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

In this document we run simulation-based calibration (SBC) for each model described in the manuscript. We assess calibration through ECDF-diff 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

We use simulation-based calibration (SBC) to assess whether our Stan programs correctly reflect our data-generating processes. 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. If the model reflects our data-generating process, then the ranks are uniformly distributed, which we assess with the `SBC::plot_ecdf_diff()` function. If our model is calibrated, then the black squiggly lines should fall within the blue ellipses. We also plot the parameter estimates alongside their input to visually assess parameter recovery.

In all models, we simulate constant state-specific mortality hazard rates and time-varying state-specific detection probabilities. Survey intervals are simulated as unequal which are accounted for in the entry and ecological processes. In multistate models, constant state-to-state specific transition rates are simulated. In Jolly-Seber models, time-varying entry probabilities are simulated with an offset for survey interval. In multistate Jolly-Seber moels, the probabilities of entering in each state are modeled as time-varying.

First we load packages and scripts and set the parameters for simulation and estimation.

```
options(digits = 3)
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 <- 3 ; n_sims <- 100
chains <- 8 ; iter_warmup <- 200 ; iter_sampling <- 500
```

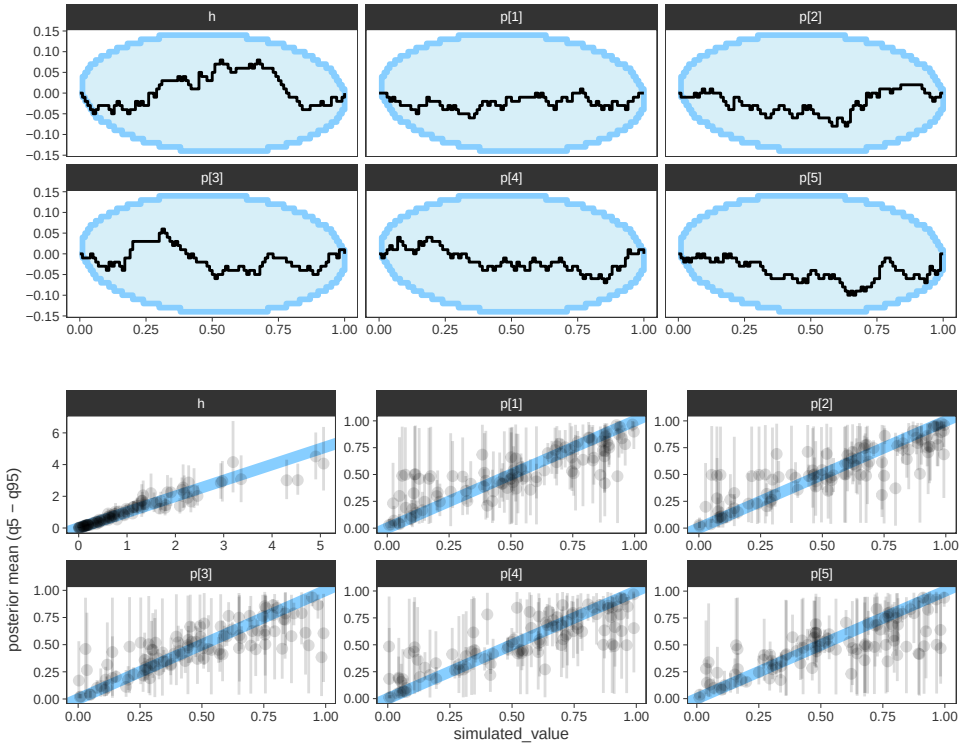
Cormack-Jolly-Seber

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")),
                                      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: 0.003 minutes.

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

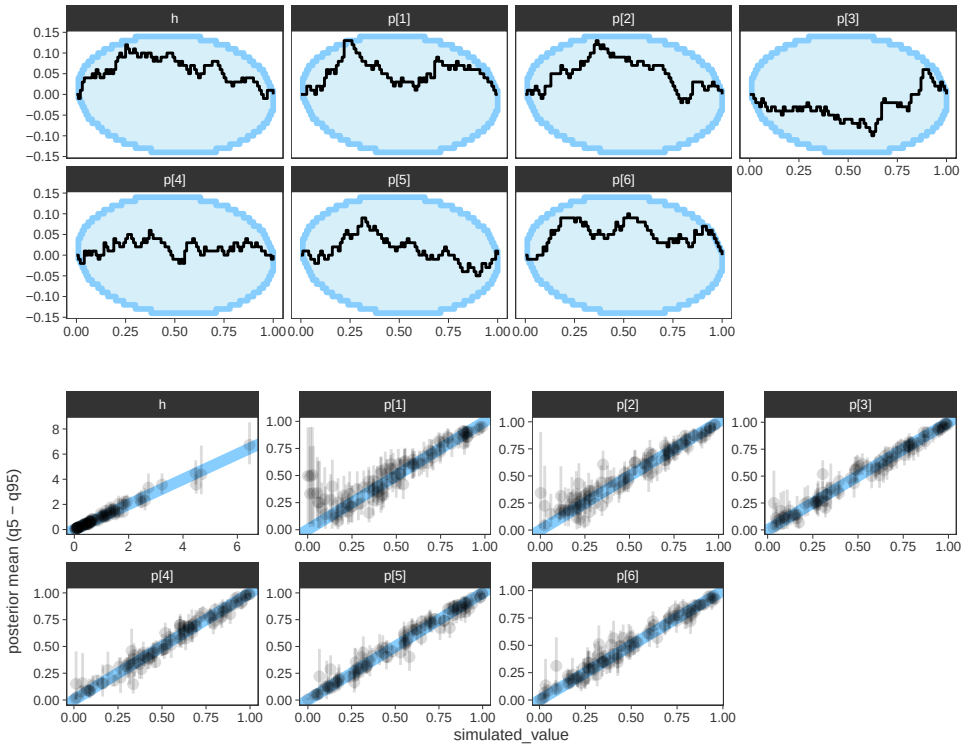


Robust design

```
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")),
                                      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)
```



Multistate Cormack-Jolly-Seber

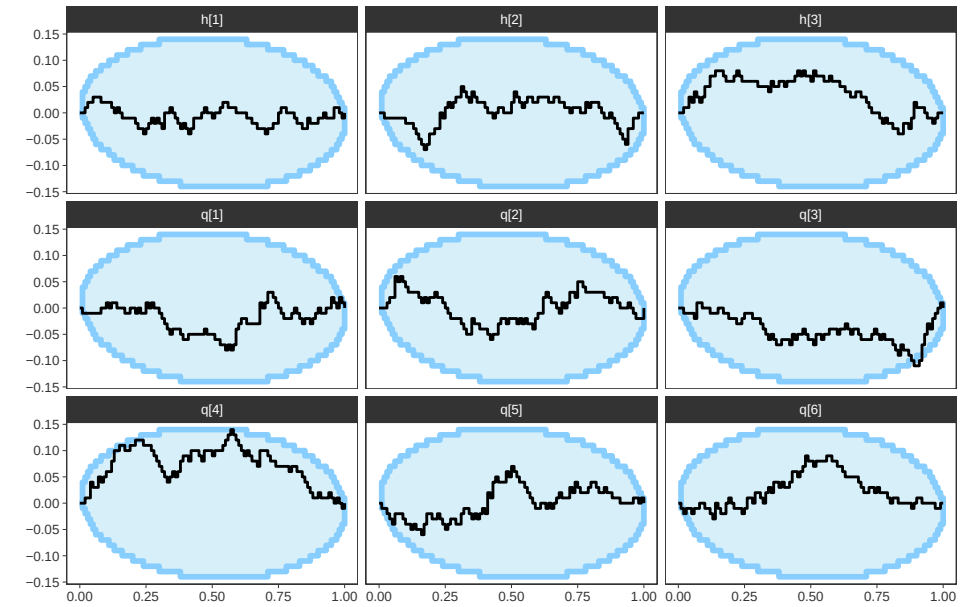
Single survey

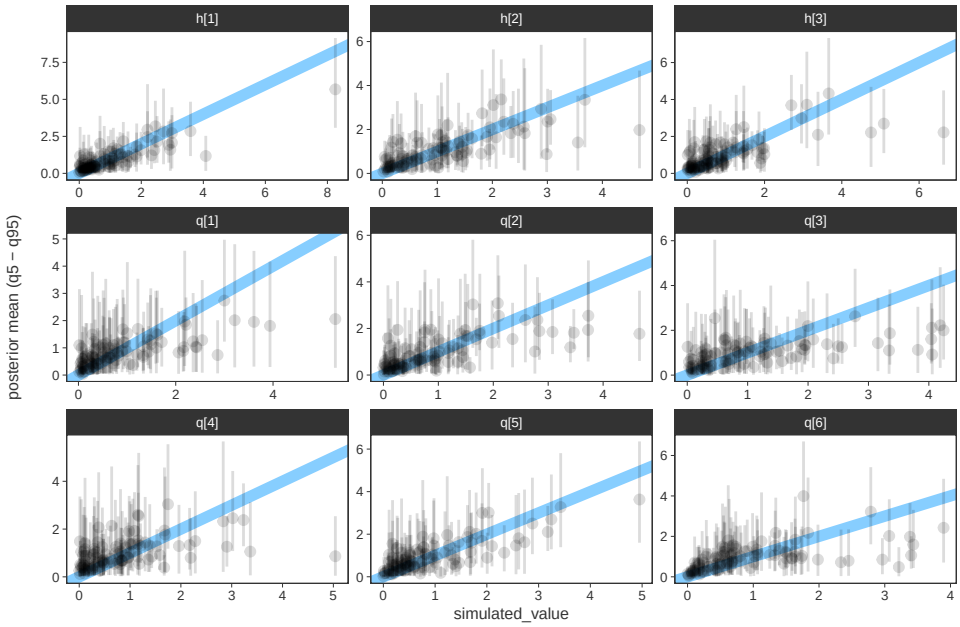
```
datasets <- SBC_generator_function(simulate_cmr,
                                   N_super = N_super,
                                   J = J,
                                   S = S) |>
  generate_datasets(n_sims)
backend <- SBC_backend_cmdstan_sample(cmdstan_model(here("stan/cjs-ms.stan")),
                                      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: 0.036 minutes.

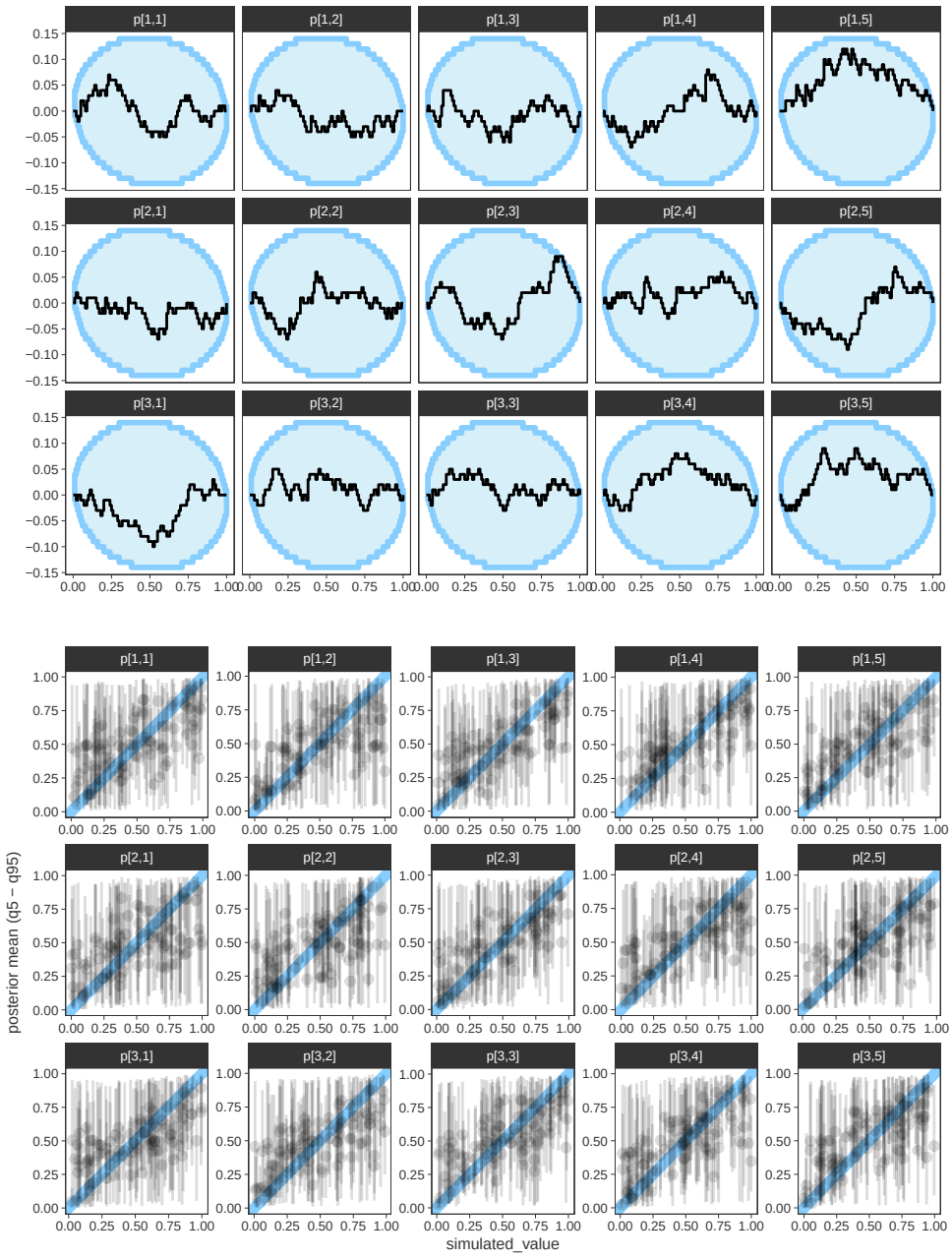
> **Warning**
>
> Single-survey multistate models with state-specific detection probabilities suffer from parameter identifiability issues in that they are confounded with the transition rates (see Hollanders & Royle 2022). Notice the difference in parameter estimates with the robust design version.

```
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)
```

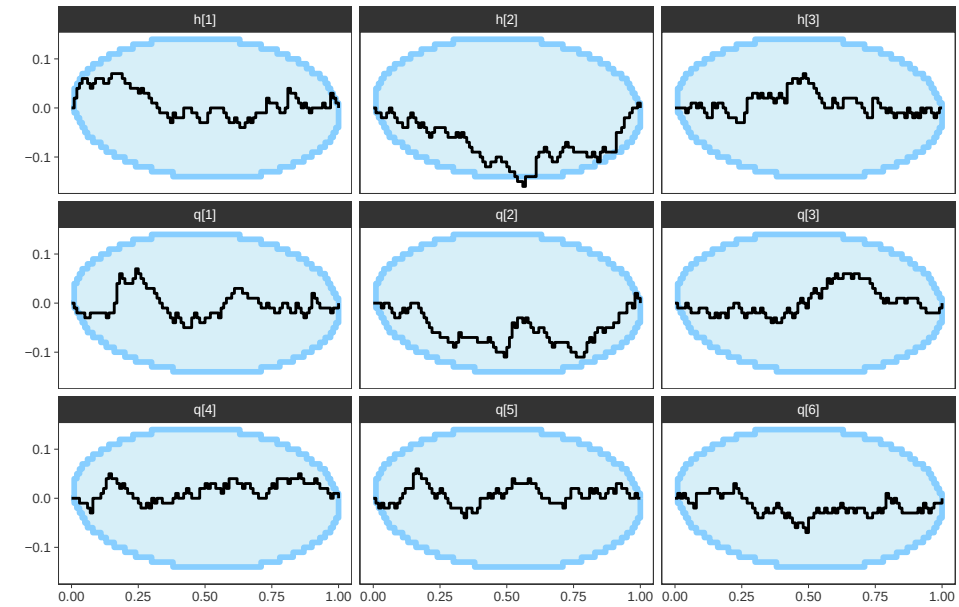


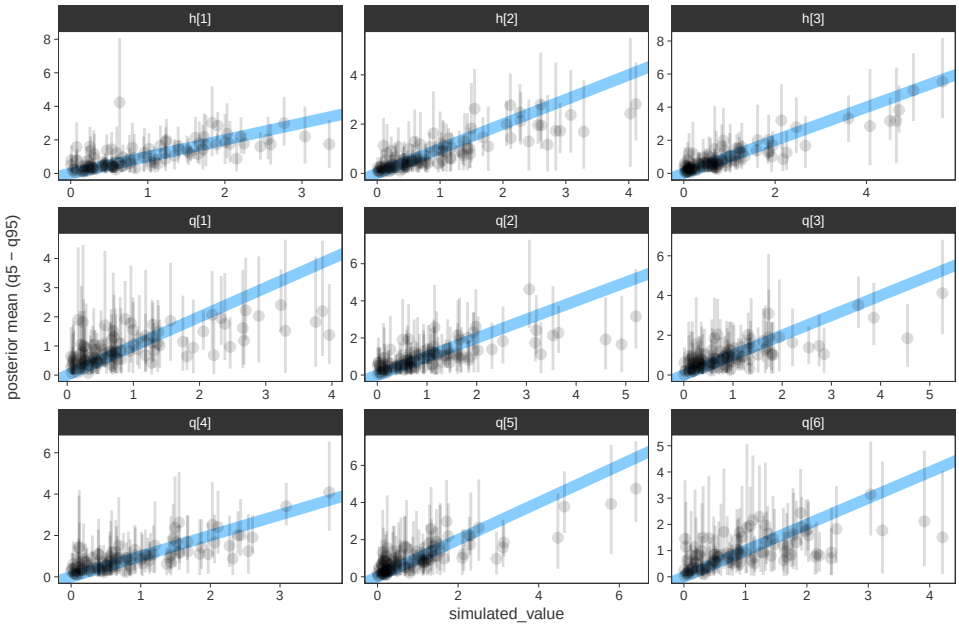
Robust design

```
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")),
                                      chains = chains,
                                      iter_warmup = iter_warmup,
                                      iter_sampling = iter_sampling)
sbc <- compute_SBC(datasets, backend, cores_per_fit = chains, keep_fits = F)
```

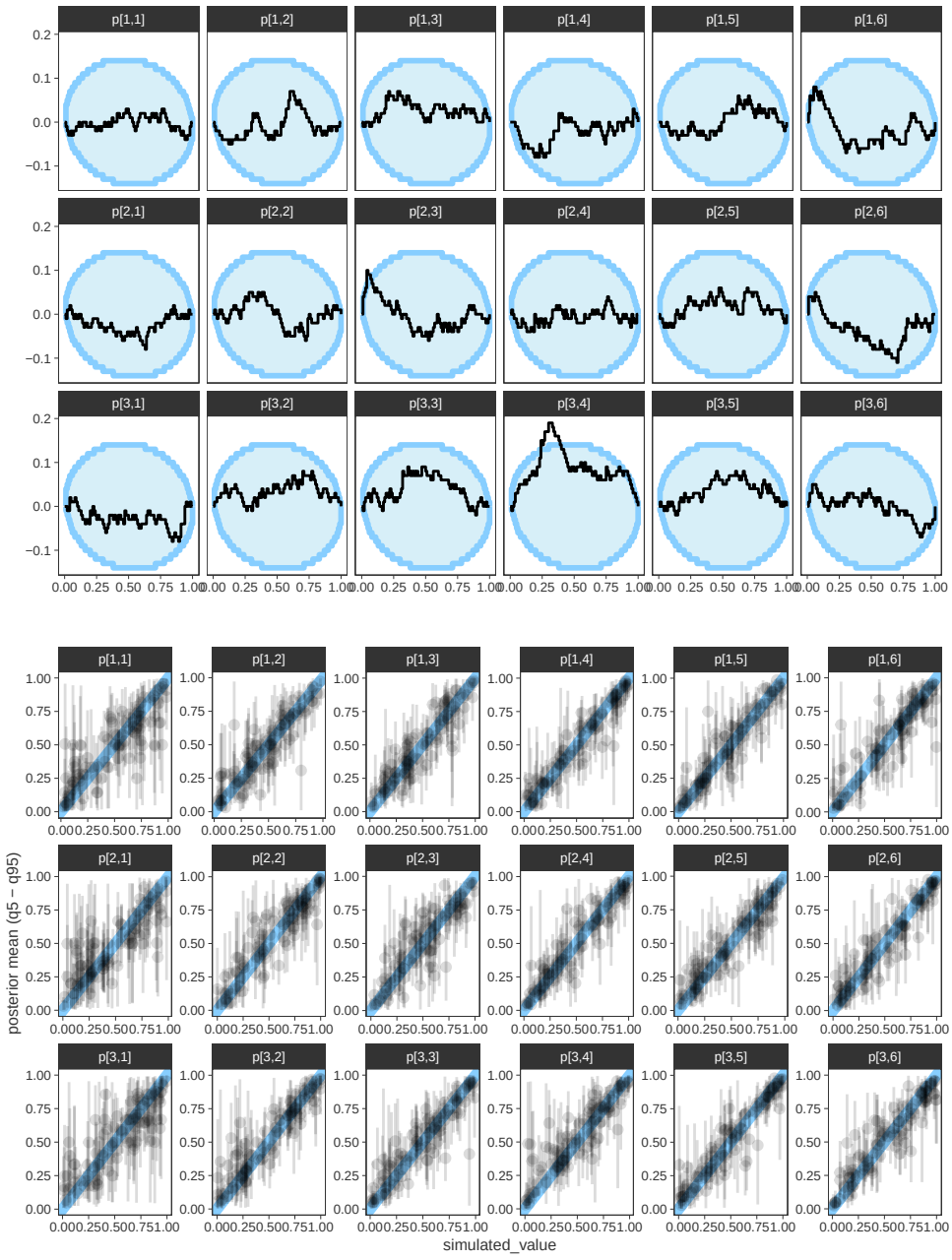
Mean time for the slowest of 8 chains with 700 iterations: 0.05 minutes.

```
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)
```



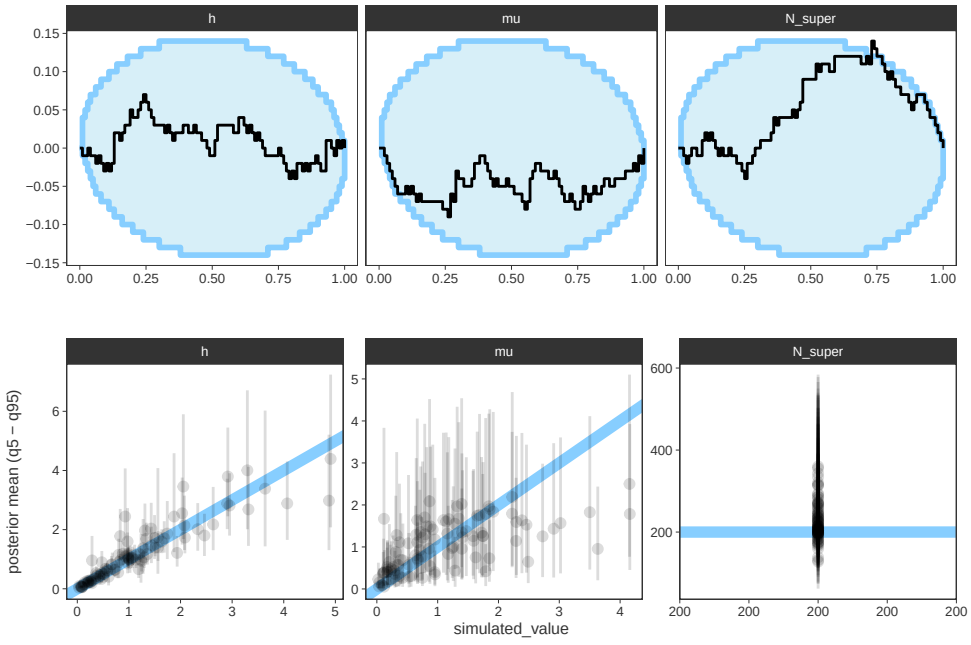
Jolly-Seber

Single survey

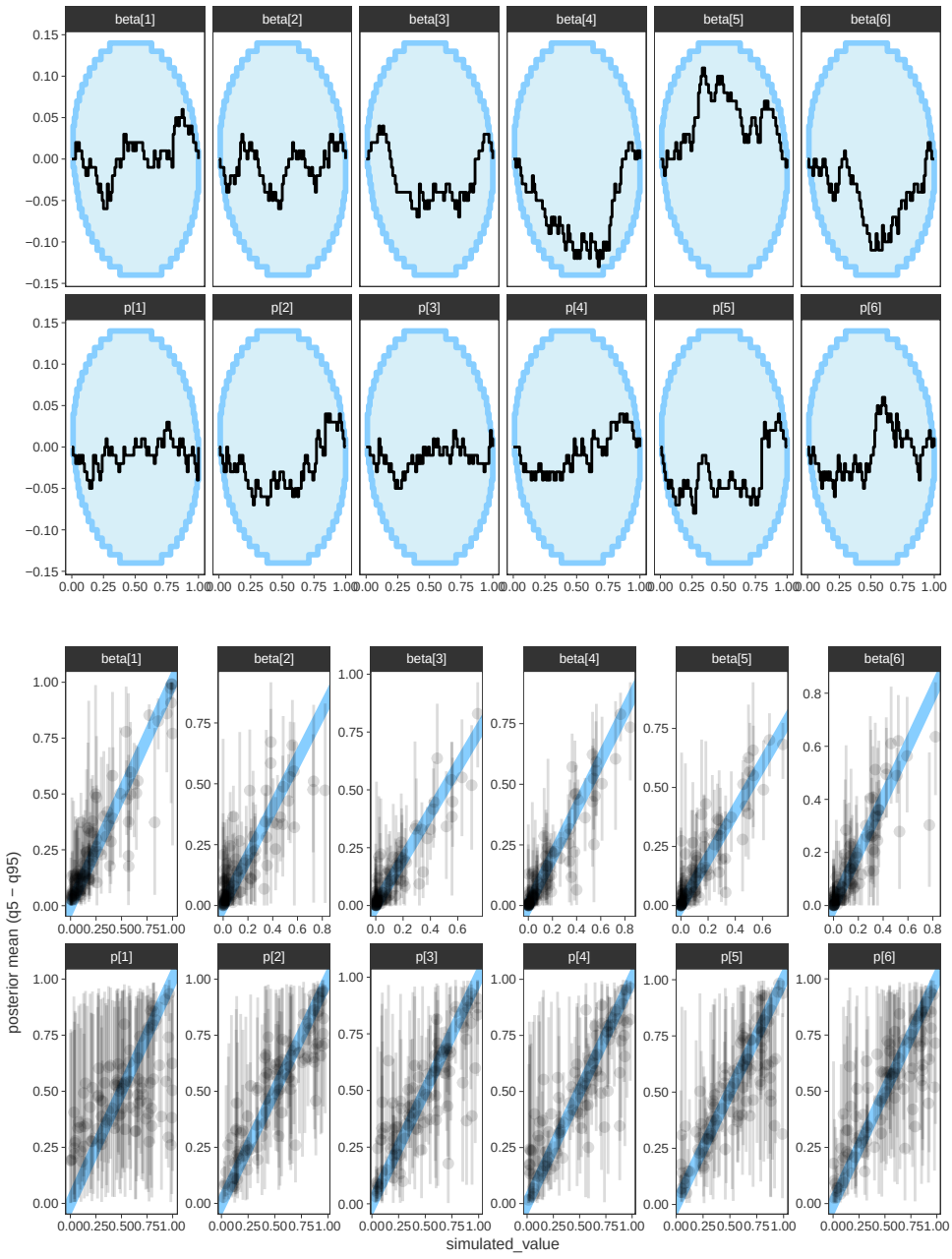
```
datasets <- SBC_generator_function(simulate_cmr,
                                   N_super = N_super,
                                   J = J,
                                   JS = T) |>
  generate_datasets(n_sims)
backend <- SBC_backend_cmdstan_sample(cmdstan_model(here("stan/js.stan")),
                                      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: 0.117 minutes.

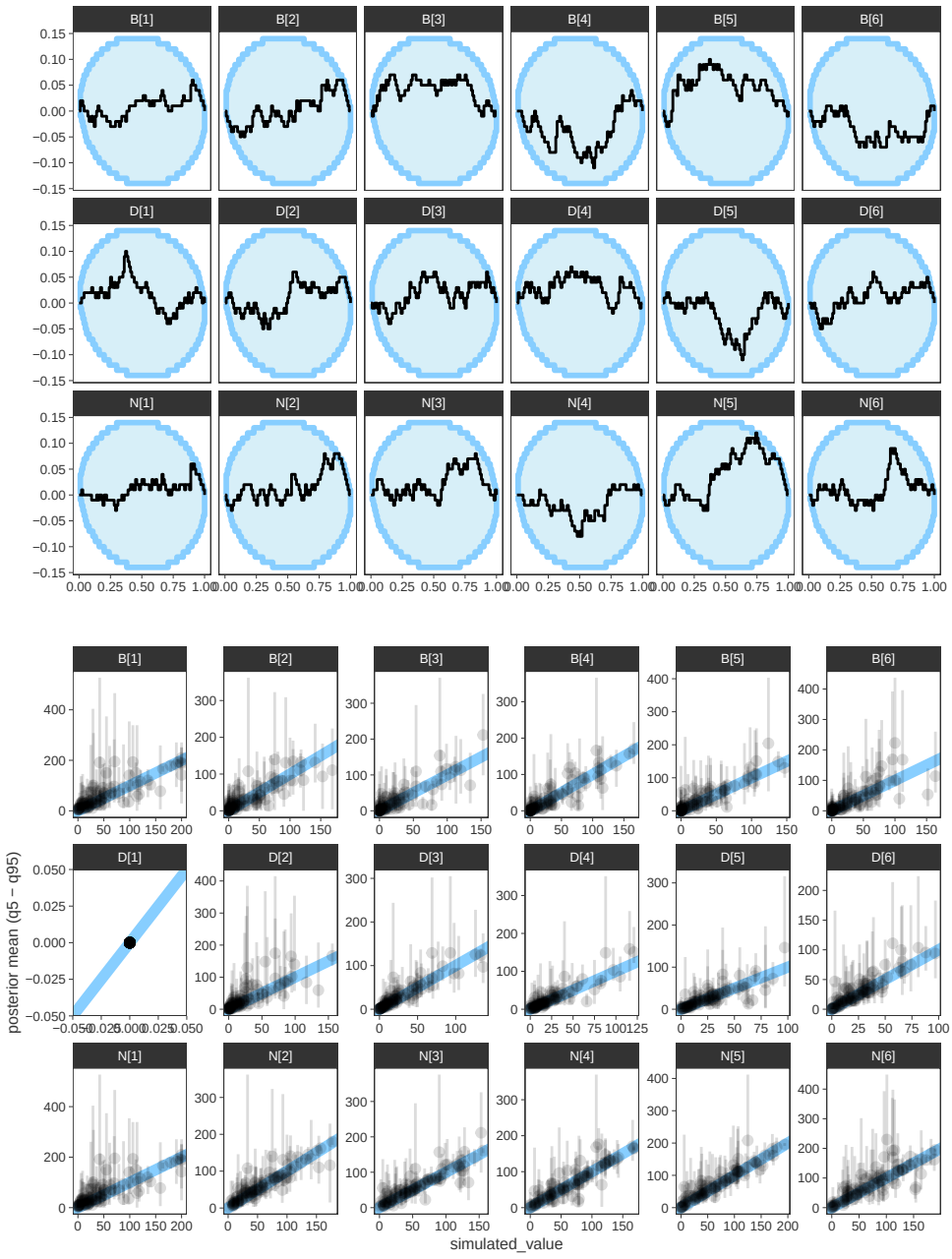
```
plot_sbc(sbc, c("h", "mu", "N_super"))
```



```
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)
```

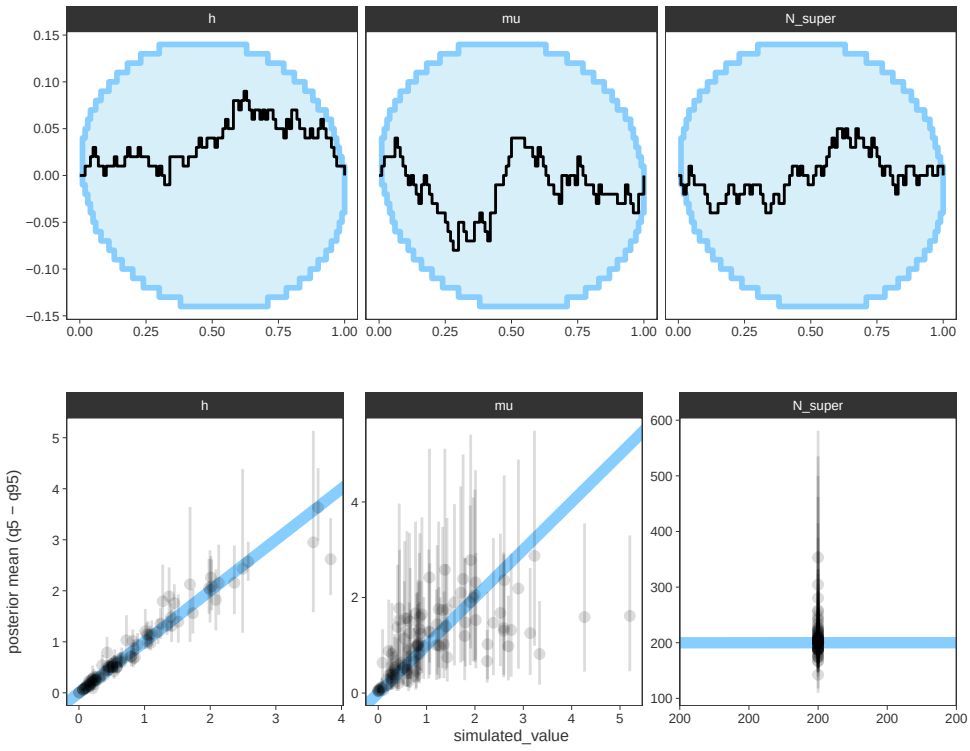


Robust design

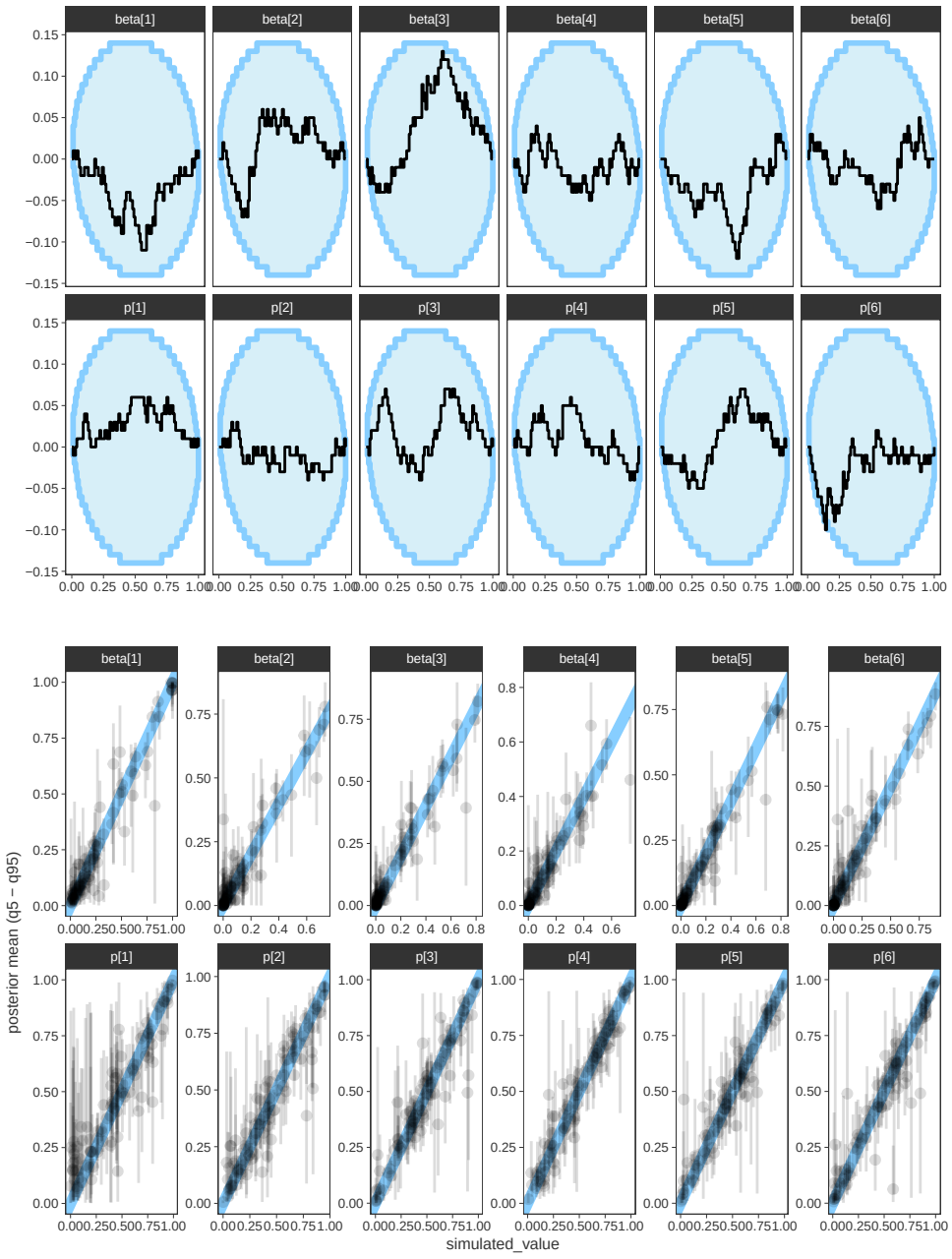
```
datasets <- SBC_generator_function(simulate_cmr,
                                   N_super = N_super,
                                   J = J,
                                   K = K,
                                   JS = T) |>
  generate_datasets(n_sims)
backend <- SBC_backend_cmdstan_sample(cmdstan_model(here("stan/js-rd.stan")),
                                      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: 0.173 minutes.

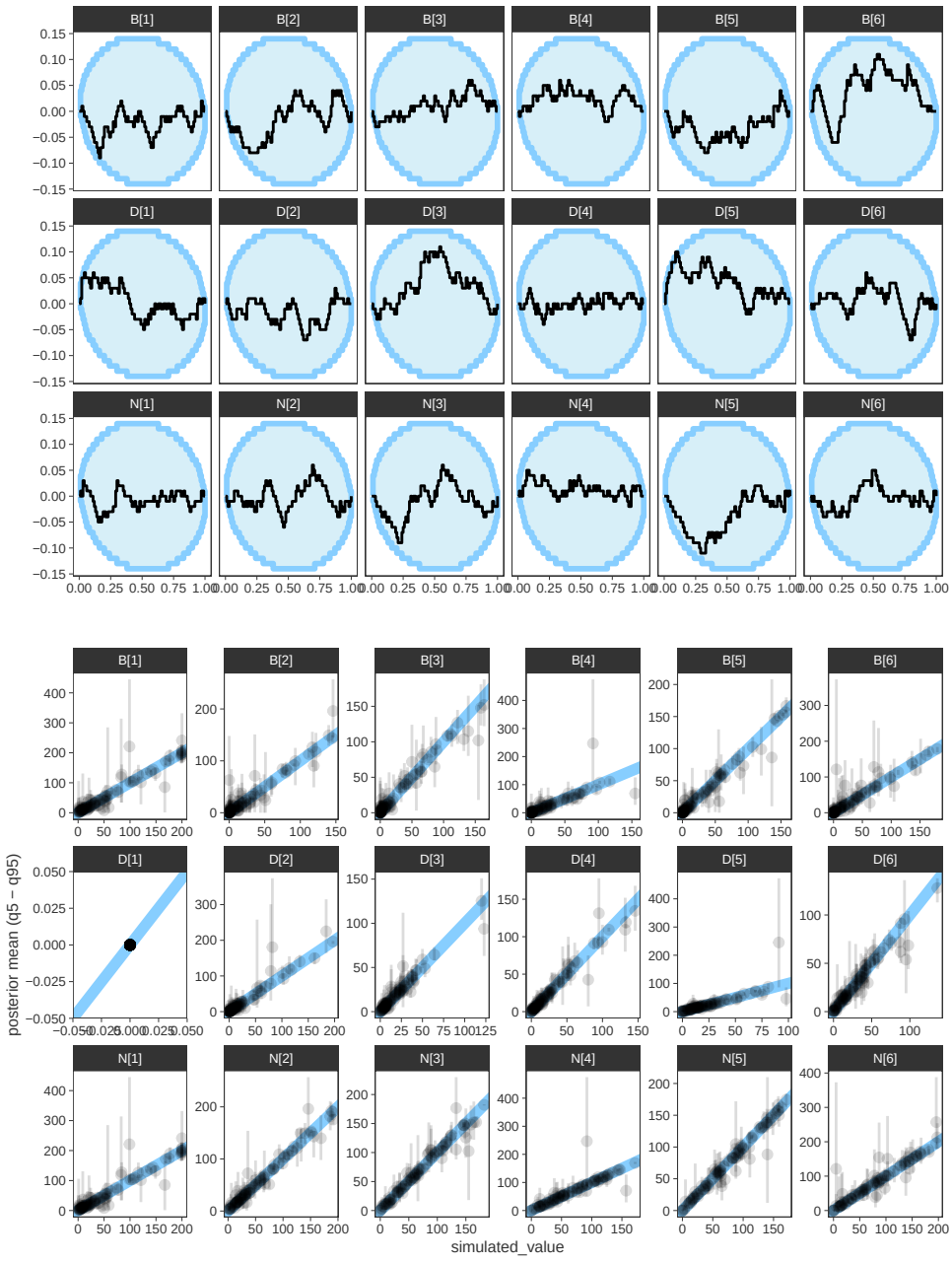
```
plot_sbc(sbc, c("h", "mu", "N_super"))
```



```
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)
```



Simulation script

```
simulate_cmr <- function(N_super = 200,
                         J = 8,
                         K = 1,
                         JS = 0,
                         I_aug = 500,
                         S = 1,
                         mu_gamma_prior = c(1, 1),
                         h_gamma_prior = c(1, 1),
                         p_beta_prior = c(1, 1),
                         eta_raw_gamma_prior = c(1, 1),
                         q_gamma_prior = c(1, 1)) {

 # metadata and parameters
 Jm1 <- J - 1
 tau <- rlnorm(Jm1)
 tau_scl <- tau / sum(tau) * Jm1
 mu <- rgamma(1, mu_gamma_prior[1], mu_gamma_prior[2])
 alpha <- c(1, mu * tau_scl)
 b_raw <- rgamma(J, alpha, 1)
 beta <- b_raw / sum(b_raw)
 b <- sort(rcat(N_super, beta))
 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)

 # single state simulation
 if (S == 1) {
   phi_tau <- exp(-h * tau)
   z <- matrix(0, N_super, J)
   y <- array(0, c(N_super, J, K))
   if (JS) {
     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 {
    f <- sample(1:J, N_super, replace = T, prob = colMeans(p))
    for (i in 1:N_super) {
      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])
        }
      }
    }
  }
  # multistate
} else {
  Sm1 <- S - 1 ; Sp1 <- S + 1
  eta_raw <- rgamma(S, eta_raw_gamma_prior[1], eta_raw_gamma_prior[2])
  eta <- eta_raw / sum(eta_raw)
  q <- rgamma(S * Sm1, q_gamma_prior[1], q_gamma_prior[2])
  P_z <- array(0, c(Jm1, Sp1, Sp1))
  for (j in 1:Jm1) {
    P_z[j, , ] <- expm::expm(rate_matrix(h, q) * tau[j])
  }
  z <- matrix(0, N_super, J)
  y <- array(0, c(N_super, J, K))
  if (JS) {
    B <- D <- matrix(0, S, J)
    for (i in 1:N_super) {
      z[i, b[i]] <- rcat(1, eta)
      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, P_z[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]
        }
      }
    }
  } else {
    f <- sample(1:J, N_super, replace = T, prob = colMeans(p))
    for (i in 1:N_super) {
      f_k <- sample(1:K, 1)
      z[i, f[i]] <- y[i, f[i], f_k] <- rcat(1, eta)
      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 (f[i] < J) {
        for (j in (f[i] + 1):J) {
          jm1 <- j - 1
          z[i, j] <- rcat(1, P_z[jm1, z[i, jm1], ])
          if (z[i, j] <= S) {
            y[i, j, ] <- rbinom(K, 1, p[z[i, j], j]) * z[i, j]
          }
        }
      }
    }
  }
}
obs <- which(rowSums(y) > 0)
I <- length(obs)
y <- y[obs, , ]

# output
if (K == 1 & !JS) {
  p <- p[1:S, -1]
```

```
  } else {
    p <- p[1:S, ]
  }
  variables <- list(h = h, p = p)
  generated <- list(I = I, J = J, tau = tau, y = y)
  if (K > 1) {
    generated <- append(generated,
                        list(K_max = K, K = rep(K, J)),
                        after = 2)
  }
  if (JS) {
    variables <- append(variables,
                        list(mu = mu,
                             beta = beta,
                             N_super = N_super,
                             B = B,
                             D = D,
                             N = colSums(z)),
                        after = 2)
    generated$I_aug <- I_aug
  }
  if (S > 1) {
    variables <- append(variables, list(q = q), after = 1)
    generated$S <- S
  }
  list(variables = variables, generated = generated)
}
```

References