Testing MCMC Samplers

Jason M.T. Roos

First European Bayesian Summit in Marketing

MCMC is awful

- The probability of any computer program having a bug is strictly positive
- Your MCMC sampler is a computer program

But my code is bug-free!!!!1

1. Lucky you
2. Bug-free MCMC code can fail to sample from the model due to
   - Non-identifiability
   - Pathological behavior of the sampling algorithm
   - Both of which are difficult to spot while coding

The crux of the problem

- We have a model,

$$p(\theta|y) \propto \pi(y|\theta)\pi(\theta),$$

  that we cannot evaluate directly

- We have a sampling procedure,

$$S : y \mapsto \theta_{1:L}, \quad \theta_{1:L} \equiv \{\theta_1, \dots, \theta_L\},$$

  implemented by people who are not trained to write software

- How do we know that
  - *[STRONG]* S is implemented correctly, or
  - *[WEAK]* S is probably not implemented incorrectly; and hence
  - $\theta_{1:L}$ is (approximately) distributed according to $p(\theta|y)$?

Agenda

1. Some theory about testing MCMC samplers
2. A detailed example showing how testing can be a valuable part of your workflow

Simple approach to testing

- Typical (among papers that even mention validating code)
  - Choose $\tilde{\theta}$ and call it the "true" value
  - Generate $\tilde{y} \sim \pi(y|\tilde{\theta})$ according to the data-generating process (DGP, i.e., likelihood)
  - Run the MCMC sampler to get $\theta_{1:L} := S(\tilde{y})$
  - If $\tilde{\theta}$ is *typical* of the $\theta_{1:L}$'s, then assume the program S is working as expected
    - What is "typical"? Maybe $\tilde{\theta}$ is within the 95% CI for $\theta_{1:L}$
- Because $p(\theta|y)$ can include values of θ with low probability, any individual simulation is not informative
  - Hence, choose a few of values of $\tilde{\theta}$ and show, on average, they are not different from their corresponding MCMC samples $\theta_{1:L}$

Exploiting the model structure

- Bayes models have a nice property that allows a better approach for choosing $\tilde{\theta}$ (Geweke 2004; Cook, Gelman, and Rubin 2006)
  1. Draw $\tilde{\theta} \sim \pi(\theta)$ from its prior distribution
  2. Simulate $\tilde{y} \sim \pi(y|\tilde{\theta})$ according to the DGP
  3. Run the sampler S to obtain MCMC samples $\theta_{1:L} := S(\tilde{y})$
- The MCMC samples $\theta_{1:L}$ are draws from the *prior* distribution $\pi(\theta)$ because

$$\pi(\theta) = \int \underbrace{\pi(\theta|\tilde{y})}_{\text{Step 3}} \underbrace{\pi(\tilde{y}|\tilde{\theta})}_{\text{Step 2}} \underbrace{\pi(\tilde{\theta})}_{\text{Step 1}} \, d\tilde{y} \, d\tilde{\theta}$$

Comparing MCMC samples to the true prior

- Discrepancy between the "true" $\tilde{\theta}$'s and their corresponding MCMC samples $\theta_{1:L}$ suggests a problem with S
- Use discrete rank statistics,

$$r(\theta_{1:L}, \tilde{\theta}) = \sum_{\ell=1}^{L} \mathbb{1}(\theta_\ell < \tilde{\theta}) \in [0, L],$$

to test for such discrepancies

Theorem 1 (Talts et al. 2018).

Let $\tilde{\theta} \sim \pi(\theta)$, $\tilde{y} \sim \pi(y|\tilde{\theta})$, and $\{\theta_1, \ldots, \theta_L\} \sim \pi(\theta|\tilde{y})$ for any distribution $\pi(y, \theta)$. The rank statistics of any one-dimensional random variable over θ is uniformly distributed over the integers $[0, L]$.

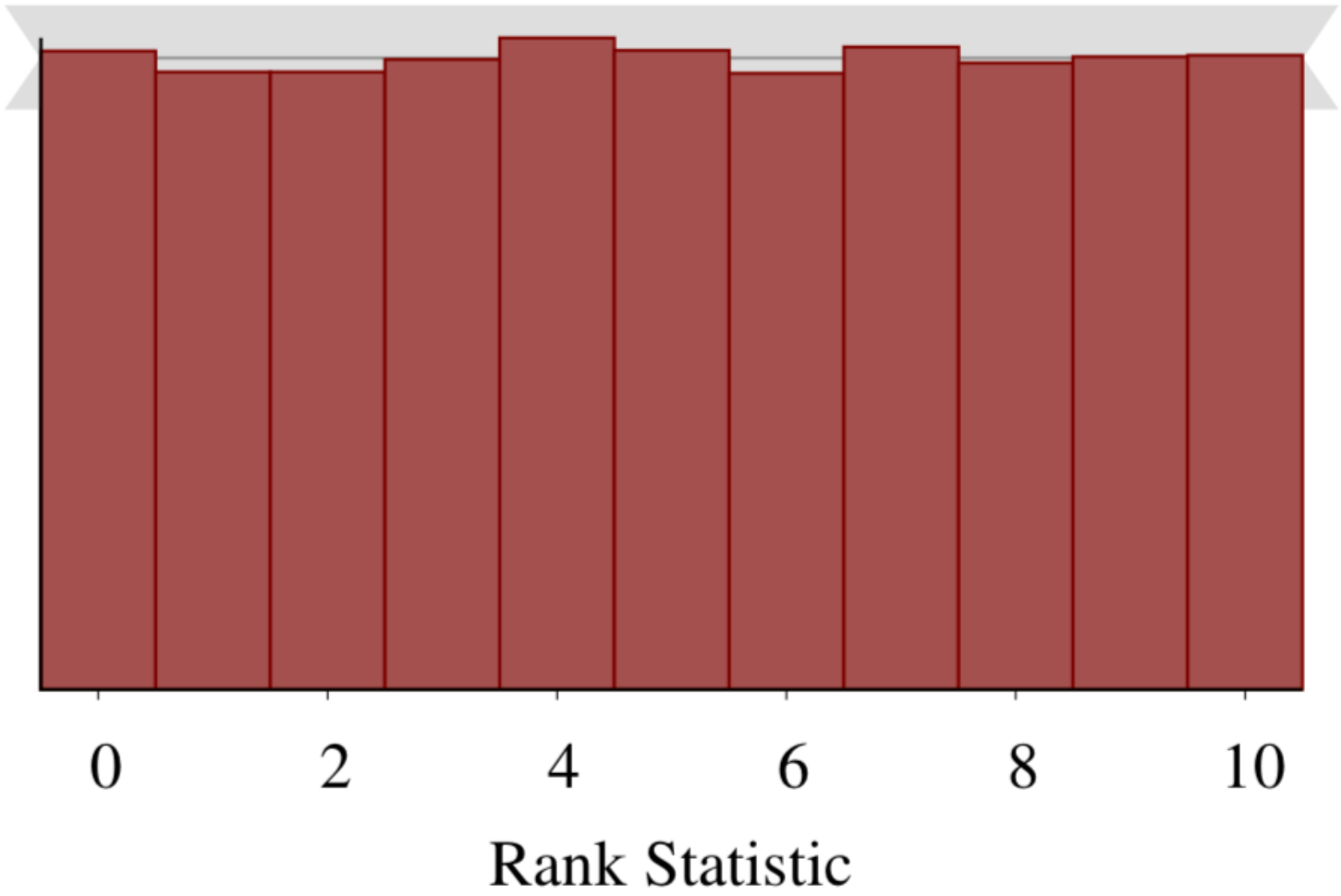
Testing procedure for a one-dimensional parameter θ

- Initialize a histogram H with $L+1$ bins
- Conduct N replications of the following:
  1. Draw $\tilde{\theta} \sim \pi(\theta)$ from its prior distribution
  2. Simulate $\tilde{y} \sim \pi(y|\tilde{\theta})$ according to the DGP
  3. Run the sampler S to obtain MCMC samples $\theta_{1:L} := S(\tilde{y})$
  4. Set $r := r(\theta_{1:L}, \tilde{\theta})$ and increment the bin H_r by 1
- The bin H_0 counts the number of times $r = 0$, H_1 counts the number of times $r = 1$, etc.
- The counts in the $L+1$ bins are distributed approximately $H_r \sim Bin(N, (L+1)^{-1})$

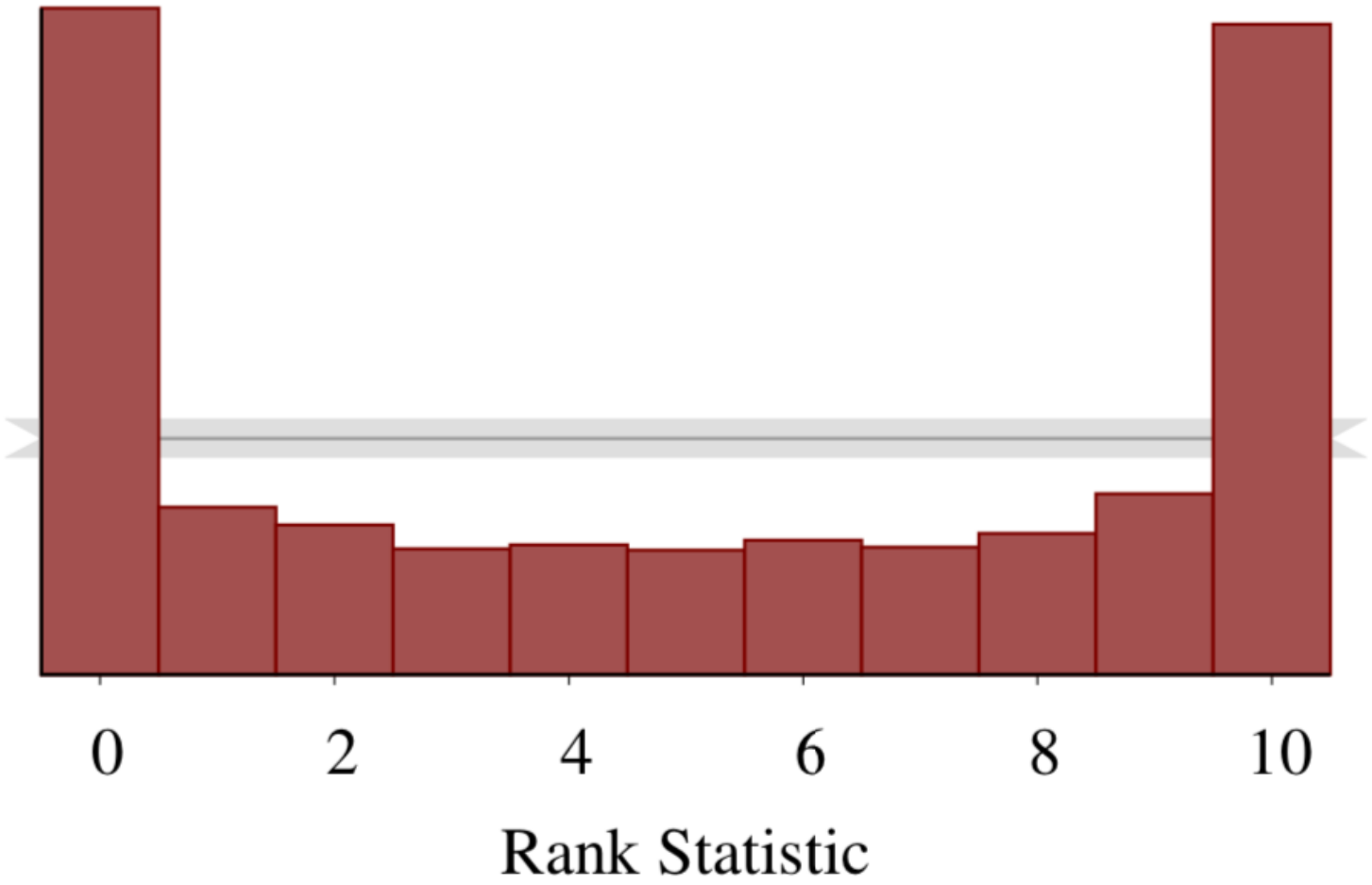
Testing procedure for many parameters

- Same procedure as before
- Each of the p (one-dimensional) elements $\tilde{\theta}_p \in \tilde{\theta}$ can have its own histogram
- Or pool all ranks for all p elements $\tilde{\theta}_p$ into a single histogram
  - Adjust the expected binomial distribution accordingly

What you expect to see if everything is working



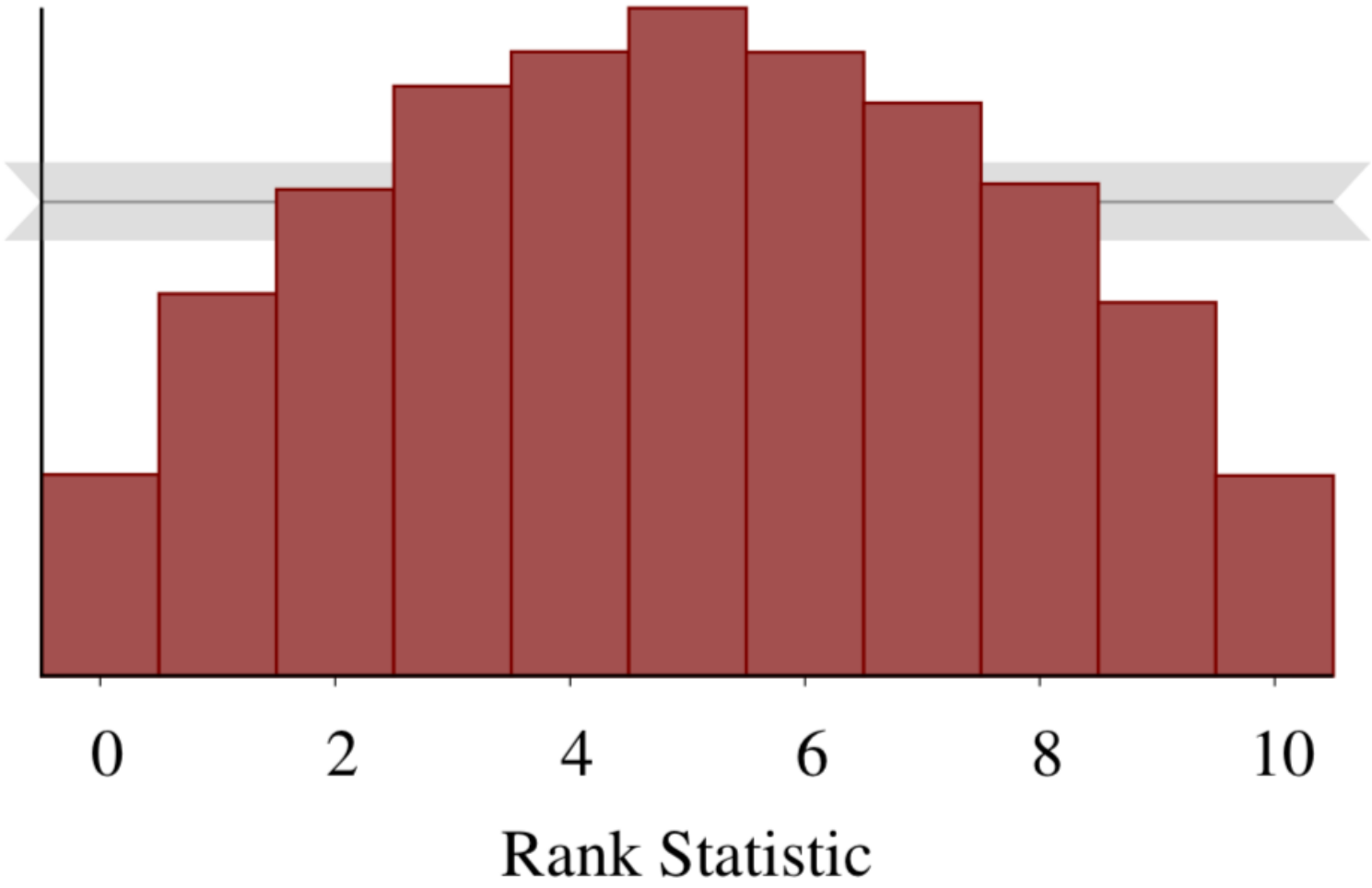
If your chain is badly autocorrelated



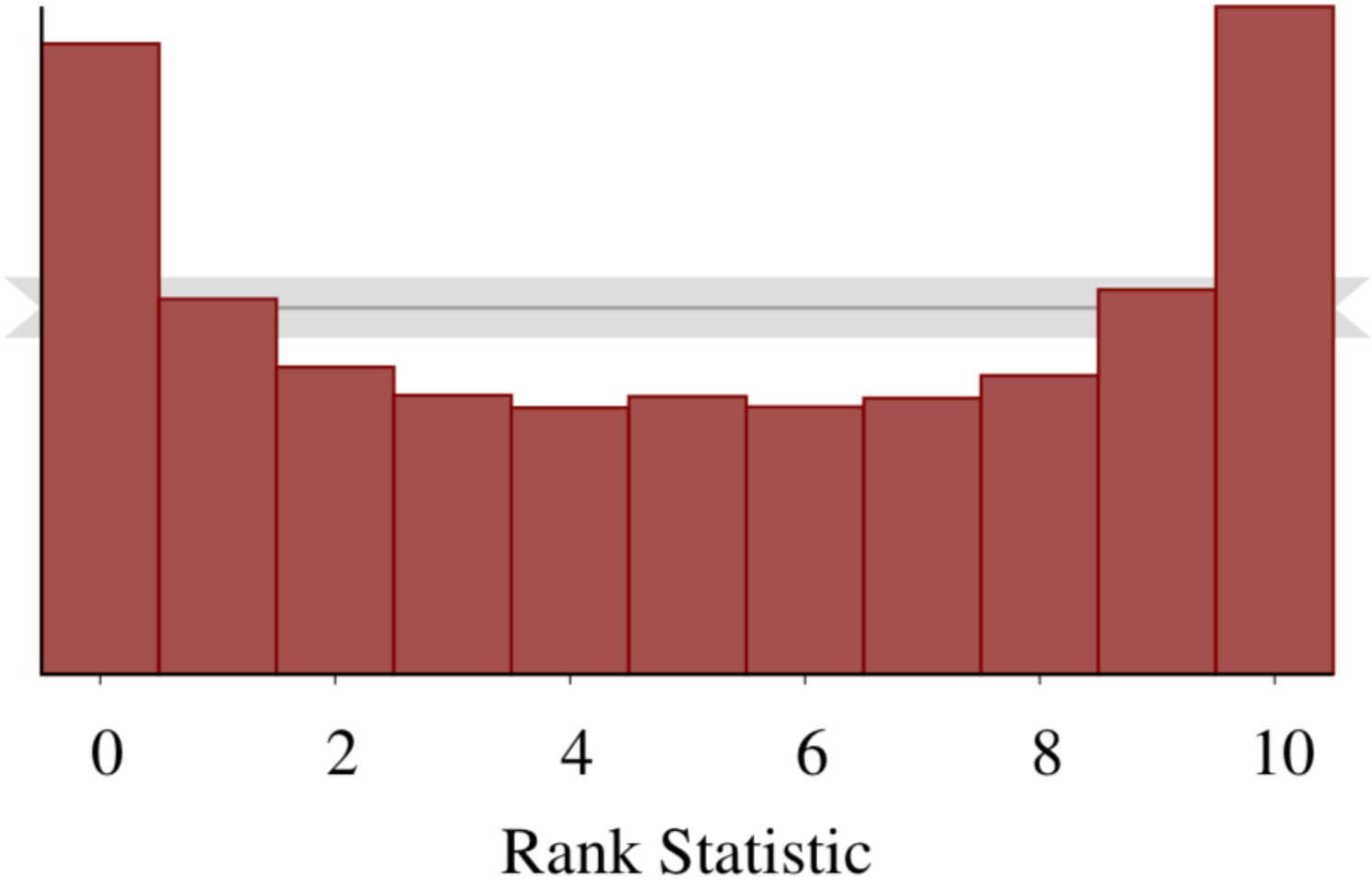
Regarding autocorrelation

- Autocorrelation prevents us from assessing if S is working
- Hence, for each of the N replications
  1. Generate an MCMC chain $\theta_{1:M}$ of length $M \geq L$
  2. If $ESS(\theta_{1:M}) < L$
     - Increase the value of M and return to step 1
  3. Else
     - Thin the chain to L iterations, or
     - Assign the M rank values to $L + 1$ bins (easy if $M \mod (L + 1) = 0$)

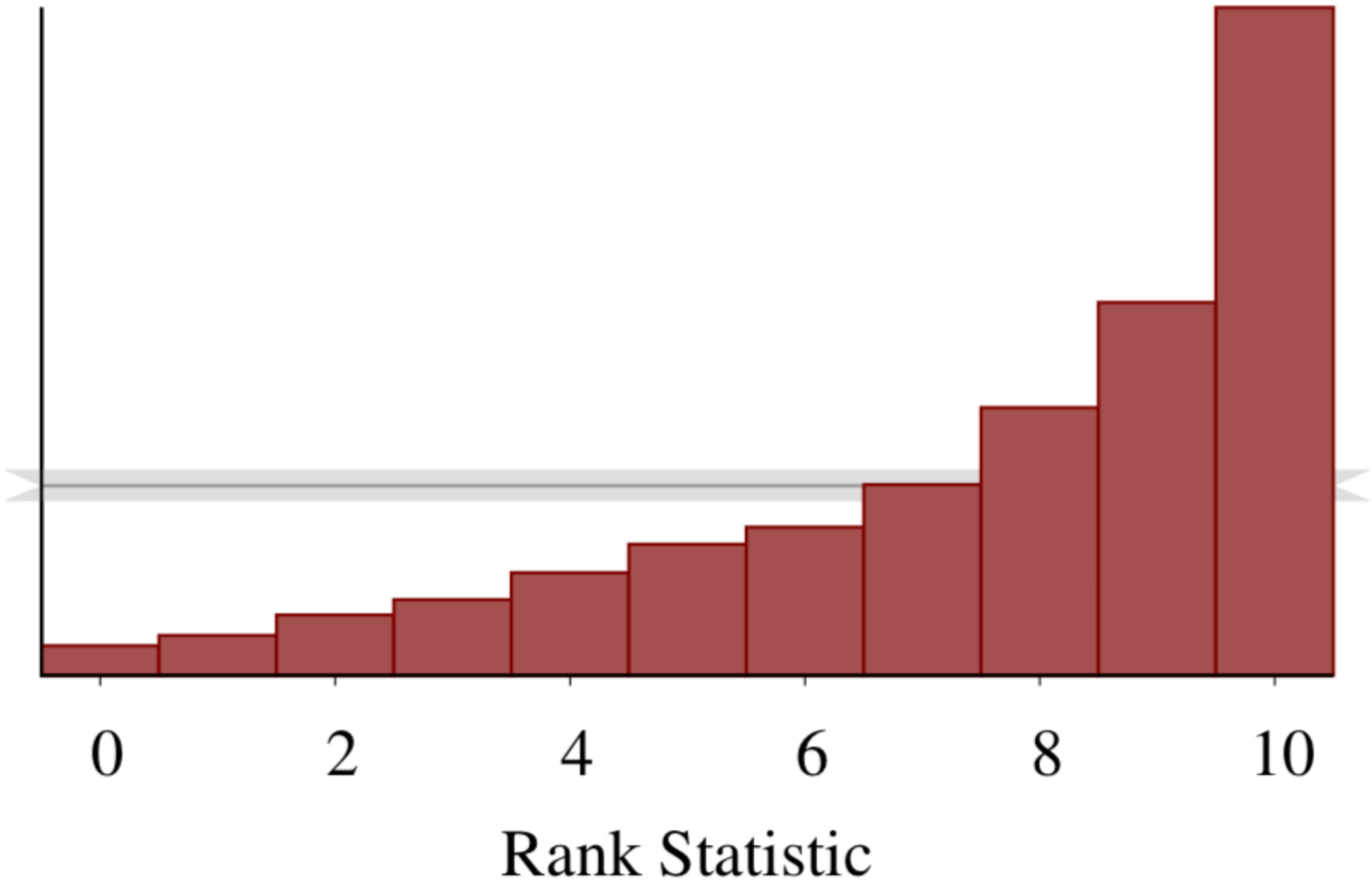
If the chain is overdispersed relative to true prior



If the chain is underdispersed relative to true prior



If the chain is biased downward relative to true prior



Continuous testing

- Primary goal
  - Satisfy the weak claim that S is probably not implemented incorrectly, and hence $\theta_{1:L}$ is (approximately) distributed according to $p(\theta|y)$
- Nice side effects from continuously testing MCMC code
  - Understand the model's structure and the implications of that structure for data
  - Catch bugs as soon as they are introduced, saving time and embarassment

Workflow

1. Write code to draw samples $\tilde{\theta} \sim \pi(\theta)$
2. Write code to simulate from the DGP $\tilde{y} \sim \pi(y|\tilde{\theta})$
   - *Congratulations:* You just wrote the code for the counterfactual simulations
   - *Fun fact:* $\tilde{y}$ is a sample from the model's prior predictive distribution of y. Does $\tilde{y}$ look reasonable compared to real data? Is it over- or under-dispersed?
3. Write the MCMC program S to generate posterior samples from $p(\theta|y)$
4. Test and debug until the weak claim is satisfied
   - *Congratulations:* You just pre-empted R2 by figuring out, e.g., that only $p - 2$ fixed effects are identified in your model

Workflow (2)

5. You now have two sets of code for the same model, but which are very different from one another
   - One samples from the prior and generates data
   - The other takes data and generates posterior samples
   - *Fun fact:* It is very likely that one or both of these will have bugs that you will need to fix
   - *Funner fact:* It is very unlikely that you will introduce a bug into one that exactly cancels out a bug introduced into the other
     - Unless you copy/paste code

A "simple" direct-utility model

- One good, standard price per unit
- Consumer chooses x number of units in period t to maximize utility subject to a budget constraint

$$u(x_t) = \frac{\alpha e^{\epsilon_t}}{\gamma} \log(\gamma x_t + 1)$$
$$u_0(x_t) = M_t - p_t x_t$$
$$\epsilon_t \sim N(0, \sigma^2)$$

- Prior and DGP implemented in R
- MCMC Sampler implemented in Stan
- Simulation and rank calculations implemented in R (`sbcrs`)

Sampling routines

1. Forward simulation from the DGP
   - Generate fake (exogenous) data
   - Sample parameters from their prior distribution
   - Simulate choices from the DGP
     - Loop over choice occasions t
     - Calculate utility for each feasible quantity x
     - Record the choice x_t yielding the highest utility
2. MCMC sampling
   - Likelihood is rather messy, likely to have bugs…

Likelihood

$$L(x|\alpha,\gamma,\sigma) = \prod_t \ell(x_t)$$

$$\ell(x_t) = \Phi\left(\frac{ub_t(x_t)}{\sigma}\right) - \Phi\left(\frac{lb_t(x_t)}{\sigma}\right)$$

$$lb_t(x_t) \equiv \begin{cases} \log\left(\frac{p_t\gamma}{\alpha}\right) - \log\left[\log\left(\frac{\gamma x_t+1}{\gamma(x_t-1)+1}\right)\right], & x_t > 0 \\ -\infty, & x_t = 0 \end{cases}$$

$$ub_t(x_t) \equiv \begin{cases} \log\left(\frac{p_t\gamma}{\alpha}\right) - \log\left[\log\left(\frac{\gamma(x_t+1)+1}{\gamma x_t+1}\right)\right], & (x_t+1)p_t \leq M_t \\ \infty, & (x_t+1)p_t > M_t \end{cases}$$

Implementation in R

```
library(tidyverse)
library(rstan)
library(sbcrs)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
```

Generate fake (exogenous) data

```
gen_data <- function(seed) {
  set.seed(1e6 + seed)

  T <- 10
  M <- rep(10, T) + sample.int(3, T, replace = TRUE) - 1L
  p <- rep(.5, T)

  pcer <- function(U, a) (-log(a) / U)
  sigma_rate <- pcer(1, .1)
  alpha_rate <- pcer(1, .5)
  gamma_rate <- pcer(1, .1)

  list(
    T = T, p = p, M = M, sigma_rate = sigma_rate,
    alpha_rate = alpha_rate, gamma_rate = gamma_rate
  )
}
```

Stan data

```
data {
  // Number of decisions / observations
  int<lower = 0> T;

  // Prices and available budgets
  real<lower = 0> p[T];
  real<lower = 0> M[T];

  // Hyperparameters
  real<lower = 0> sigma_rate;
  real<lower = 0> alpha_rate;
  real<lower = 0> gamma_rate;
  ...
}
```

Sample from prior distribution of model parameters

```
gen_params <- function(seed, data) {
  set.seed(2e6 + seed)

  sigma_raw <- rgamma(1, 2, rate = .5)
  alpha_raw <- rexp(1)
  gamma_raw <- rexp(1)

  list(
    sigma_raw = sigma_raw, alpha_raw = alpha_raw,
    gamma_raw = gamma_raw
  )
}
```

Stan parameters and priors

```
parameters {
  // Parameters on a natural unit scale
  real<lower = 0> sigma_raw;
  real<lower = 0> alpha_raw;
  real<lower = 0> gamma_raw;
}
model {
  // Prior distributions for parameters on unit scale
  sigma_raw ~ gamma(2, .5);
  alpha_raw ~ exponential(1);
  gamma_raw ~ exponential(1);
  ...
}
```

Simulate from the DGP: Utility function

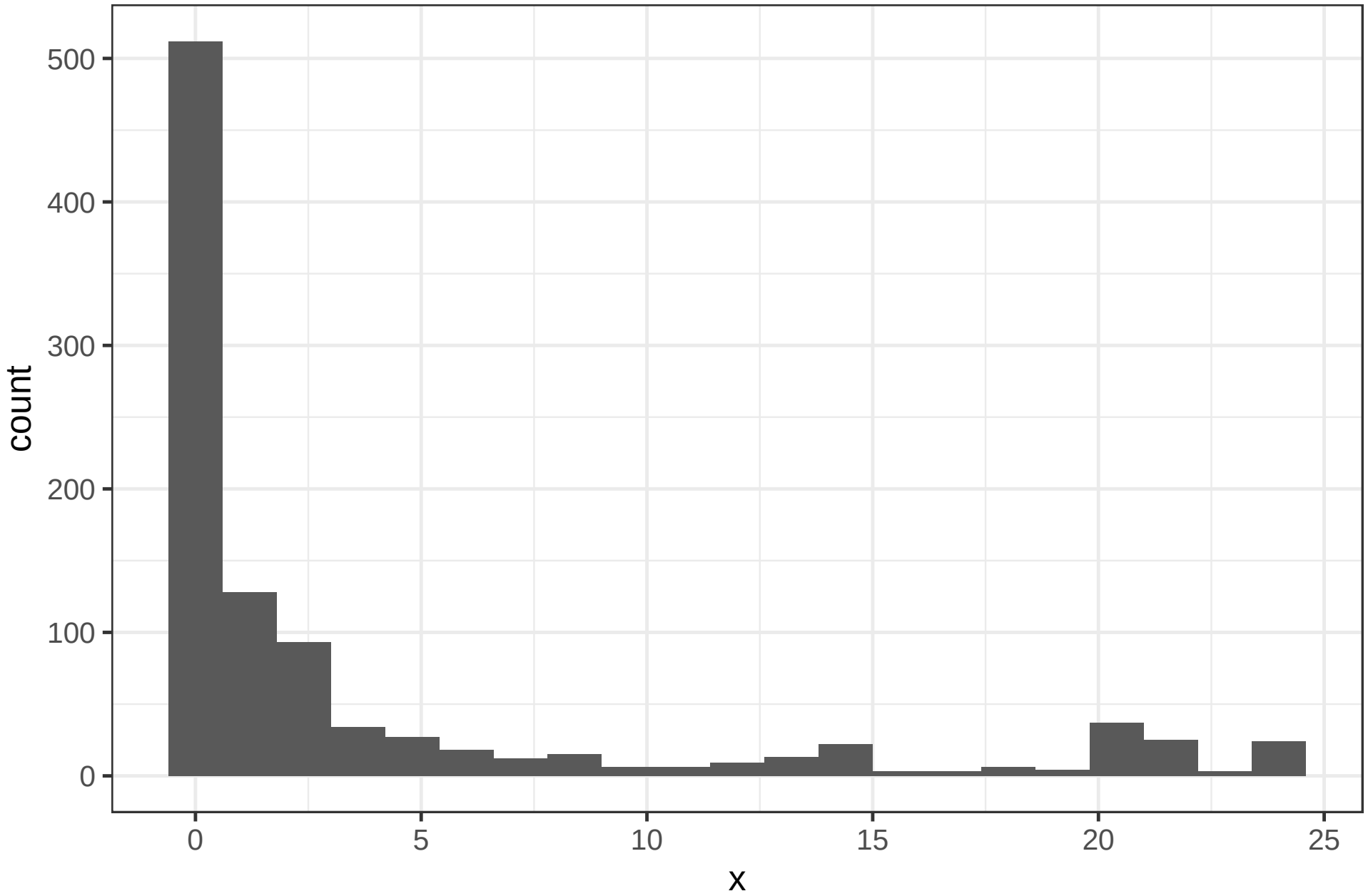
```
Ux <- function(x, alpha, gamma, epsilon_t, M_t, p_t) {
  u <- alpha * exp(epsilon_t) / gamma * log1p(gamma * x) +
    M_t - p_t * x
  u[M_t < p_t * x] <- -Inf
  u
}
```

Simulate from the DGP: Choices

```
gen_modeled_data <- function(seed, data, params) {
  set.seed(3e6 + seed)

  sigma <- params$sigma_raw / data$sigma_rate
  alpha <- params$alpha_raw / data$alpha_rate
  gamma <- 1 + params$gamma_raw / data$gamma_rate
  epsilon <- rnorm(data$T, 0, 1) * sigma
  optx <- function(alpha, sigma, gamma, epsilon_t, M_t, p_t) {
    which.max(Ux(
      seq_len(floor(M_t / p_t) + 1) - 1,
      alpha, gamma, epsilon_t, M_t, p_t
    )) - 1L
  }
  x <- map_dbl(
    seq_len(data$T),
    ~ optx(alpha, sigma, gamma, epsilon[.x], data$M[.x], data$p[.x])
  )
  list(x = x)
}
```

Detour: Visualizing the prior predictive distribution

```
prior_predictive_distribution <- function(seed) {
  data <- gen_data(seed)
  params <- gen_params(seed, data)
  gen_modeled_data(seed, data, params)
}
seq_len(100) %>%
  map(~ prior_predictive_distribution(.x)) %>%
  unlist() %>%
  as.vector() %>%
  tibble(x = .) %>%
  ggplot(aes(x = x)) +
  stat_bin(bins = 21) +
  theme_bw()
```


500
400
300
200
100
0
count
0
5
10
15
20
25
x

Stan transformed parameters

```
transformed parameters {
  // Parameters on the model scale
  real sigma;
  real alpha;
  real gamma;

  sigma = sigma_raw / sigma_rate;
  alpha = alpha_raw / alpha_rate;
  gamma = 1 + gamma_raw / gamma_rate;
}
```

Stan dependent data

```
data {
  ...
  // Observed quantities
  real<lower = 0> x[T];

  // Indicators for cases when lb = -Inf or ub = Inf
  int <lower = 0, upper = 1> no_lb[T];
  int <lower = 0, upper = 1> no_ub[T];
}
```

Stan likelihood: upper and lower bound calculation

```
functions {
  // calculate upper and lower bounds given x^+ and x or x and x^- and p
  // given dp, the marginal price per unit of x
  real bound(real ux, real lx, real dp, real gamma, real alpha) {
    return log(dp * gamma) - log(alpha)
         - log(log1p(gamma * ux) - log1p(gamma * lx));
  }
}
```

Stan likelihood: choices (1)

```
model {
  ...
  // Likelihood calculation, iterating over each observation
  for (t in 1:T) {
    real ub;
    real lb;

    // Determine lower bound (if any)
    if (no_lb[t]) {
      lb = negative_infinity();
    } else {
      lb = bound(x[t], x[t] - 1, p[t], gamma, alpha);
    }

    // Determine upper bound (if any)
    if (no_ub[t]) {
      ub = positive_infinity();
    } else {
      ub = bound(x[t] + 1, x[t], p[t], gamma, alpha);
    }
    ...
```

Stan likelihood: choices (2)

```
    ...
    // Three cases for the likelihood calculation.
    // 1. There is no lower bound but there is an upper bound
    //    The likelihood is F(ub) - F(-Inf) <=> F(ub) - 0 <=> F(ub)
    if (no_lb[t]) {
      target += normal_lcdf(ub | 0, sigma);
    }
    // 2. There is no upper bound but there is a lower bound
    //    The likelihood if F(Inf) - F(lb) <=> 1 - F(lb)
    else if (no_ub[t]) {
      target += normal_lccdf(lb | 0, sigma);
    }
    // 3. There are both upper and lower bounds
    //    The likelihood is F(ub) - F(lb)
    else {
      real Fu;
      real Fl;
      Fu = normal_lcdf(ub | 0, sigma);
      Fl = normal_lcdf(lb | 0, sigma);
      target += log_diff_exp(Fu, Fl);
    }
  }
}
```

Sampling from Stan

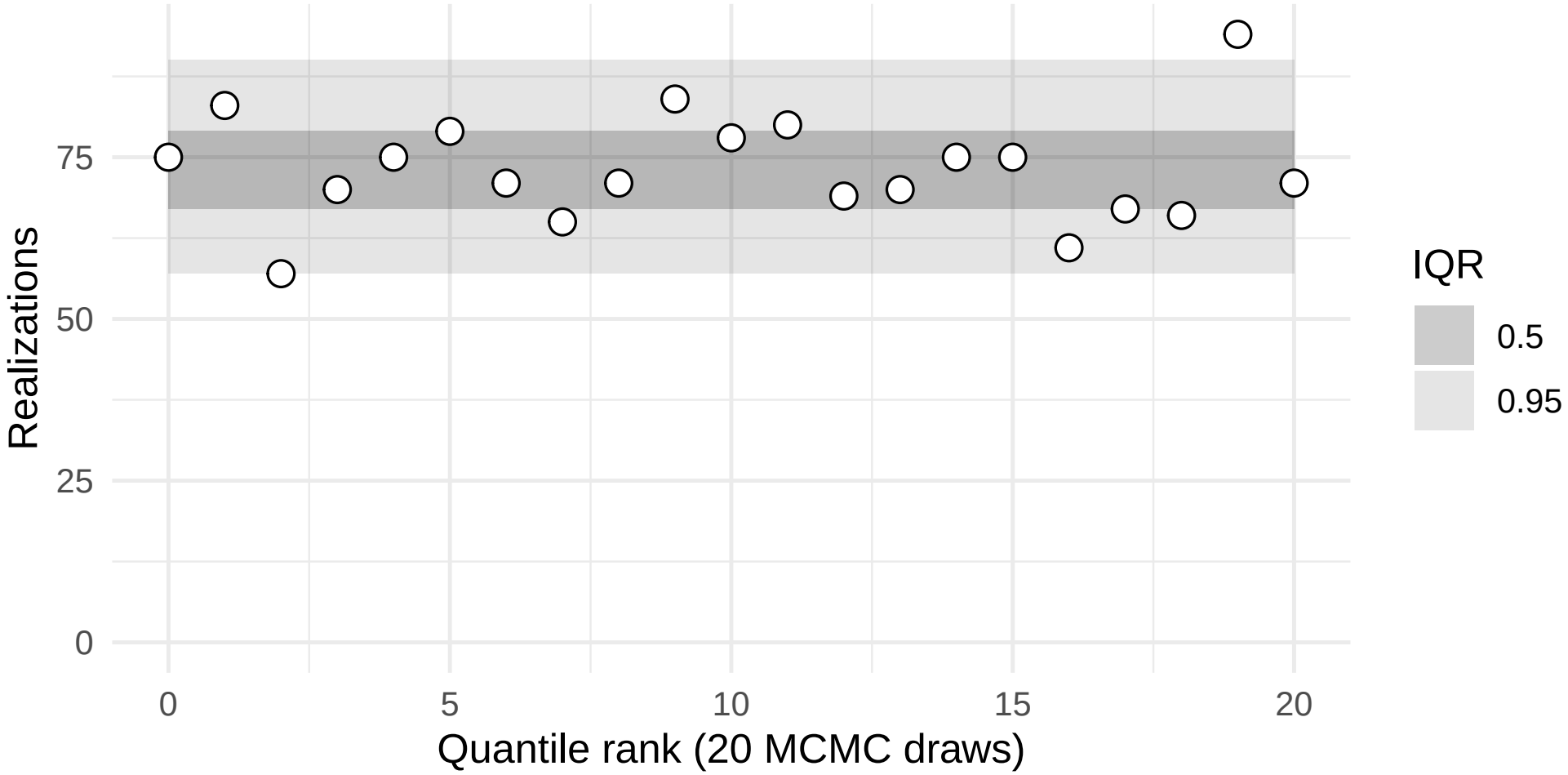
```
the_model <- stan_model("model1.stan", save_dso = TRUE)
sample_from_stan <- function(seed, data, params, modeled_data, iters) {
  modeled_data$x <- as.array(modeled_data$x)
  data$p <- as.array(data$p)
  data$M <- as.array(data$M)
  data$no_lb <- as.array(1L * (modeled_data$x == 0))
  data$no_ub <- as.array(1L * ((modeled_data$x + 1) * data$p > data$M))

  data_for_stan <- c(data, modeled_data)
  sampling(the_model,
    data = data_for_stan, seed = seed,
    chains = 1, iter = 2 * iters, warmup = iters,
    open_progress = FALSE, show_messages = FALSE,
    refresh = 1000
  )
}
```

Testing

```
doParallel::registerDoParallel(
  cores = parallel::detectCores()
)
sbc <- SBC$new(
  data = gen_data,
  params = gen_params,
  modeled_data = gen_modeled_data,
  sampling = sample_from_stan
)
sbc$calibrate(N = 512, L = 20, keep_stan_fit = FALSE)
```

Plotting the histogram

```
sbc$plot()
```



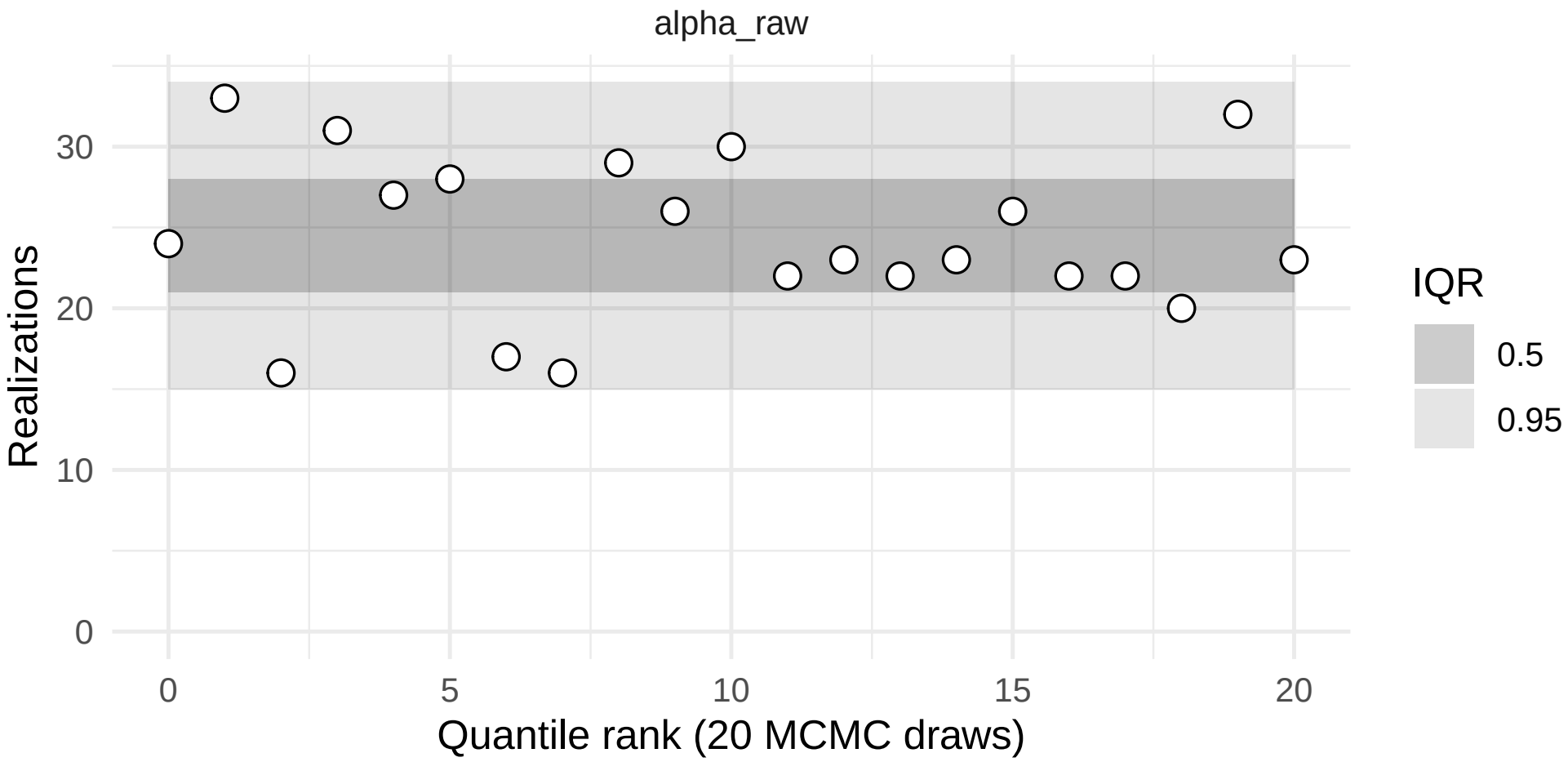
Comparing to binomial expectations

```
sbc$summary()
```

```
##
##
##     iq expected.outside actual.outside
##   0.50              0.50     0.38095238
##   0.95              0.05     0.04761905
```

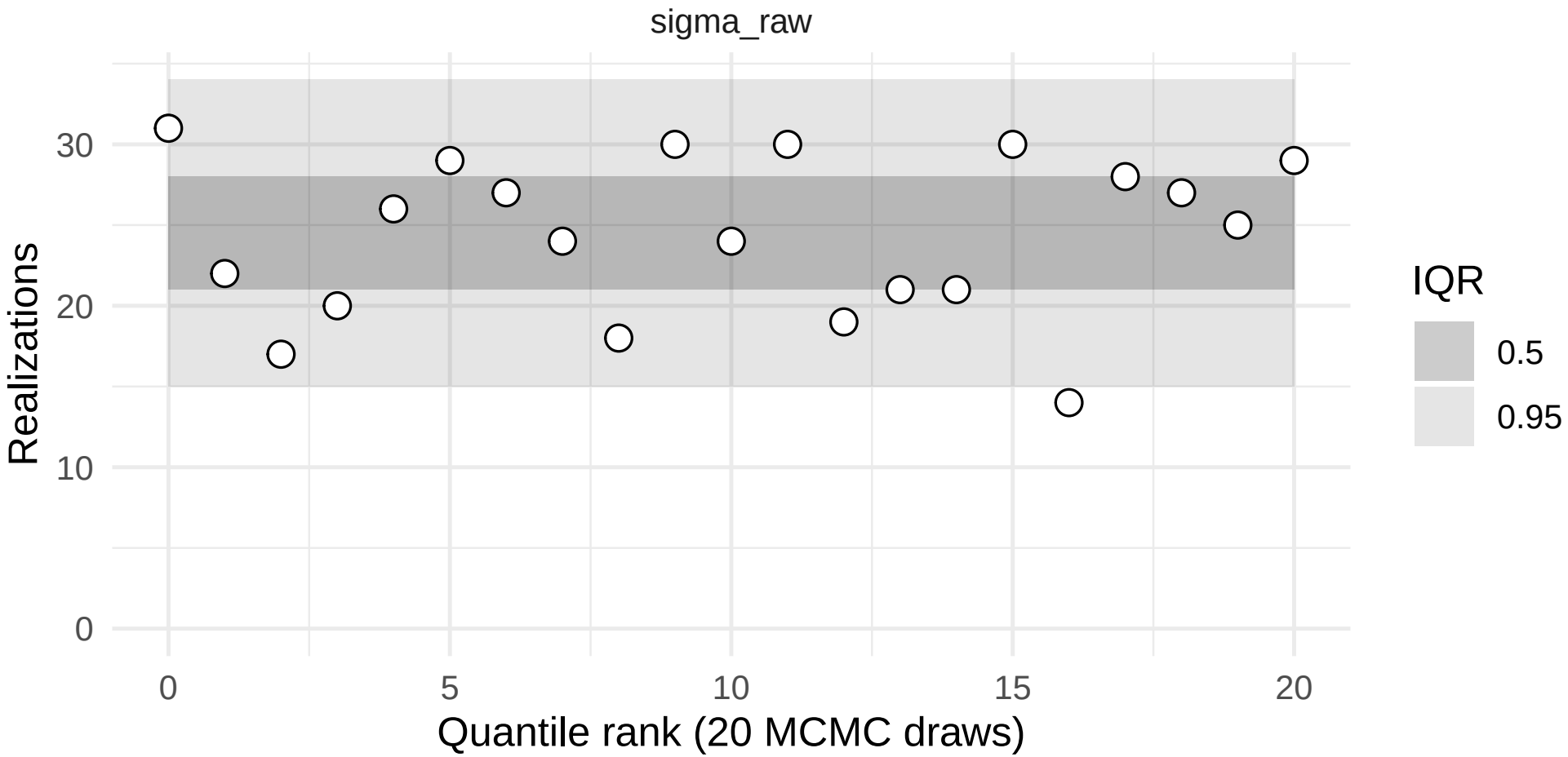
Looking at individual parameters: α

```
sbc$plot(var = "alpha_raw")
```



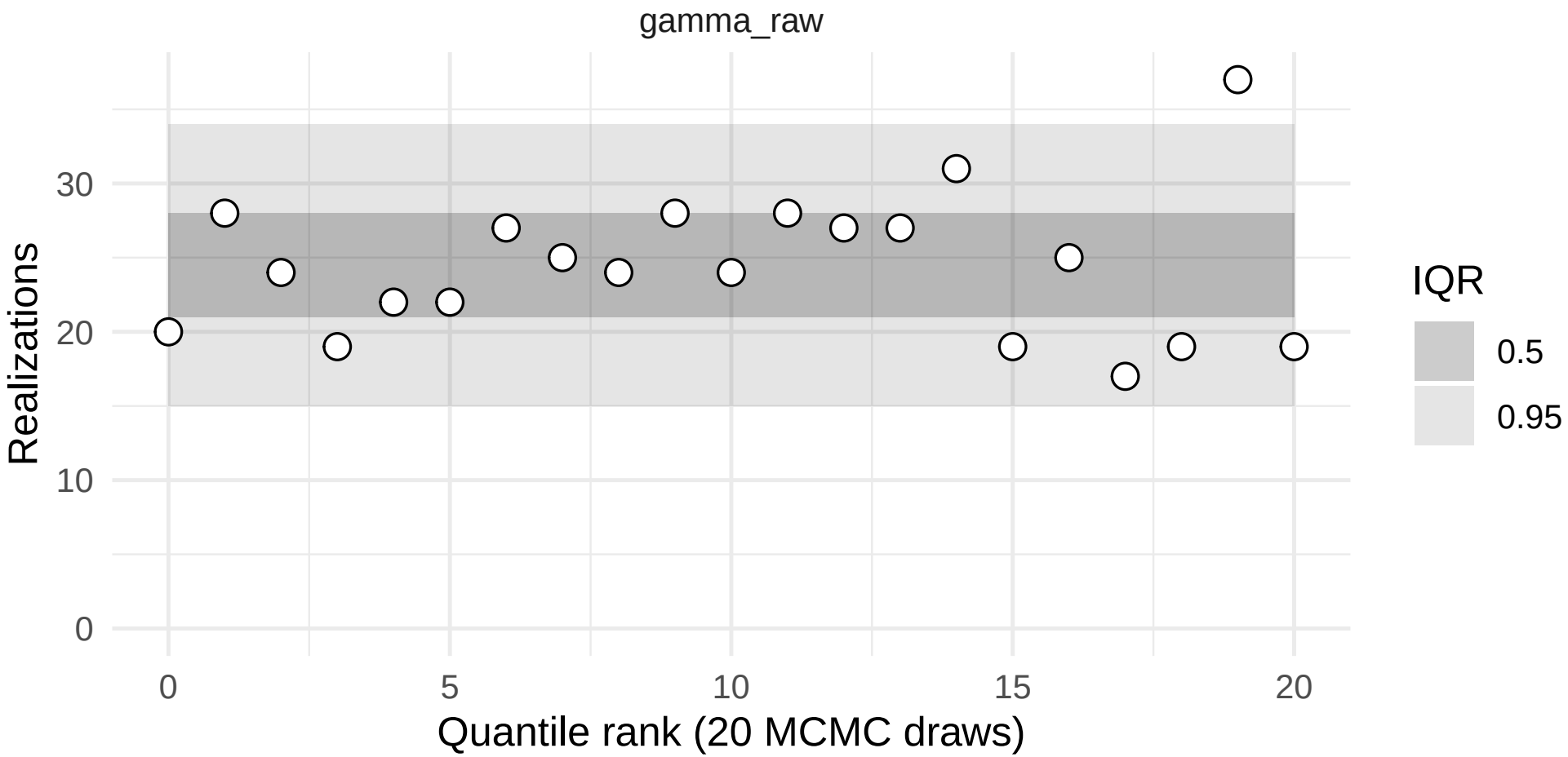
Looking at individual parameters: σ

```
sbc$plot(var = "sigma_raw")
```



Looking at individual parameters: γ

```
sbc$plot(var = "gamma_raw")
```



References

Cook, Samantha R, Andrew Gelman, and Donald B Rubin. 2006. "Validation of Software for Bayesian Models Using Posterior Quantiles." *Journal of Computational and Graphical Statistics* 15 (3): 675–92.

Geweke, John. 2004. "Getting It Right: Joint Distribution Tests of Posterior Simulators." *Journal of the American Statistical Association* 99 (467): 799–804.

Roos, Jason M.T. 2019. *sbcrs: R Package for Implementing Simulation Based Calibration Using Rank Statistics*. https://github.com/jasonmtroos/sbcrs.

Talts, Sean, Michael Betancourt, Daniel Simpson, Aki Vehtari, and Andrew Gelman. 2018. "Validating Bayesian Inference Algorithms with Simulation-Based Calibration." https://arxiv.org/abs/1804.06788.