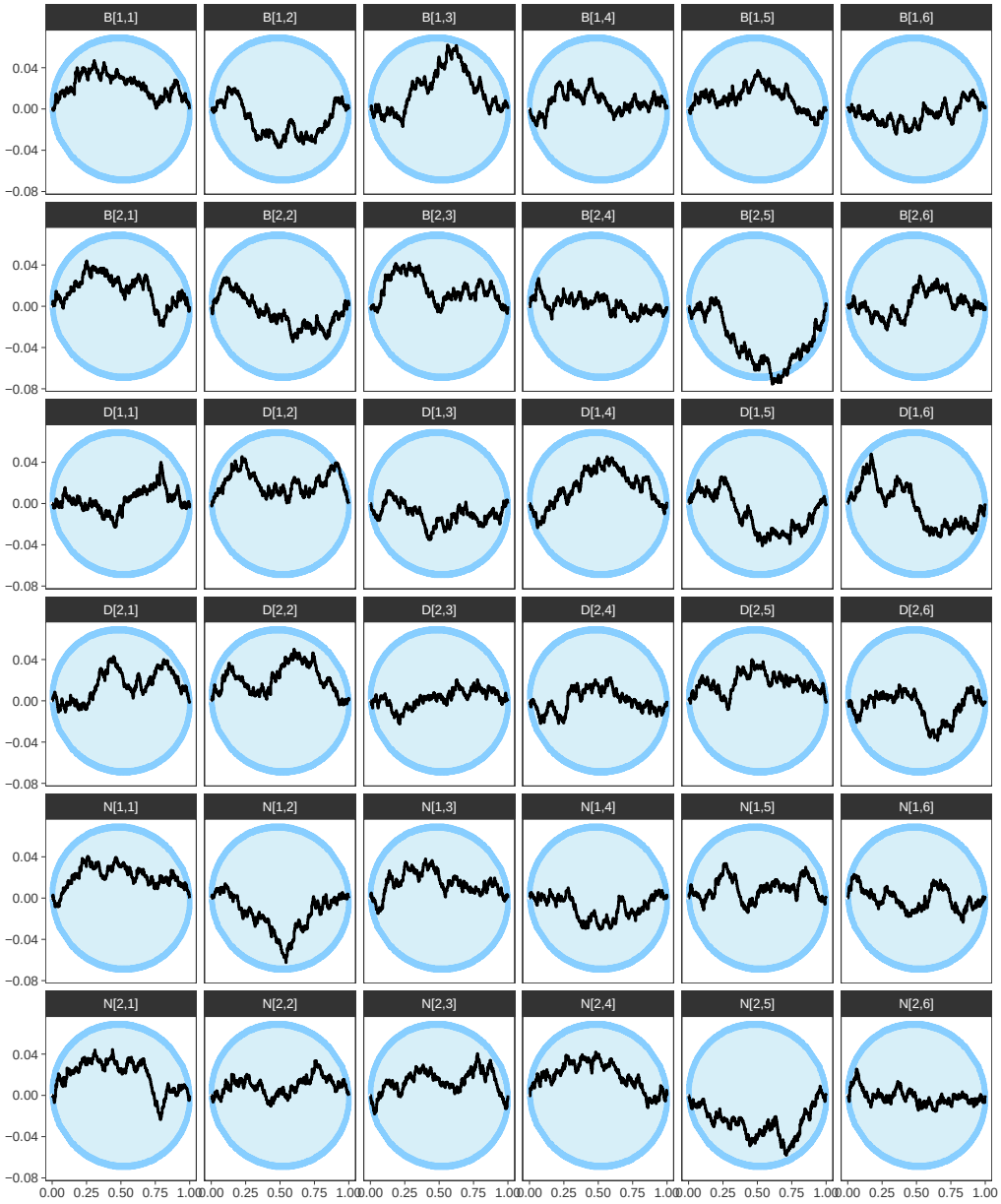
```
params <- map(c("p", "eta", "N", "B", "D"), \(x)
              flatten_chr(map(1:S, \(s) str_c(x, "[", s, ",", 1:J, "]"))))
plot_ecdf_diff(sbc, flatten_chr(params[1:2])) +
  facet_wrap(~ group, ncol = J) +
  theme(legend.position = "none")
```



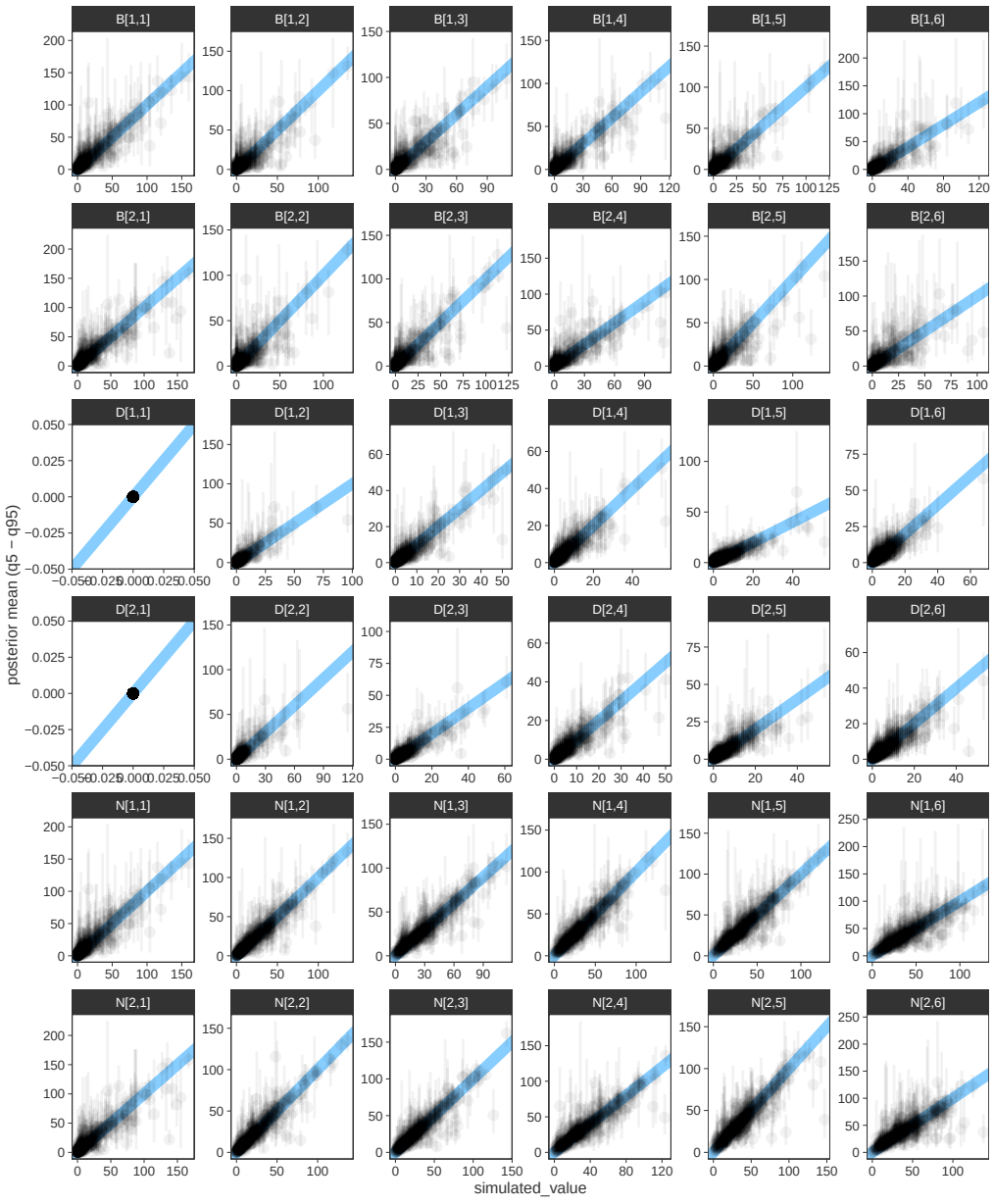
```
plot_sim_estimated(sbc, flatten_chr(params[1:2])) +
  facet_wrap(~ variable, ncol = J, scales = "free")
```



```
plot_ecdf_diff(sbc, flatten_chr(params[3:5])) +
  facet_wrap(~ variable, ncol = J) +
  theme(legend.position = "none")
```



```
plot_sim_estimated(sbc, flatten_chr(params[3:5])) +
  facet_wrap(~ variable, ncol = J, scales = "free")
```

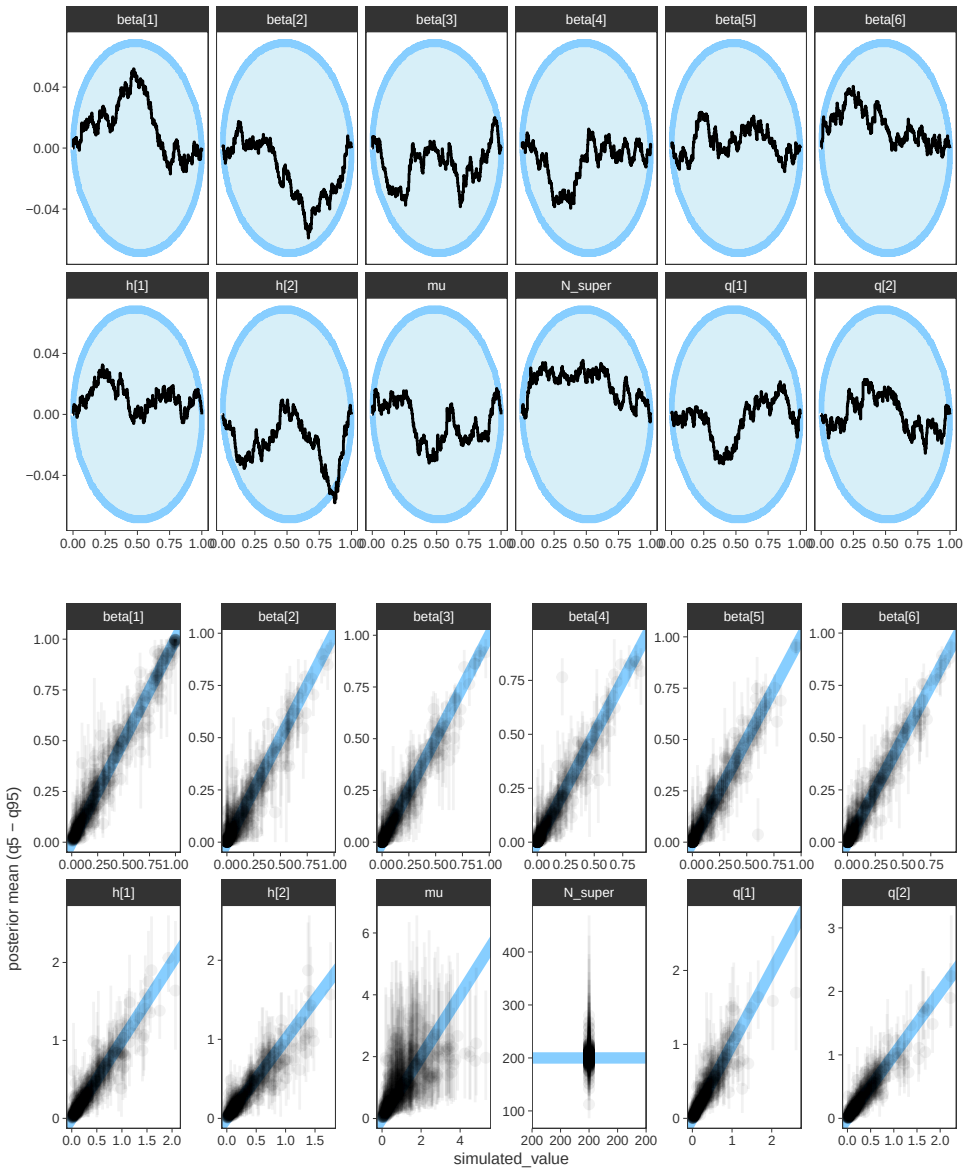


Robust design

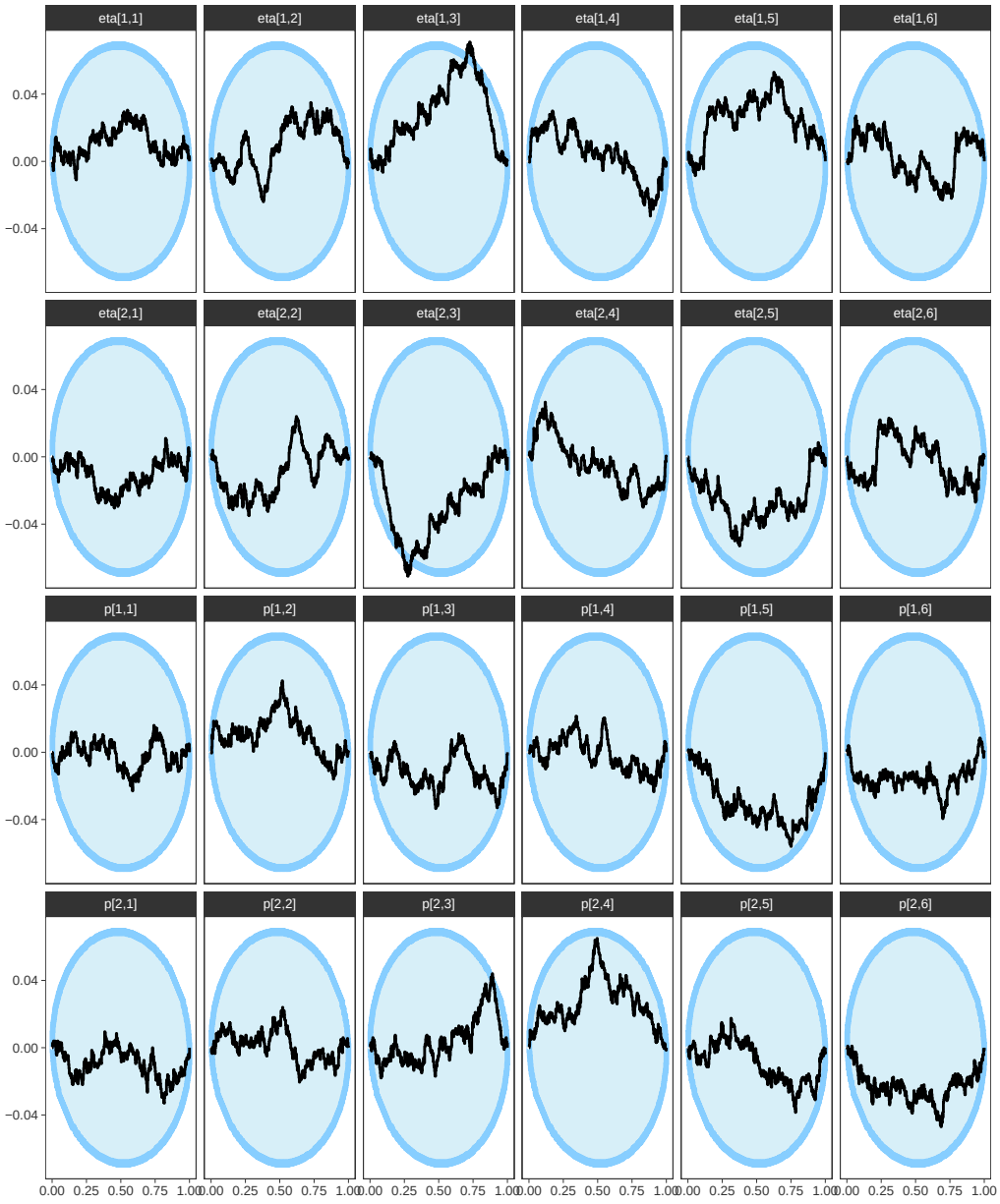
```
datasets <- SBC_generator_function(simulate_cmr, N_super = N_super, J = J,
                                   K = K, S = S, JS = T) |>
  generate_datasets(n_sims)
backend <- SBC_backend_cmdstan_sample(cmdstan_model(here("stan/js-ms-rd.stan")),
                                      init = 0.1, chains = chains,
                                      iter_warmup = iter_warmup,
                                      iter_sampling = iter_sampling)
sbc <- compute_SBC(datasets, backend, cores_per_fit = chains, keep_fits = F)
```

Mean time for the slowest of 8 chains with 700 iterations: 0.775 minutes.

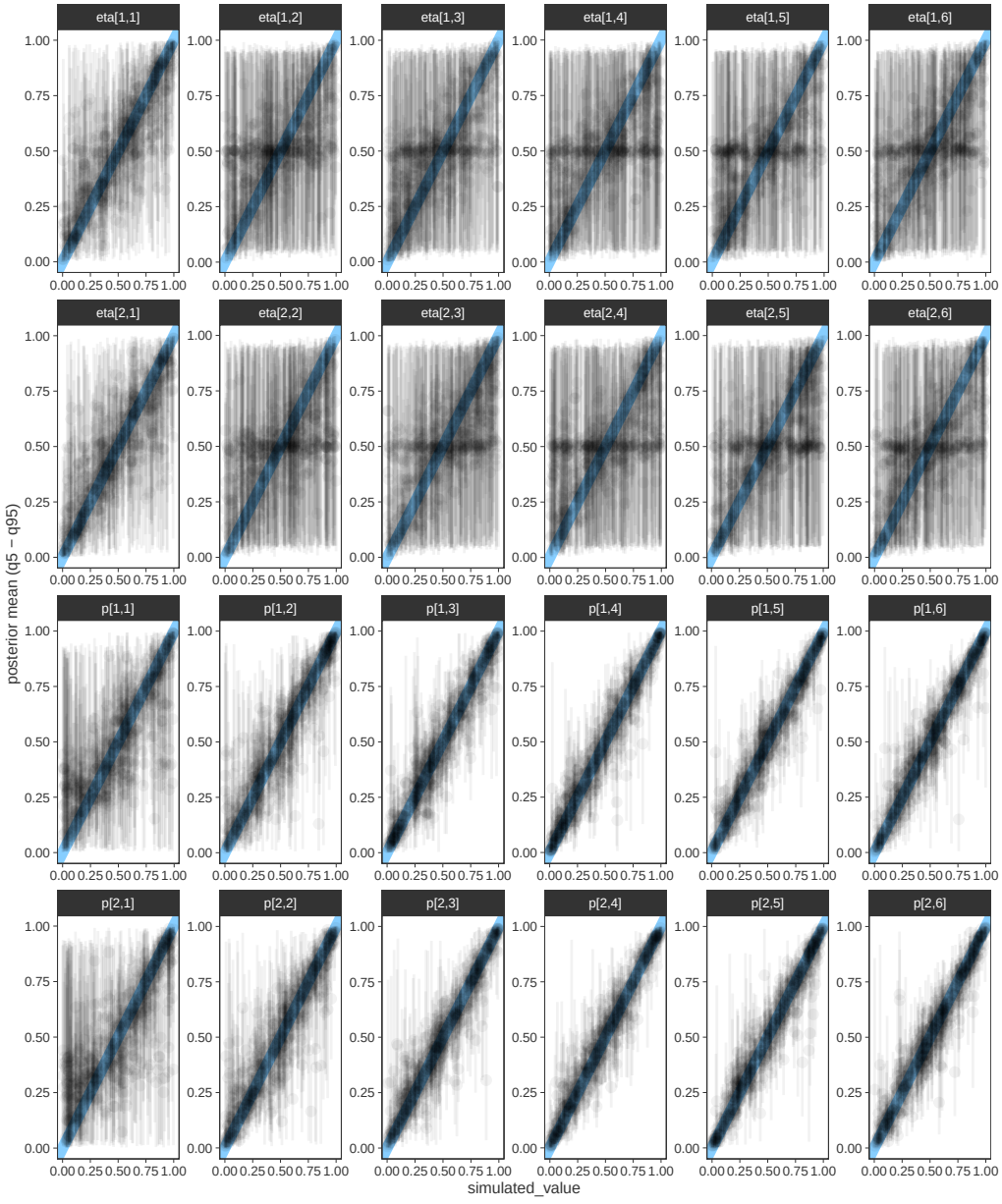
```
plot_sbc(sbc, c(str_c("h[", 1:S, "]"), str_c("q[", 1:(S * (S - 1)), "]"),
                str_c("beta[", 1:J, "]"), "mu", "N_super"), ncol = J)
```



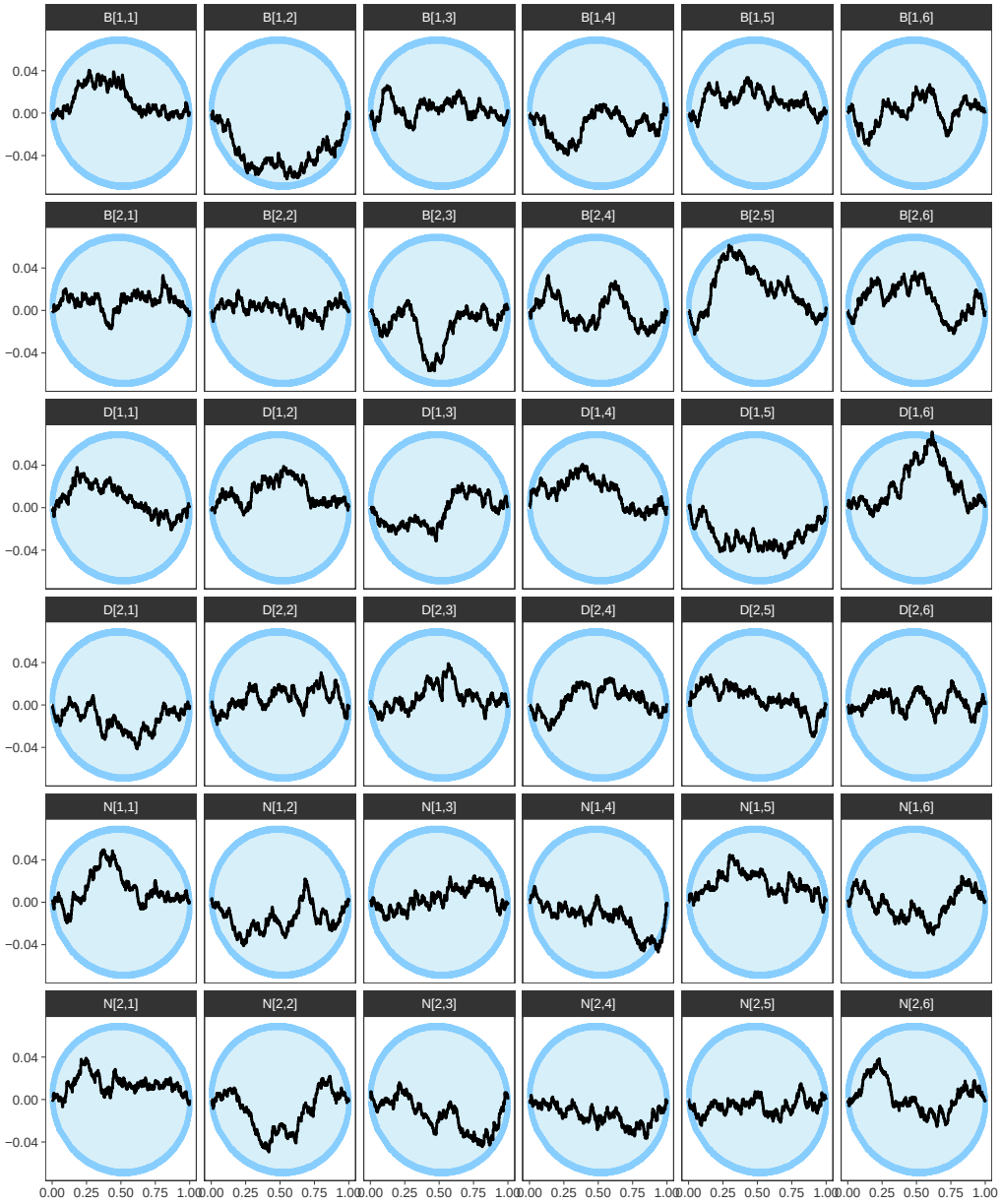
```
params <- map(c("p", "eta", "N", "B", "D"), \(x)
              flatten_chr(map(1:S, \(s) str_c(x, "[", s, ",", 1:J, "]"))))
plot_ecdf_diff(sbc, flatten_chr(params[1:2])) +
  facet_wrap(~ group, ncol = J) +
  theme(legend.position = "none")
```



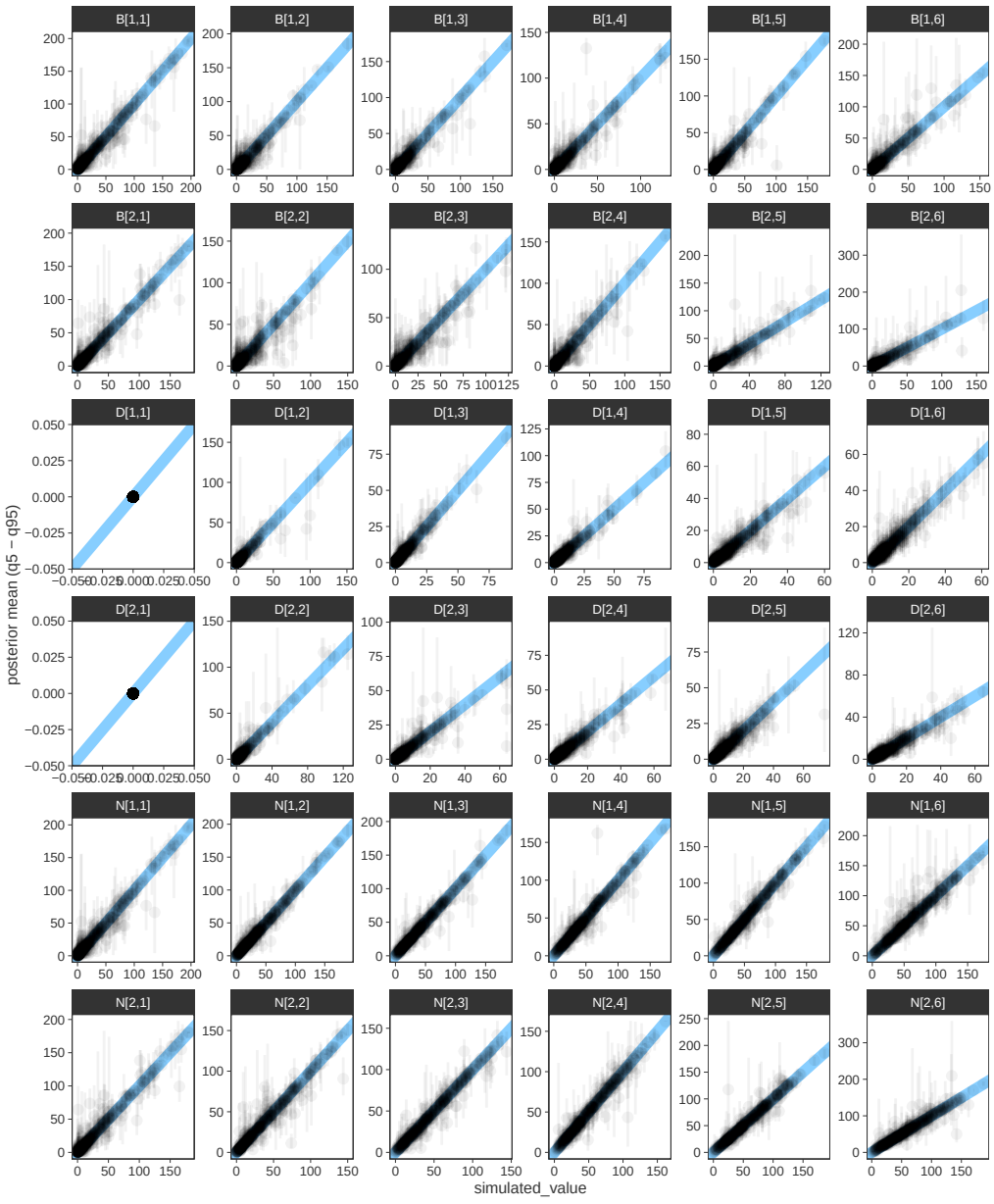
```
plot_sim_estimated(sbc, flatten_chr(params[1:2])) +
  facet_wrap(~ variable, ncol = J, scales = "free")
```



```
plot_ecdf_diff(sbc, flatten_chr(params[3:5])) +
  facet_wrap(~ variable, ncol = J) +
  theme(legend.position = "none")
```



```
plot_sim_estimated(sbc, flatten_chr(params[3:5])) +
  facet_wrap(~ variable, ncol = J, scales = "free")
```



Multievent Jolly-Seber

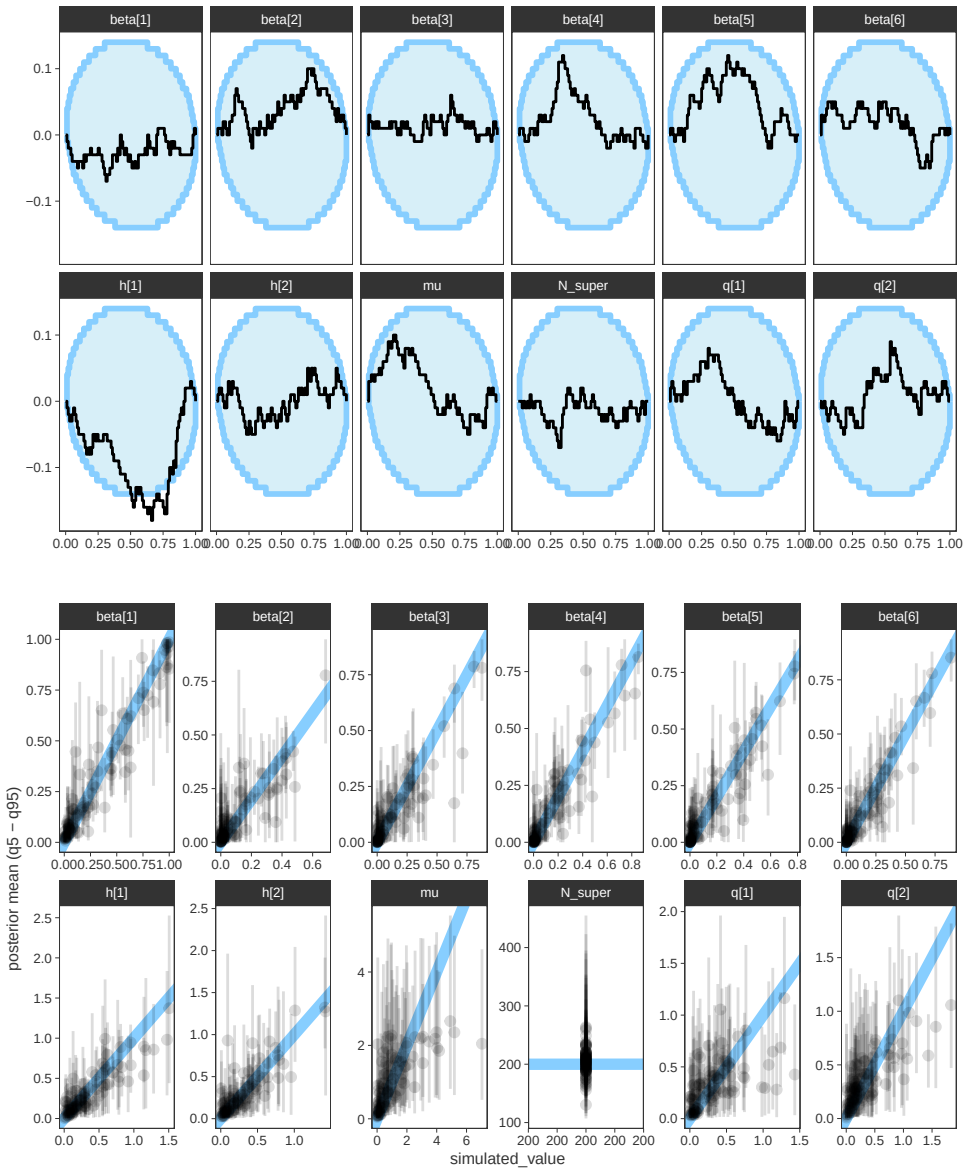
Multievent Jolly-Seber models are computationally most demanding because the likelihood doesn't reduce to many terms that can be shared across individuals and surveys. To reduce computational burden of this document, SBC is performed with a reduced 100 simulations per model.

Single survey

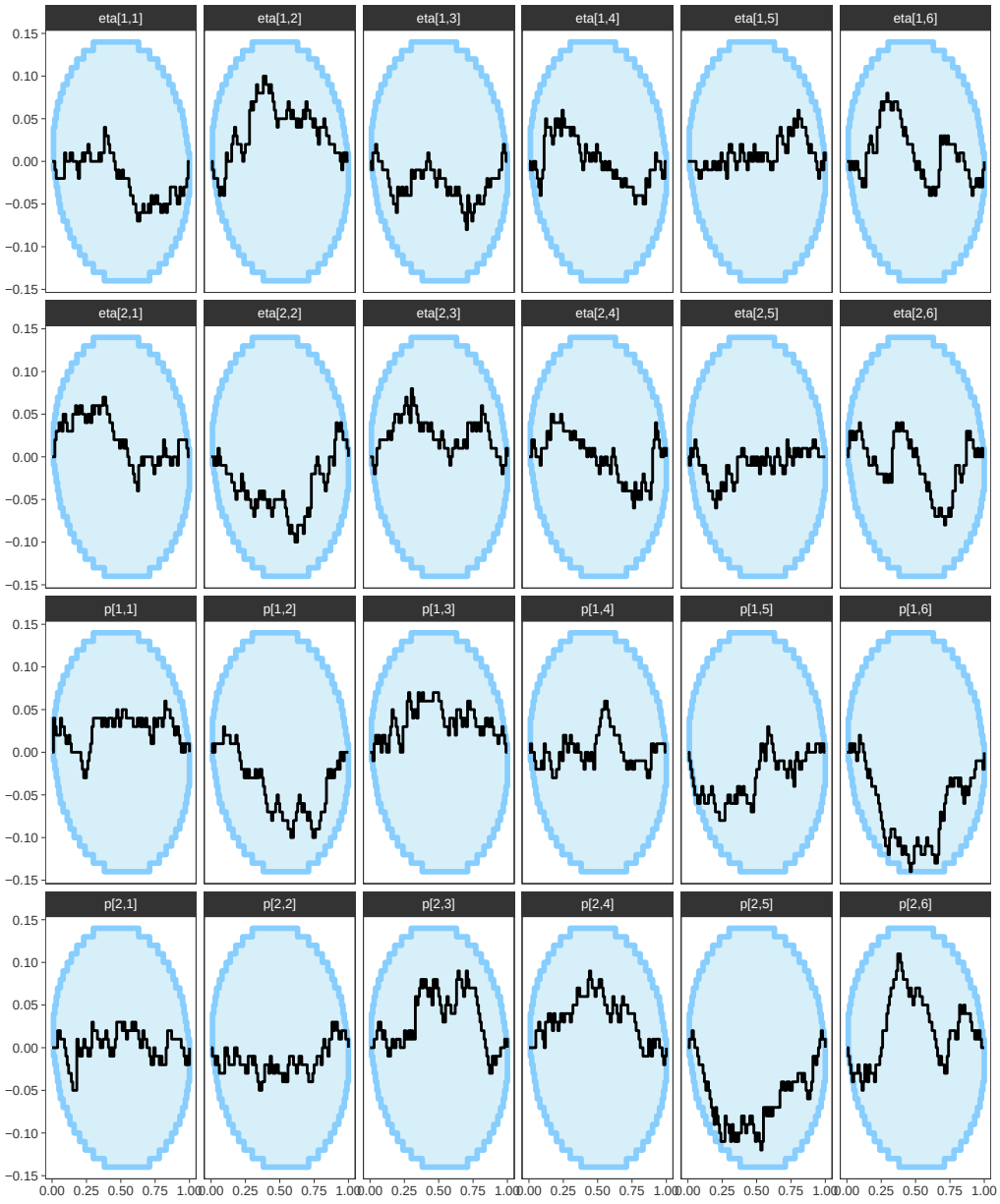
```
datasets <- SBC_generator_function(simulate_cmr, N_super = N_super, J = J,
                                   S = S, ME = T, JS = T) |>
  generate_datasets(n_sims <- 100)
backend <- SBC_backend_cmdstan_sample(cmdstan_model(here("stan/js-me.stan")),
                                      init = 0.1, chains = chains,
                                      iter_warmup = iter_warmup,
                                      iter_sampling = iter_sampling)
sbc <- compute_SBC(datasets, backend, cores_per_fit = chains, keep_fits = F)
```

Mean time for the slowest of 8 chains with 700 iterations: 5.089 minutes.

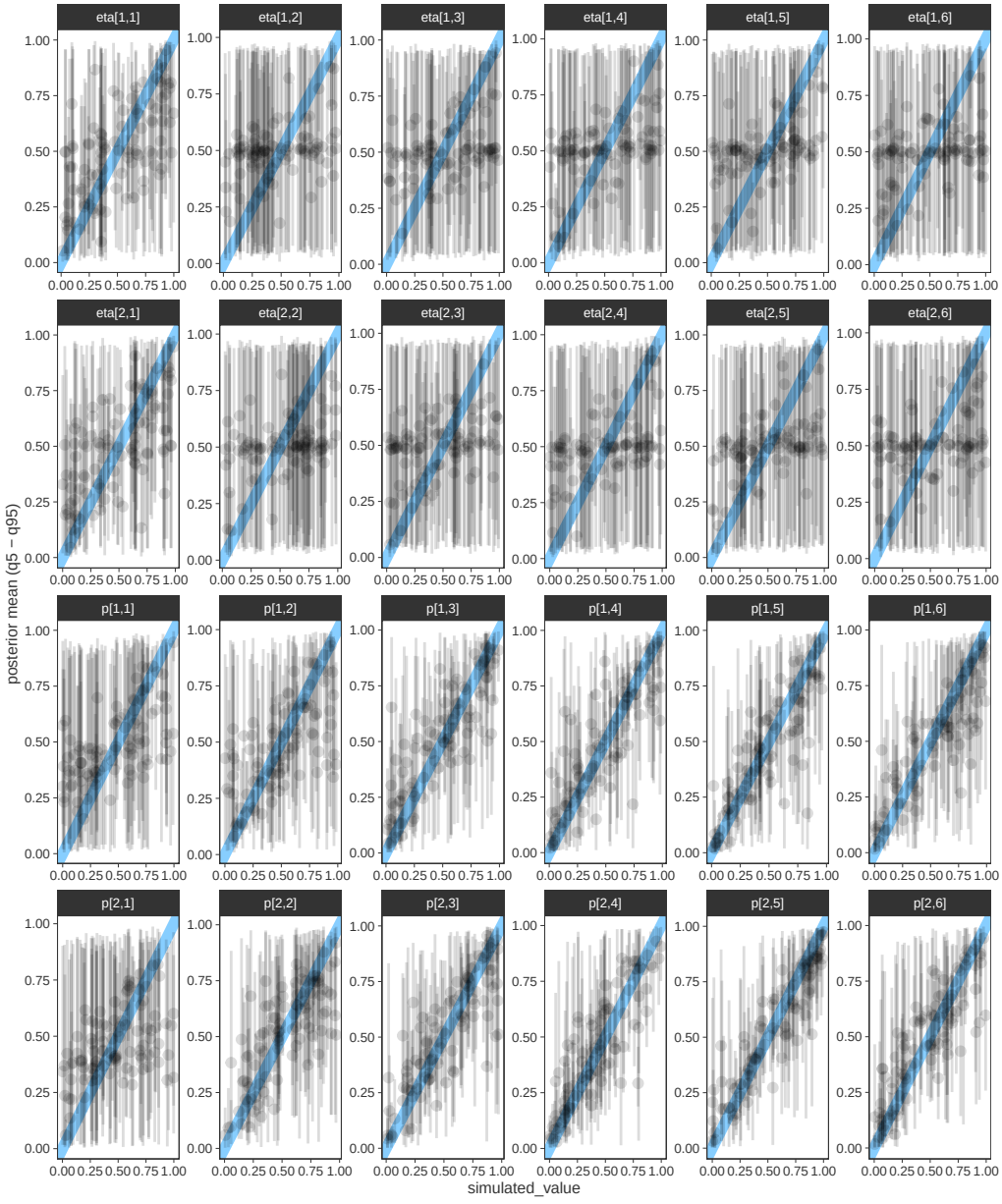
```
plot_sbc(sbc, c(str_c("h[", 1:S, "]"), str_c("q[", 1:(S * (S - 1)), "]"),
                str_c("beta[", 1:J, "]"), str_c("delta[", 1:(S - 1), "]"),
                "mu", "N_super"), ncol = J)
```



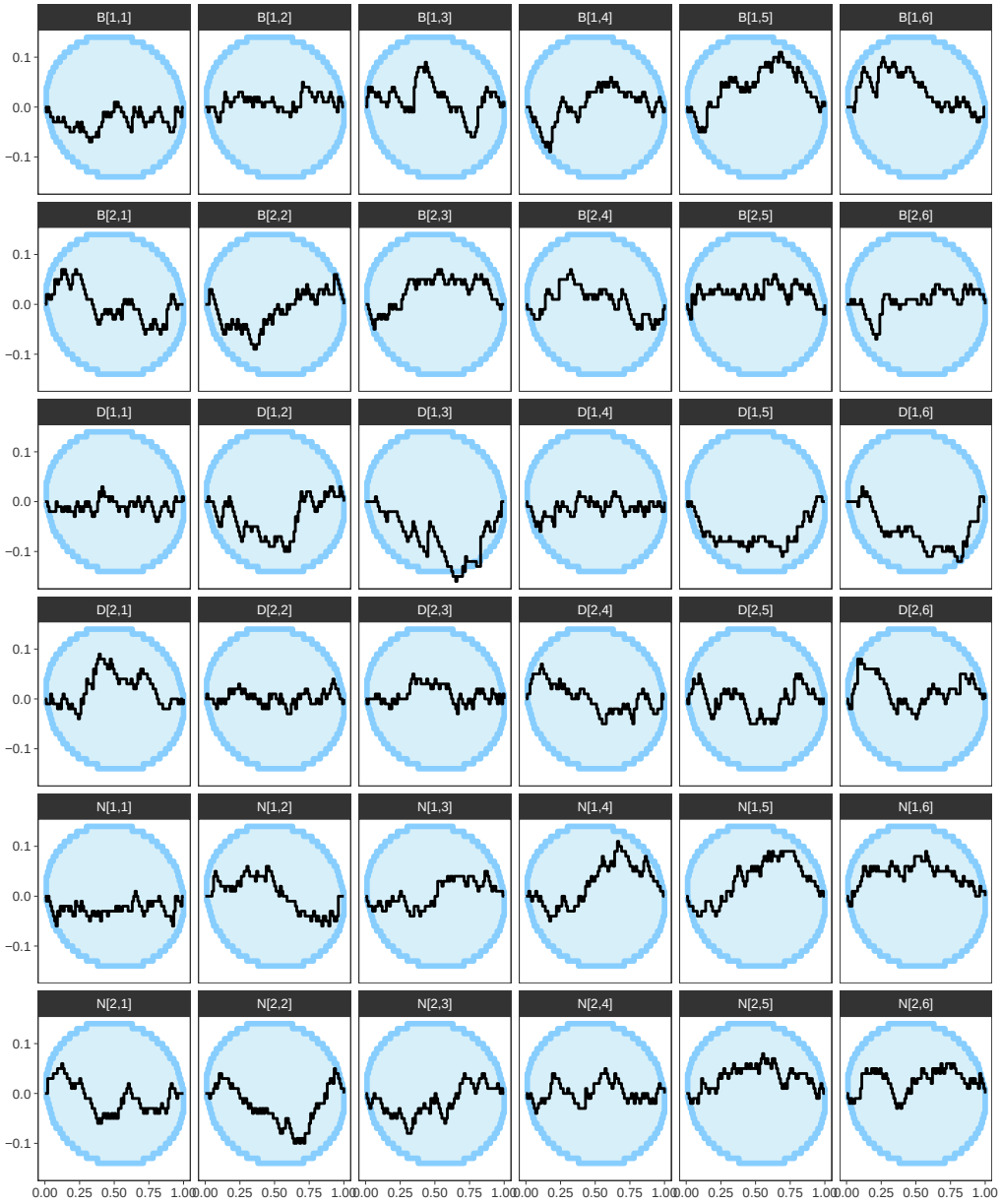
```
params <- map(c("p", "eta", "N", "B", "D"), \(x)
              flatten_chr(map(1:S, \(s) str_c(x, "[", s, ",", 1:J, "]"))))
plot_ecdf_diff(sbc, flatten_chr(params[1:2])) +
  facet_wrap(~ group, ncol = J) +
  theme(legend.position = "none")
```



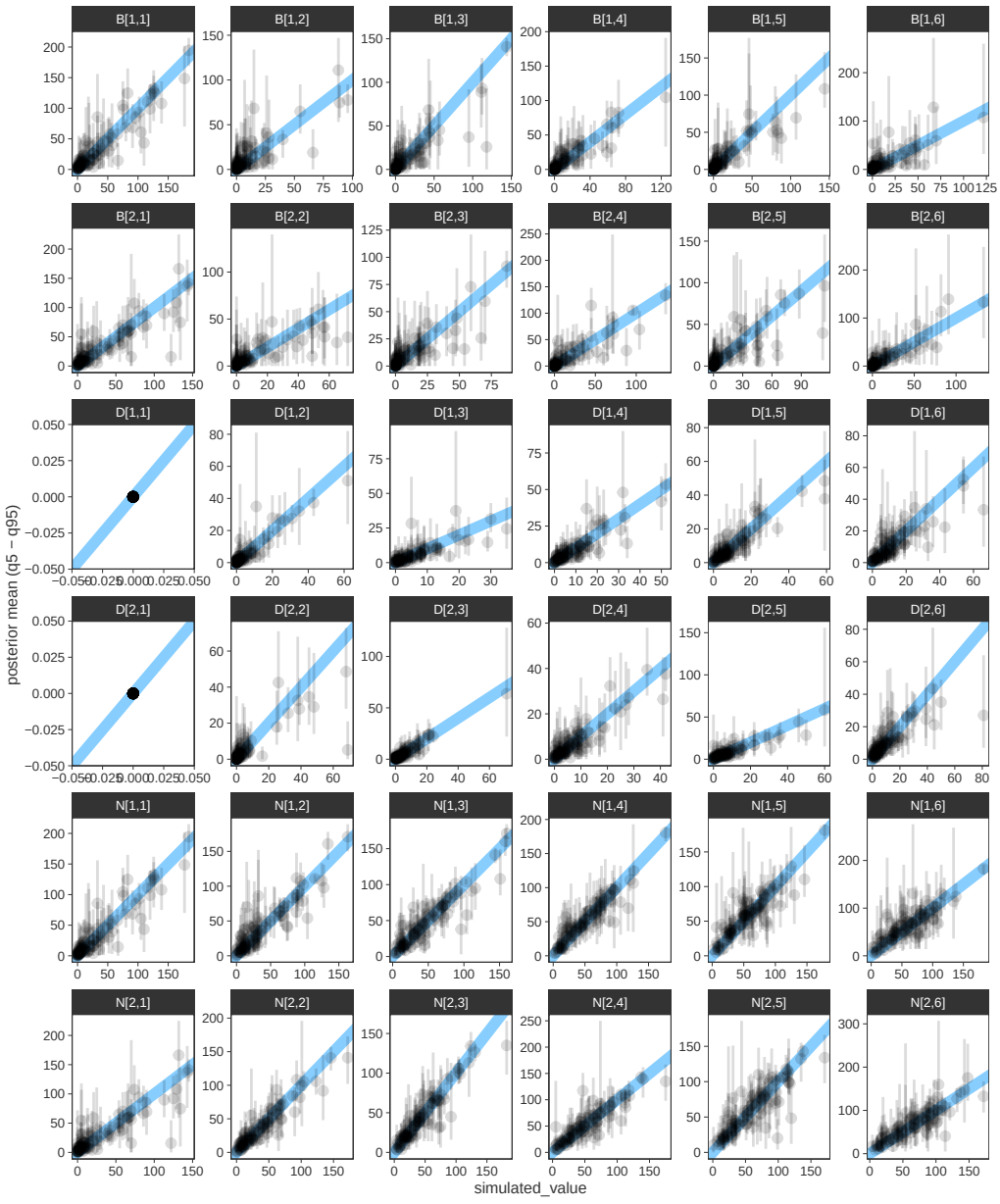
```
plot_sim_estimated(sbc, flatten_chr(params[1:2])) +
  facet_wrap(~ variable, ncol = J, scales = "free")
```



```
plot_ecdf_diff(sbc, flatten_chr(params[3:5])) +
  facet_wrap(~ variable, ncol = J) +
  theme(legend.position = "none")
```



```
plot_sim_estimated(sbc, flatten_chr(params[3:5])) +
  facet_wrap(~ variable, ncol = J, scales = "free")
```

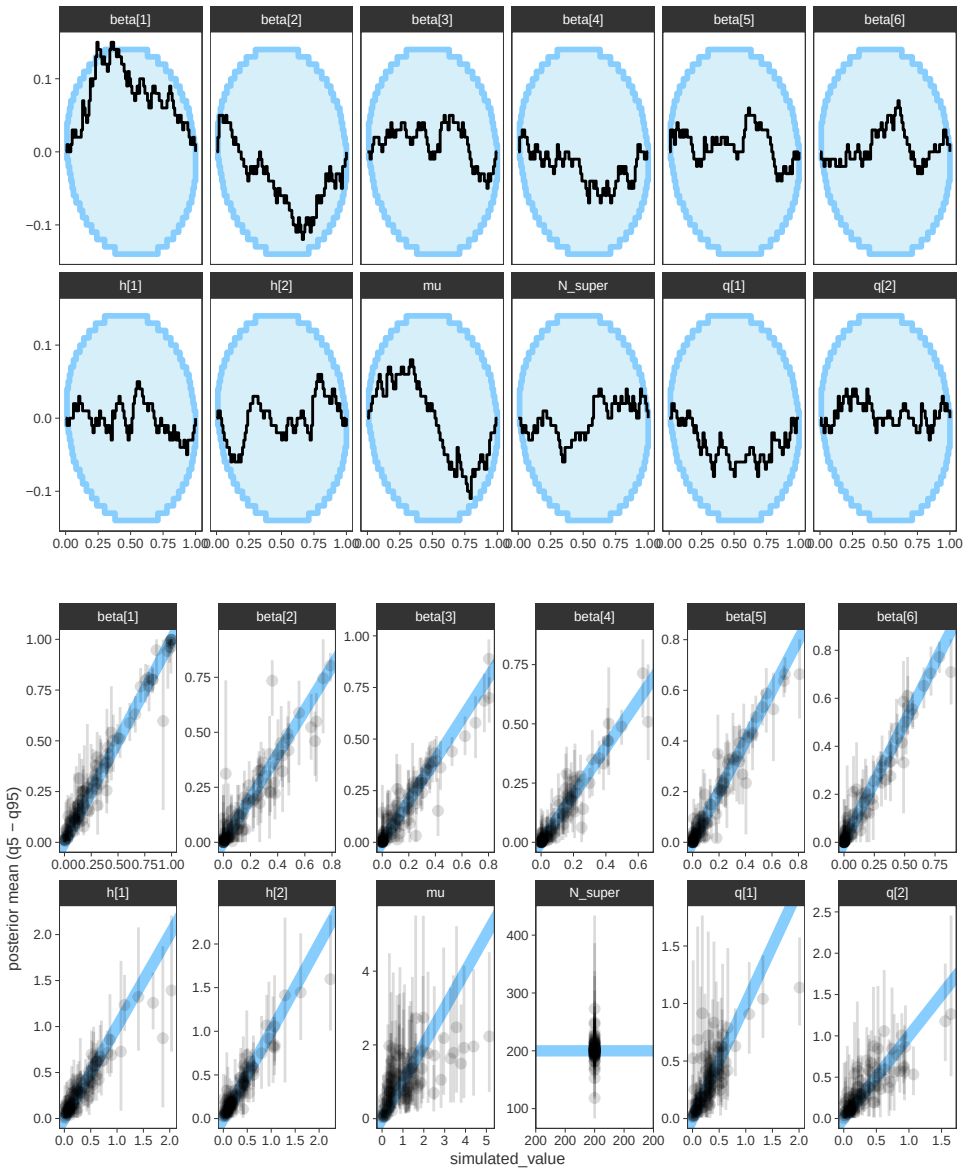


Robust design

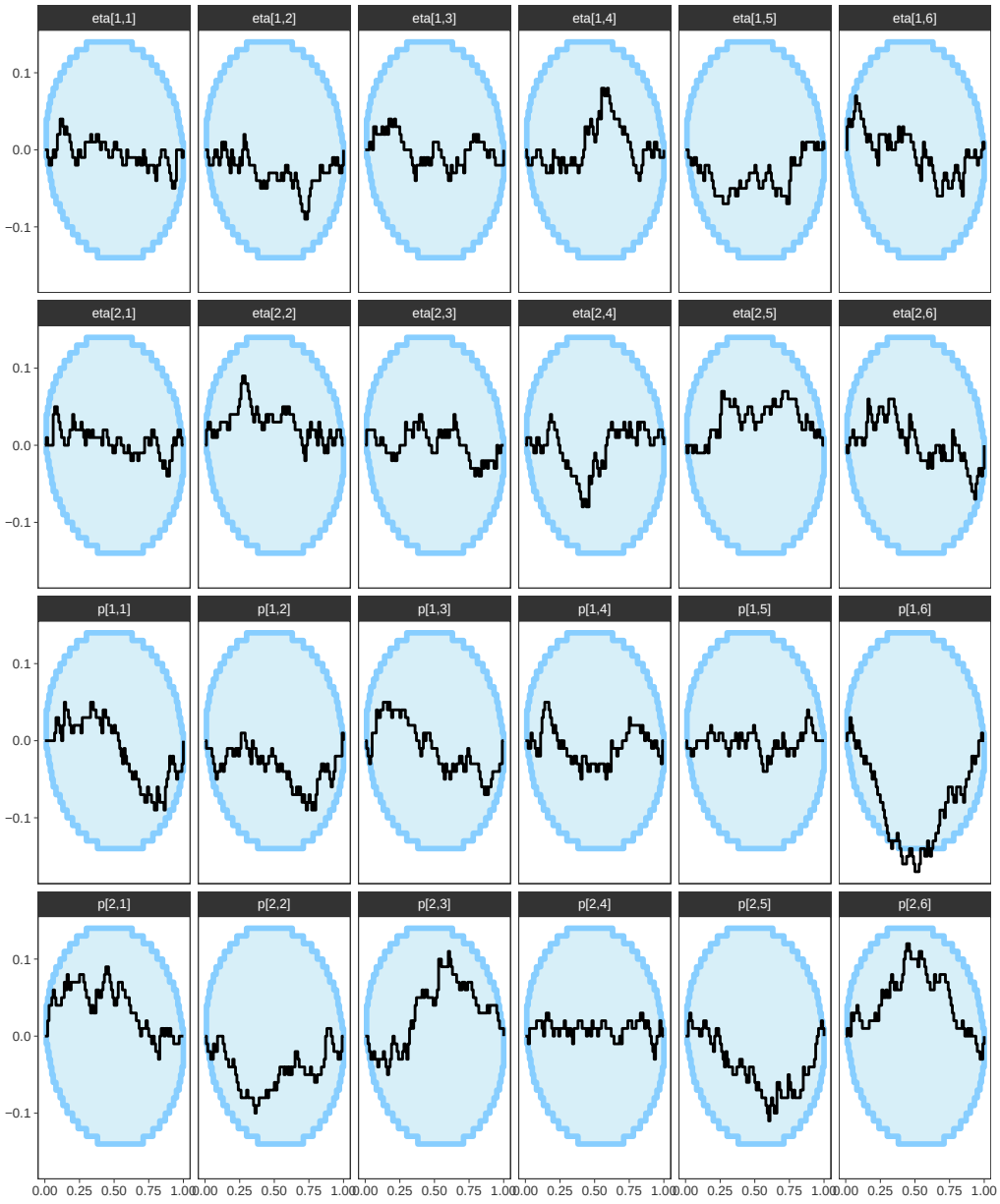
```
datasets <- SBC_generator_function(simulate_cmr, N_super = N_super, J = J,
                                   K = K, S = S, ME = T, JS = T) |>
  generate_datasets(n_sims)
backend <- SBC_backend_cmdstan_sample(cmdstan_model(here("stan/js-me-rd.stan")),
                                      init = 0.1, chains = chains,
                                      iter_warmup = iter_warmup,
                                      iter_sampling = iter_sampling)
sbc <- compute_SBC(datasets, backend, cores_per_fit = chains, keep_fits = F)
```

Mean time for the slowest of 8 chains with 700 iterations: 3.871 minutes.

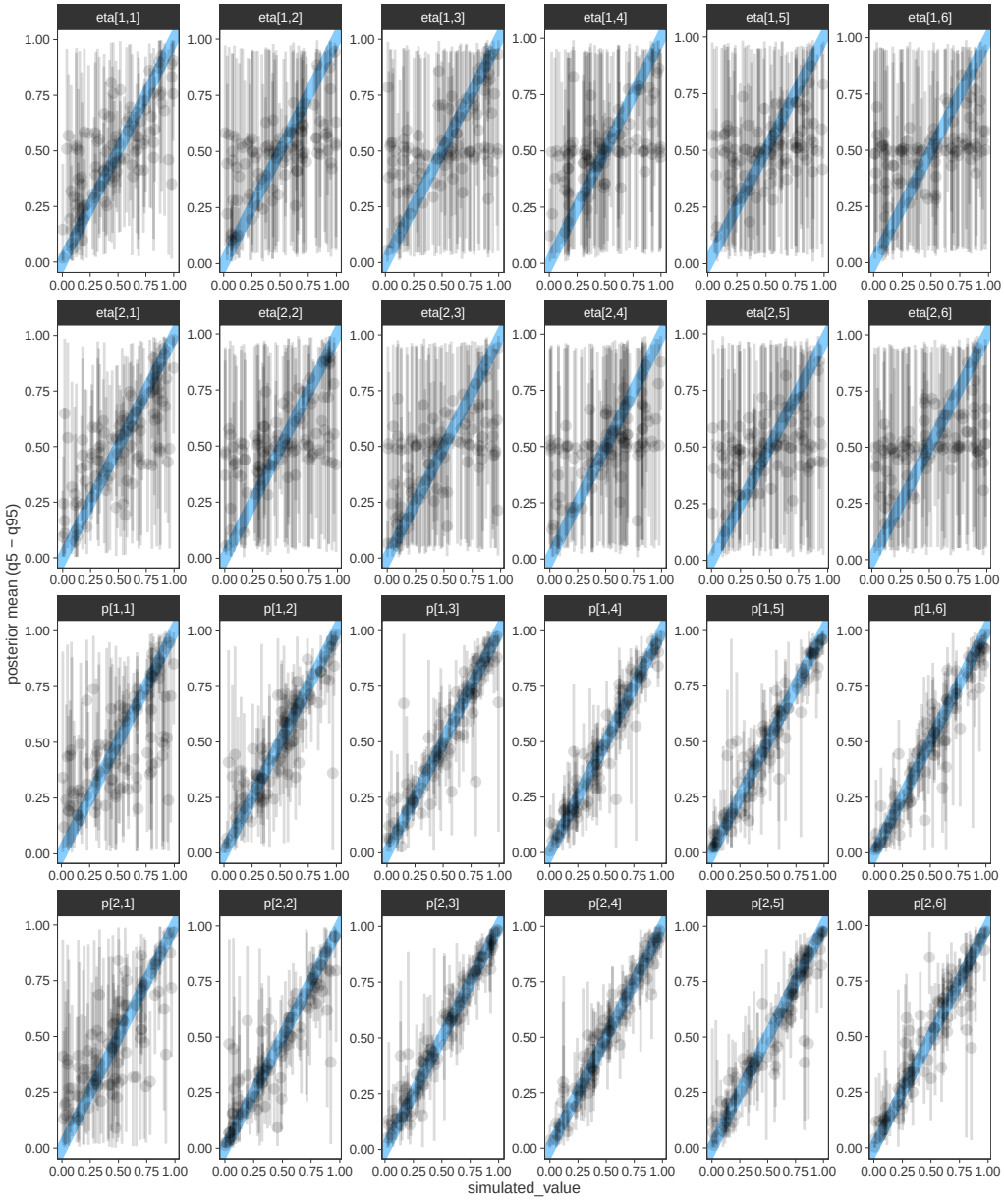
```
plot_sbc(sbc, c(str_c("h[", 1:S, "]"), str_c("q[", 1:(S * (S - 1)), "]"),
                str_c("beta[", 1:J, "]"), str_c("delta[", 1:(S - 1), "]"),
                "mu", "N_super"), ncol = J)
```



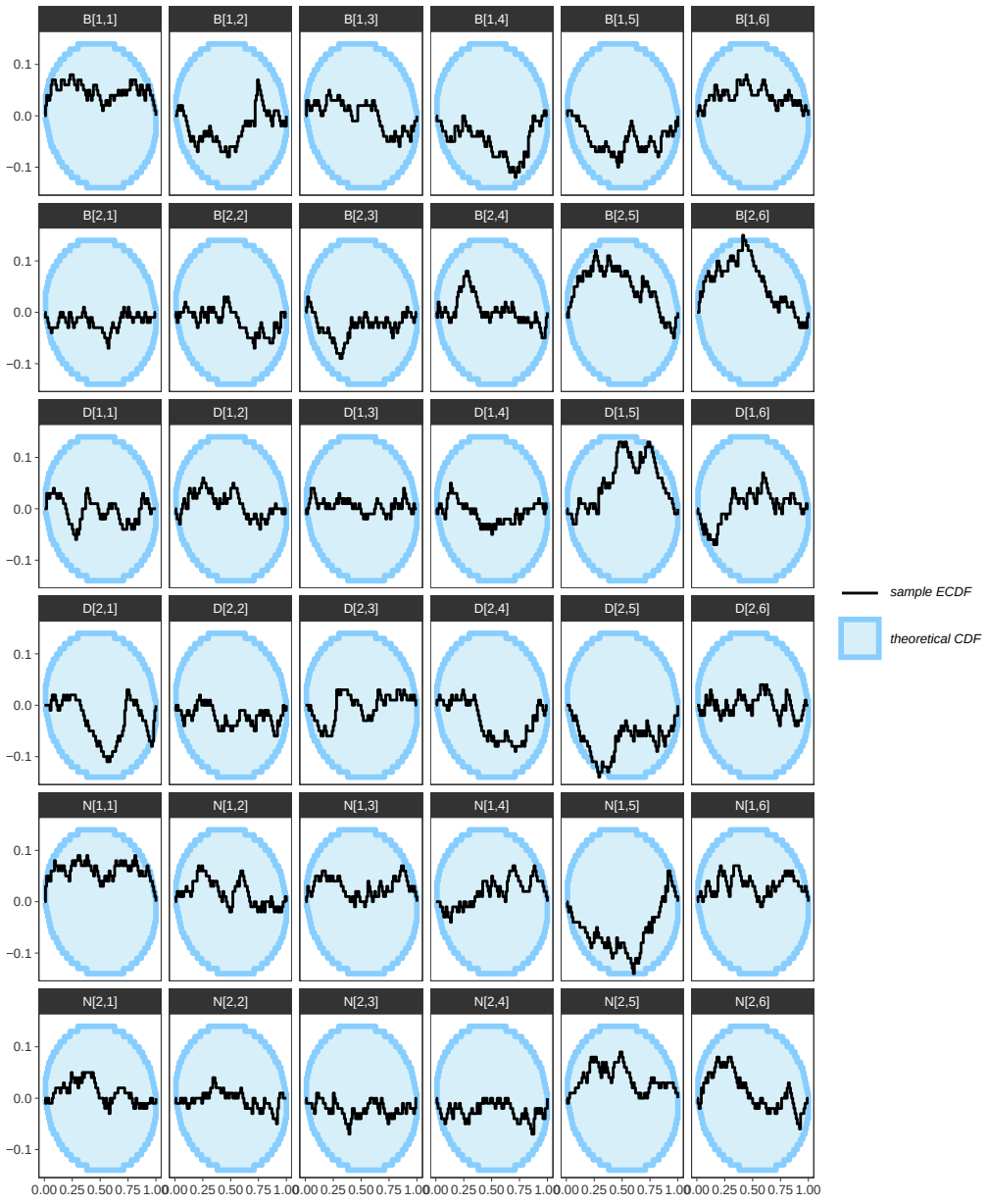
```
params <- map(c("p", "eta", "N", "B", "D"), \(x)
              flatten_chr(map(1:S, \(s) str_c(x, "[", s, ",", 1:J, "]"))))
plot_ecdf_diff(sbc, flatten_chr(params[1:2])) +
  facet_wrap(~ group, ncol = J) +
  theme(legend.position = "none")
```



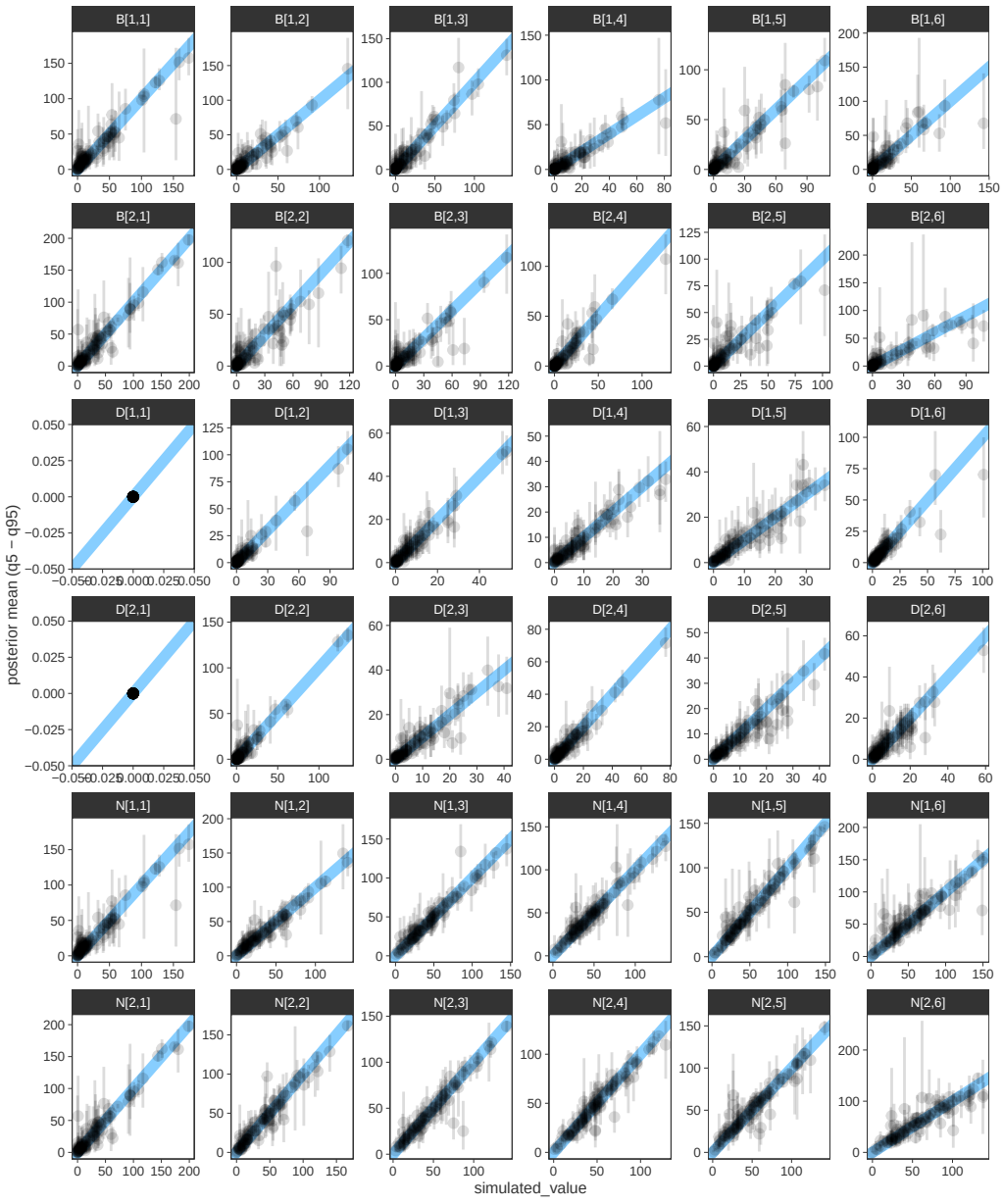
```
plot_sim_estimated(sbc, flatten_chr(params[1:2])) +
  facet_wrap(~ variable, ncol = J, scales = "free")
```



```
plot_ecdf_diff(sbc, flatten_chr(params[3:5])) +
  facet_wrap(~ variable, ncol = J)
```



```
plot_sim_estimated(sbc, flatten_chr(params[3:5])) +
  facet_wrap(~ variable, ncol = J, scales = "free")
```



Simulation script

```r
 if (!require(expm)) install.packages("expm")
simulate_cmr <- function(N_super = 100,
                         J = 8,
                         K = 1,
                         S = 1,
                         ME = 0,
                         JS = 0,
                         I_aug = 500,
                         grainsize = 0,
                         mu_gamma_prior = c(1, 1),
                         h_gamma_prior = c(1, 3),
                         p_beta_prior = c(1, 1),
                         q_gamma_prior = c(1, 3),
                         delta_beta_prior = c(1, 1),
                         eta_dirichlet_prior = 1) {

  # metadata, parameters, and detection histories
  Jm1 <- J - 1
  tau <- rlnorm(Jm1)
  tau_scl <- tau / mean(tau)
  h <- rgamma(S, h_gamma_prior[1], h_gamma_prior[2])
  p <- matrix(rbeta(S * J, p_beta_prior[1], p_beta_prior[2]), S, J)
  if (S == 1) {
    phi_tau <- exp(-h * tau)
  } else {
    Sm1 <- S - 1 ; Sp1 <- S + 1
    q <- rgamma(S * Sm1, q_gamma_prior[1], q_gamma_prior[2])
    H <- array(0, c(Jm1, Sp1, Sp1))
    for (j in 1:Jm1) {
      H[j, , ] <- expm::expm(rate_matrix(h, q) * tau_scl[j])
    }
    if (ME) {
      delta <- rbeta(Sm1, delta_beta_prior[1], delta_beta_prior[2])
      E <- triangular_bidiagonal_stochastic_matrix(delta)
    }
    if (ME | JS) {
      eta <- matrix(0, S, J)
      if (length(eta_dirichlet_prior) == 1) {
        alpha <- rep(eta_dirichlet_prior, S)
```

```
    } else {
      alpha <- eta_dirichlet_prior
    }
    for (j in 1:J) {
      eta[, j] <- rdirch(1, alpha)
    }
  }
}
z <- matrix(0, N_super, J)
y <- array(0, c(N_super, J, K))

# simulate
if (JS) {
  mu <- rgamma(1, mu_gamma_prior[1], mu_gamma_prior[2])
  beta <- rdirch(1, c(1, mu * tau_scl))
  b <- sort(rcat(N_super, beta))
  if (S == 1) {
    B <- D <- numeric(J)
    for (i in 1:N_super) {
      z[i, b[i]] <- 1
      B[b[i]] <- B[b[i]] + 1
      if (b[i] < J) {
        for (j in (b[i] + 1):J) {
          jm1 <- j - 1
          z[i, j] <- rbinom(1, 1, z[i, jm1] * phi_tau[jm1])
          if (z[i, jm1] == 1 & z[i, j] == 0) {
            D[j] <- D[j] + 1
          }
        }
      }
      for (j in b[i]:J) {
        y[i, j, ] <- rbinom(K, 1, z[i, j] * p[j])
      }
    }
  } else {
    B <- D <- matrix(0, S, J)
    for (i in 1:N_super) {
      z[i, b[i]] <- rcat(1, eta[, b[i]])
      B[z[i, b[i]], b[i]] <- B[z[i, b[i]], b[i]] + 1
      if (b[i] < J) {
        for (j in (b[i] + 1):J) {
          jm1 <- j - 1
```

```
            z[i, j] <- rcat(1, H[jm1, z[i, jm1], ])
            if (z[i, jm1] <= S & z[i, j] == Sp1) {
              D[z[i, jm1], j] <- D[z[i, jm1], j] + 1
            }
          }
        }
        for (j in b[i]:J) {
          if (z[i, j] <= S) {
            y[i, j, ] <- rbinom(K, 1, p[z[i, j], j]) * z[i, j]
            if (ME) {
              for (k in 1:K) {
                if (y[i, j, k]) {
                  y[i, j, k] <- rcat(1, E[z[i, j], ])
                }
              }
            }
          }
        }
      }
    }
    obs <- which(rowSums(y) > 0)
    I <- length(obs)
    y <- y[obs, , 1:K]
  } else {
    I <- N_super
    f <- sort(sample(1:ifelse(K > 1, J, Jm1), I, replace = T))
    if (S == 1) {
      for (i in 1:I) {
        f_k <- sample(1:K, 1)
        z[i, f[i]] <- y[i, f[i], f_k] <- 1
        for (k in setdiff(1:K, f_k)) {
          y[i, f[i], k] <- rbinom(1, 1, p[f[i]])
        }
        if (f[i] < J) {
          for (j in (f[i] + 1):J) {
            jm1 <- j - 1
            z[i, j] <- rbinom(1, 1, z[i, jm1] * phi_tau[jm1])
            y[i, j, ] <- rbinom(K, 1, z[i, j] * p[j])
          }
        }
      }
    } else {
```

```
    for (i in 1:I) {
      f_k <- sample(1:K, 1)
      if (ME) {
        z[i, f[i]] <- rcat(1, eta[, f[i]])
        y[i, f[i], f_k] <- rcat(1, E[z[i, f[i]], ])
      } else {
        z[i, f[i]] <- y[i, f[i], f_k] <- sample(1:S, 1)
      }
      for (k in setdiff(1:K, f_k)) {
        y[i, f[i], k] <- rbinom(1, 1, p[z[i, f[i]], f[i]]) * z[i, f[i]]
        if (ME) {
          if (y[i, f[i], k]) {
            y[i, f[i], k] <- rcat(1, E[z[i, f[i]], ])
          }
        }
      }
      if (f[i] < J) {
        for (j in (f[i] + 1):J) {
          jm1 <- j - 1
          z[i, j] <- rcat(1, H[jm1, z[i, jm1], ])
          if (z[i, j] <= S) {
            y[i, j, ] <- rbinom(K, 1, p[z[i, j], j]) * z[i, j]
            if (ME) {
              for (k in 1:K) {
                if (y[i, j, k]) {
                  y[i, j, k] <- rcat(1, E[z[i, j], ])
                }
              }
            }
          }
        }
      }
    }
  }
  y <- y[, , 1:K]
}

# output
if (JS | K > 1) {
  p <- p[1:S, ]
} else {
  p <- p[1:S, -1]
```

```
  }
  variables <- list(h = h, p = p)
  generated <- list(I = I, J = J, tau = tau, y = y, grainsize = grainsize)
  if (K > 1) {
    generated <- append(generated,
                        list(K_max = K, K = rep(K, J)),
                        after = 2)
  }
  if (S > 1) {
    generated$S <- S
    variables <- append(variables, list(q = q))
    if (ME & !JS) {
      variables <- append(variables, list(eta = eta, delta = delta))
    }
  }
  if (JS) {
    generated$I_aug <- I_aug
    variables <- append(variables,
                        list(mu = mu,
                             beta = beta,
                             N_super = N_super,
                             B = B,
                             D = D,
                             N = apply(z, 2, \(j)
                                       sapply(1:S, \(s) sum(j == s)))))
    if (S > 1) {
      variables <- append(variables, list(eta = eta))
    }
  }
  list(variables = variables, generated = generated)
}
```

Utility functions

```r
# transition rate matrix from vectors of mortality and transition rates
rate_matrix <- function(h, q) {
  S <- length(h)
  Sp1 <- S + 1 ; Sm1 = S - 1
  Q <- matrix(0, Sp1, Sp1)
  Q[1:S, Sp1] <- h
  q_s <- head(q, Sm1)
  Q[1, 1] <- -(h[1] + sum(q_s))
  Q[1, 2:S] <- q_s
  if (S > 2) {
    idx <- S
    for (s in 2:Sm1) {
      q_s <- q[idx:(idx + S - 2)]
      Q[s, 1:(s - 1)] <- head(q_s, s - 1)
      Q[s, s] <- -(h[s] + sum(q_s))
      Q[s, (s + 1):S] <- tail(q_s, S - s)
      idx <- idx + Sm1
    }
  }
  q_s <- tail(q, Sm1)
  Q[S, 1:Sm1] <- q_s
  Q[S, S] <- -(h[S] + sum(q_s))
  Q
}

# triangular biadiagonal stochastic matrix
triangular_bidiagonal_stochastic_matrix <- function(delta) {
  E <- diag(c(1, delta))
  for (s in 2:(length(delta) + 1)) {
    sm1 <- s - 1
    E[s, sm1:s] <- c(1 - delta[sm1], delta[sm1])
  }
  E
}

# random categorical draws
rcat <- function(n = 1, prob) {
  rmultinom(n, size = 1, prob) |>
    apply(2, \(x) which(x == 1))
```

```
}

# random Dirichlet draws
rdirch <- function(n = 1, alpha) {
  D <- length(alpha)
  out <- matrix(NA, n, D)
  for (i in 1:n) {
    raw <- rgamma(D, alpha, 1)
    out[i, ] <- raw / sum(raw)
  }
  if (n == 1) {
    out[1, ]
  } else {
    out
  }
}

# plot ECDF-diff and estimates plots together
if (!require(patchwork)) install.packages("patchwork")
plot_sbc <- function(sbc, ..., nrow = NULL, ncol = NULL) {
  patchwork::wrap_plots(
    plot_ecdf_diff(sbc, ...) +
      ggplot2::facet_wrap(~ group,
                          nrow = nrow,
                          ncol = ncol) +
      ggplot2::theme(legend.position = "none"),
    plot_sim_estimated(sbc, ...) +
      ggplot2::facet_wrap(~ variable,
                          nrow = nrow,
                          ncol = ncol,
                          scales = "free"),
    ncol = 1
  )
}
```

References

Gabry, J., R. Češnovar, A. Johnson, and S. Bronder. 2025. cmdstanr: R Interface to 'CmdStan'. Manual.

Hollanders, M., and J. A. Royle. 2022. Know what you don't know: Embracing state uncertainty in disease-structured multistate models. Methods in Ecology and Evolution 13:2827–2837.

Modrák, M., A. H. Moon, S. Kim, P. Bürkner, N. Huurre, K. Faltejsková, A. Gelman, and A. Vehtari. 2023. Simulation-based calibration checking for Bayesian computation: The choice of test quantities shapes sensitivity. Bayesian Analysis -1.

Schwarz, C. J., and A. N. Arnason. 2008. Chapter 12. Jolly-Seber models in MARK. Program MARK: A Gentle Introduction. 19th edition.

Wickham, H., M. Averick, J. Bryan, W. Chang, L. D. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. L. Pedersen, E. Miller, S. M. Bache, K. Müller, J. Ooms, D. Robinson, D. P. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, and H. Yutani. 2019. Welcome to the tidyverse. Journal of Open Source Software 4:1686.