

An introduction to Stan for applied Bayesian inference

FW 891

Christopher Cahill
6 September 2023



Quantitative Fisheries Center
MICHIGAN STATE UNIVERSITY

Purpose

- Learn the basic syntax of the Stan language
- Write code to elicit simple models and implement Bayesian inference in Stan
- Use the cmdstanr interface
- Develop familiarity with a few packages that make your life easier
- Walk through some model diagnostics
- Make sure you have these programs/packages installed



Installing CmdStanR

- See the [installation instructions here](#)

```
1 library(cmdstanr)
2 # use a built in file that comes with cmdstanr:
3 file <- file.path(
4   cmdstan_path(), "examples",
5   "bernoulli", "bernoulli.stan"
6 )
7 mod <- cmdstan_model(file)
```

see also [CmdStan user's guide](#)

Now let's make sure it works

```
1 # tagged list where names correspond to the .stan data block
2 stan_data <- list(N = 10, y = c(0, 1, 0, 0, 0, 0, 0, 0, 0, 1))
3
4 fit <- mod$sample(
5   data = stan_data,
6   seed = 123,
7   chains = 4,
8   parallel_chains = 4,
9   refresh = 500 # print update every 500 iters
10 )
```

Do the bottom numbers match up?

Running MCMC with 4 parallel chains...

```
Chain 1 Iteration:    1 / 2000 [  0%] (Warmup)
Chain 1 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 1 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 2 Iteration:    1 / 2000 [  0%] (Warmup)
Chain 2 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 2 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 3 Iteration:    1 / 2000 [  0%] (Warmup)
Chain 3 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 3 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 4 Iteration:    1 / 2000 [  0%] (Warmup)
Chain 4 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 4 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 1 finished in 0.0 seconds.
Chain 2 finished in 0.0 seconds.
Chain 3 finished in 0.0 seconds.
Chain 4 finished in 0.0 seconds.
```

All 4 chains finished successfully.
Mean chain execution time: 0.0 seconds.
Total execution time: 0.3 seconds.

```
1 fit$summary() # you should get these numbers:
```

```
# A tibble: 2 × 10
  variable    mean median    sd  mad    q5    q95  rhat ess_bulk ess_tail
  <chr>      <num>  <num> <num> <num> <num> <num> <num>    <num>    <num>
1 lp__      -7.26  -6.99  0.719 0.329 -8.73  -6.75  1.00   1658.   1861.
2 theta      0.246  0.231 0.118 0.118  0.0811 0.463  1.00   1378.   1236.
```

Presumably this broke someone



Onward!

Stan: the basics

Stan is a probabilistic modeling language <https://mc-stan.org/>

- Freely available
- Implements HMC, and an algorithm called NUTS
 - No U-Turn Sampler
 - We are using it for full Bayesian inference, but it can do other things too (we will not talk about these things)
- The Stan [documentation](#) and [community](#) is legendary in my opinion, albeit dense at times

Using Stan requires writing a `.stan` file

- Coding in Stan is something of a cross between R, WINBUGS/JAGS, and C++
- It is a Turing complete programming language
- Stan requires you to be explicit
 - Need to tell it whether something is a real, integer, vector, matrix, array, etc.
 - Lines need to end in a `;`
- A `.stan` file relies on program blocks to read in your data and construct your model
- Many built in functions you can use
- Why must we confront misery of a new language?

A linear regression in Stan

Let's build a linear regression model, which can be written a few ways:

$$y_i = \beta_0 + \beta x_i + \epsilon_i \quad \text{where} \quad \epsilon_i \sim \text{normal}(0, \sigma).$$

which is the same as

$$y_i - (\beta_0 + \beta X_i) \sim \text{normal}(0, \sigma)$$

and reducing further:

$$y_i \sim \text{normal}(\beta_0 + \beta X_i, \sigma).$$

Linear regression in Stan cont'd

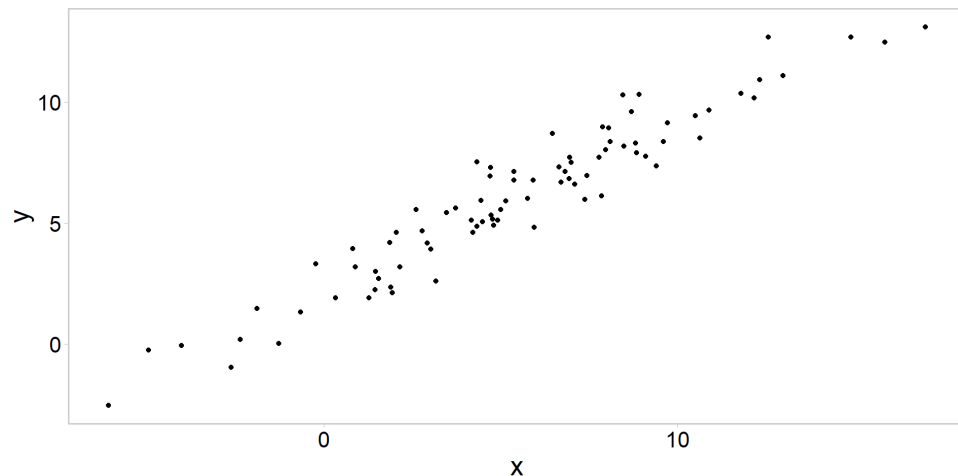
- Let's build a simple linear regression model in Stan

The data

- What do we do when we get some data?

Always plot the data

```
1 library(tidyverse)
2 library(ggqfc)
3 library(bayesplot)
4 library(cmdstanr)
5
6 data <- readRDS("data/linreg.rds")
7 p <- data %>% ggplot(aes(y = y, x = x)) +
8   geom_point() +
9   theme_qfc() +
10   theme(text = element_text(size = 20))
11 p
```



Always plot the data

```
1 p + geom_smooth(method = lm, se = F)
```

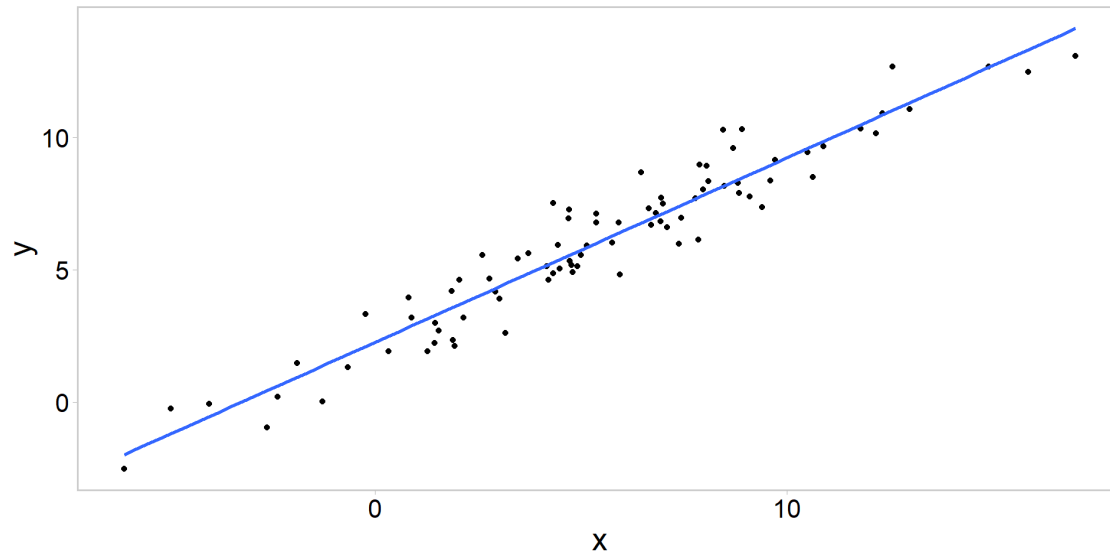
```
1 lm(data$y ~ data$x) # fit  $y = a + bx + e$ , where  $e \sim N(0, sd)$ 
```

Call:

```
lm(formula = data$y ~ data$x)
```

Coefficients:

| | |
|-------------|---------|
| (Intercept) | data\$x |
| 2.2559 | 0.6979 |



Thinking through our model

$$y_i \sim \text{normal}(\beta_0 + \beta X_i, \sigma).$$

signal = deterministic component + random component

Thinking through our model

$$y_i \sim \text{normal}(\beta_0 + \beta X_i, \sigma).$$

signal = deterministic component + random component

If $\mu_i \in \mathbb{R}$ and $\sigma \in \mathbb{R}^+$, then for $y_i \in \mathbb{R}$,

$$\text{Normal}(y_i \mid \mu_i, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2} \left(\frac{y_i - \mu_i}{\sigma}\right)^2\right)$$

Thinking through our model

$$y_i \sim \text{normal}(\beta_0 + \beta X_i, \sigma).$$

signal = deterministic component + random component

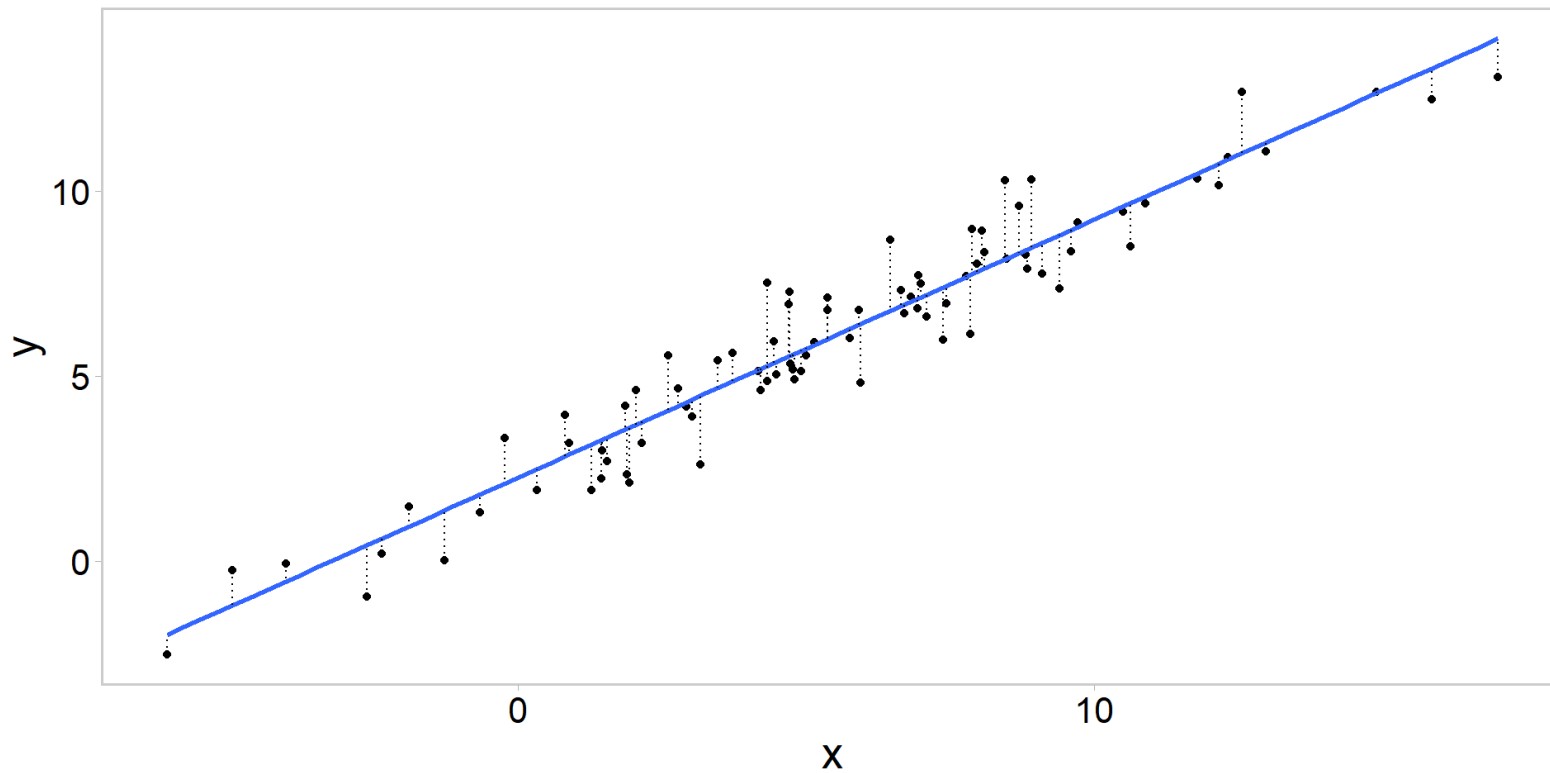
If $\mu_i \in \mathbb{R}$ and $\sigma \in \mathbb{R}^+$, then for $y_i \in \mathbb{R}$,

$$\text{Normal}(y_i \mid \mu_i, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2} \left(\frac{y_i - \mu_i}{\sigma}\right)^2\right)$$

where $\mu_i = \beta_0 + \beta X_i$

Thinking through our model

$$y_i \sim \text{normal}(\beta_0 + \beta X_i, \sigma).$$



Writing our first `.stan` model

Code to do what we are going through is in the [week2/](#) Github directory

`linreg.R` and `linreg.stan`



mc-stan.org

Structure of a `.stan` file

```
1 // this is a comment
2 // program block demonstration
3 data{
4   // read in data here -- this section is executed one time per Stan run
5 }
6 transformed data {
7   // transform the data here -- this section is also executed one time per Stan run
8 }
9 parameters {
10   // declare the **estimated** parameters here
11 }
12 transformed parameters{
13   // this section takes parameter estimates and data (or transformed data)
14   // and transforms them for use later on in model section
15 }
16 model{
17   // this section specifies the prior(s) and likelihood terms,
18   // and defines a log probability function (i.e., log posterior) of the model
19 }
20 generated quantities{
21   // this section creates derived quantities based on parameters,
22   // models, data, and (optionally) pseudo-random numbers.
23 }
```

- Can also write custom functions (although we won't in this class)

In words, rather than code

As per the comments in the code, each of the program blocks does certain stuff

- `data{ }` reads data into the .stan program
- `transformed data{ }` runs calculations on those data (once)
- `parameters{ }` declares the *estimated* parameters in a Stan program
- `transformed parameters{ }` takes the parameters, data, and transformed data, and calculates stuff you need for your model
- `model{ }` constructs a log probability function:
 - $\log(\text{posterior}) = \log(\text{priors}) + \log(\text{likelihood})$
- `generated quantities{ }` is only executed after you have your sampled posterior
 - useful for calculating derived quantities given your model, data, and parameters

Writing the `linreg.stan` file

```
1 data {
2   int<lower=0> n; // number of observations
3   vector[n] y;   // vector of responses
4   vector[n] x;   // covariate x
5 }
6 parameters {
7   real b0;
8   real b1;
9   real<lower = 0> sd;
10 }
11 model {
12   // priors
13   b0 ~ normal(0, 10);
14   b1 ~ normal(0, 10);
15   sd ~ normal(0, 1);
16
17   // likelihood - one way:
18   y ~ normal(b0 + b1*x, sd); // (vectorized, dropping constant, additive terms)
19 }
```

Writing the `linreg.stan` file

```
1 data {
2   int<lower=0> n; // number of observations
3   vector[n] y;   // vector of responses
4   vector[n] x;   // covariate x
5 }
6 parameters {
7   real b0;
8   real b1;
9   real<lower = 0> sd;
10 }
11 model {
12   // priors
13   b0 ~ normal(0, 10);
14   b1 ~ normal(0, 10);
15   sd ~ normal(0, 1);
16
17   // likelihood - loopy way:
18   for(i in 1:nobs){
19     y[i] ~ normal(b0 + b1*x[i], sd);
20   }
21 }
```

Writing the `linreg.stan` file

```
1 data {
2   int<lower=0> n; // number of observations
3   vector[n] y;   // vector of responses
4   vector[n] x;   // covariate x
5 }
6 parameters {
7   real b0;
8   real b1;
9   real<lower = 0> sd;
10 }
11 model {
12   // priors
13   b0 ~ normal(0, 10);
14   b1 ~ normal(0, 10);
15   sd ~ normal(0, 10);
16
17   // likelihood - yet another way:
18   target += normal_lpdf(y | b0 + b1*x, sd); // log(normal dens) (constants included)
19 }
```

Key points

- These three likelihood configurations result in the same parameter estimates, but option (3) will give you a different log posterior (`lp__`)
 - Vectorized option is the fastest, but sometimes these other configurations are helpful in specific applications
- Stan sets up the $\log(\text{posterior})$ as the $\log(\text{likelihood}) + \log(\text{priors})$ if you specify likelihood and priors

Some notes on priors in Stan

- If you don't specify priors, *Stan will specify flat priors for you*
 - Not always a good thing, and it can lead to problems
- In this class we are either going to use vague or uninformative priors, OR we will use informative priors that incorporate domain expertise or information from previous studies
- When we say a prior is “weakly informative,” what we mean is that if there's a large amount of data, the likelihood will dominate, and the prior will not be important
 - Prior can often only be understood in the context of the likelihood (Gelman et al. 2017; see also [prior recommendations in Stan](#))

Some tips for debugging `.stan` code

- Use one chain (else prepare for impending doomies)
- Use a low number of iterations (i.e., like 1-30)
- wrap things in `print()` statements in Stan
- Simulate fake data representing your model (you'll know what truth is)
- Build fast, fail fast
- Plot everything

Controlling everything from `linreg.R`

```
1 # compile the .stan model
2 mod <- cmdstan_model("src/linreg.stan")
3
4 # create a tagged data list
5 # names must correspond to data block{} in .stan
6 stan_data <- list(n = nrow(data), y = data$y, x = data$x)
7
8 # write a function to set starting values
9 inits <- function() {
10   list(
11     b0 = jitter(0, amount = 0.05),
12     b1 = jitter(0, amount = 1),
13     sd = jitter(1, amount = 0.5)
14   )
15 }
```

Controlling everything from `linreg.R`

```
1 # what happens when we call the inits() function
2 inits()
```

```
$b0
[1] -0.04361975
```

```
$b1
[1] 0.5690925
```

```
$sd
[1] 0.9183216
```

```
1 inits()
```

```
$b0
[1] 0.04810181
```

```
$b1
[1] -0.4342321
```

```
$sd
[1] 1.347882
```

- Can pass this `inits` function to `stan` to initialize each of our MCMC chains at different parameter values

Running the model

```
1 fit <- mod$sample(  
2   data = stan_data, # tagged stan_data list  
3   init = inits, # `inits()` is the function here  
4   seed = 13, # ensure simulations are reproducible  
5   chains = 4, # multiple chains  
6   iter_warmup = 1000, # how long to warm up the chains  
7   iter_sampling = 1000, # how many samples after warmup  
8   parallel_chains = 4, # run them in parallel?  
9   refresh = 500 # print update every 500 iters  
10 )
```

- see also `?sampling` for *many* other options
- 1000 iterations for warmup and sampling is not a bad place to start (however see debugging tips at the end of the lecture)

Running the model

Running MCMC with 4 parallel chains...

```
Chain 1 Iteration:    1 / 2000 [  0%] (Warmup)
Chain 1 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 1 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 1 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 2 Iteration:    1 / 2000 [  0%] (Warmup)
Chain 2 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 2 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 3 Iteration:    1 / 2000 [  0%] (Warmup)
Chain 3 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 3 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 3 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 4 Iteration:    1 / 2000 [  0%] (Warmup)
Chain 4 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 4 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 1 finished in 0.1 seconds.
Chain 2 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 2 finished in 0.1 seconds.
Chain 3 finished in 0.1 seconds.
Chain 4 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 4 finished in 0.1 seconds.
```

All 4 chains finished successfully.
Mean chain execution time: 0.1 seconds.
Total execution time: 0.4 seconds.

Bayesian model diagnostics

- Diagnostics for Bayesian models can be lumped into two categories:
 1. Diagnostics that evaluate the performance of your MCMC algorithm
 2. Diagnostics that help you understand your model fit vs. observed data

Both types of checks are required to ensure the reliability of your inferences in a Bayesian setting

- We will start with MCMC diagnostics

Did Stan run into any obvious issues?

```
1 fit$diagnostic_summary() # sampler diagnostic summaries
```

```
$num_divergent
```

```
[1] 0 0 0 0
```

```
$num_max_treedepth
```

```
[1] 0 0 0 0
```

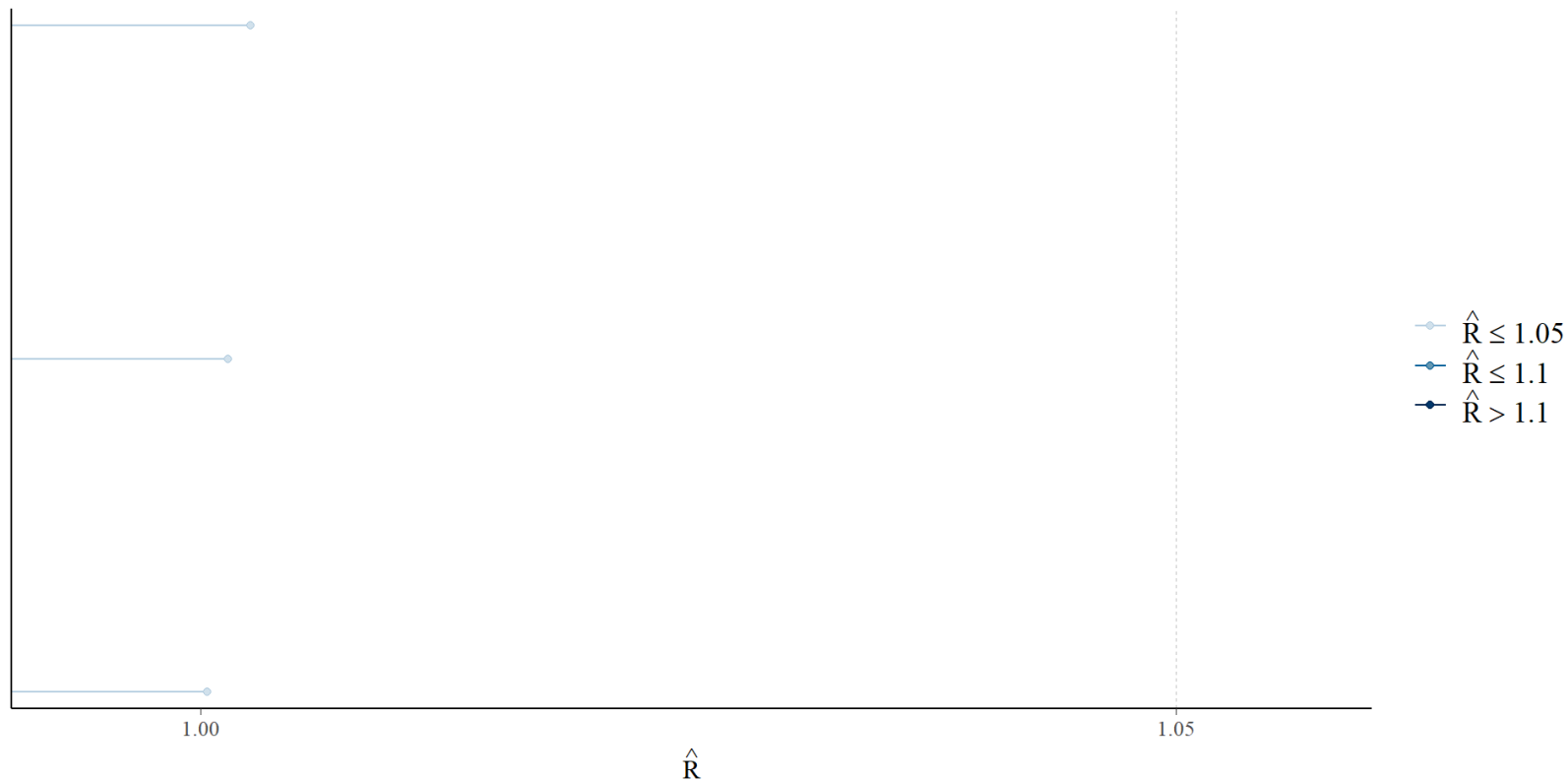
```
$ebfmi
```

```
[1] 1.115859 1.040690 1.054994 1.177122
```

- [see here](#) for a description of runtime warnings and issues related to convergence problems
- Hamiltonian based Estimated Bayesian Fraction of Missing Information (e-bfmi) quantifies how hard it is to sample level sets at each iteration
 - if very low (i.e., < 0.3), sampler is having a difficult time sampling the target distribution ([Betancourt 2017](#))

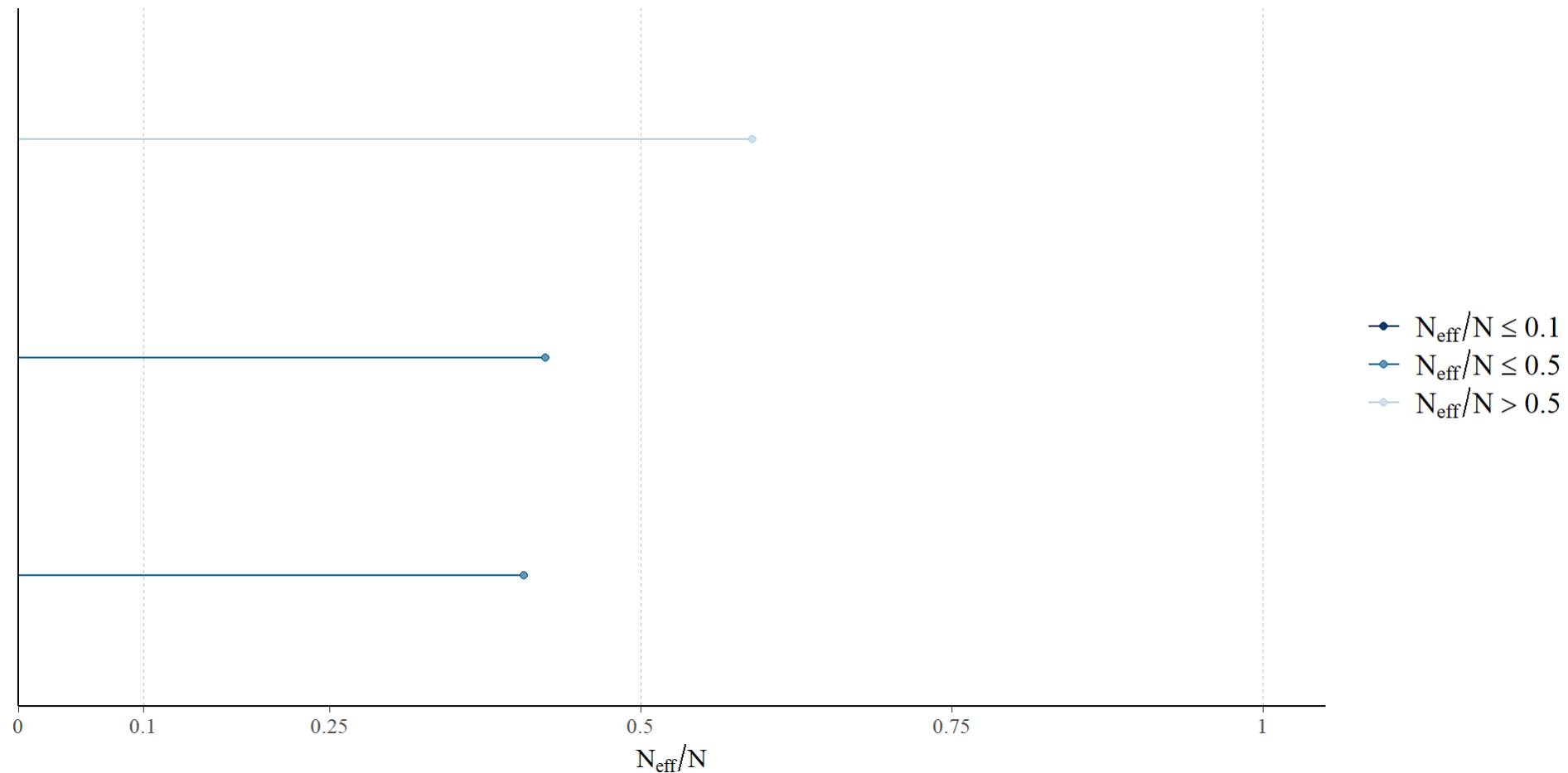
Examine \hat{R}

```
1 # Have the chains converged to a common distribution?  
2 # compares the between- and within-chain estimates for parameters  
3 rhats <- rhat(fit)  
4 mcmc_rhat(rhats) # should all be less than 1.05 as rule of thumb
```



Examine the number of effective samples

```
1 eff <- neff_ratio(fit)
2 mcmc_neff(eff) # rule of thumb is worry about ratios < 0.1
```



Extracting the posterior draws from our CmdStanFit object

```
1 # extract the posterior draws
2 posterior <- fit$draws(format = "df") # extract draws x variables df
3 head(posterior)
```

```
# A draws_df: 6 iterations, 1 chains, and 4 variables
```

| | lp__ | b0 | b1 | sd |
|---|------|-----|------|------|
| 1 | -37 | 2.3 | 0.69 | 0.90 |
| 2 | -37 | 2.1 | 0.71 | 0.97 |
| 3 | -38 | 2.0 | 0.73 | 0.95 |
| 4 | -38 | 2.0 | 0.74 | 0.98 |
| 5 | -37 | 2.4 | 0.68 | 0.92 |
| 6 | -37 | 2.4 | 0.69 | 0.99 |

```
# ... hidden reserved variables {'.chain', '.iteration', '.draw'}
```

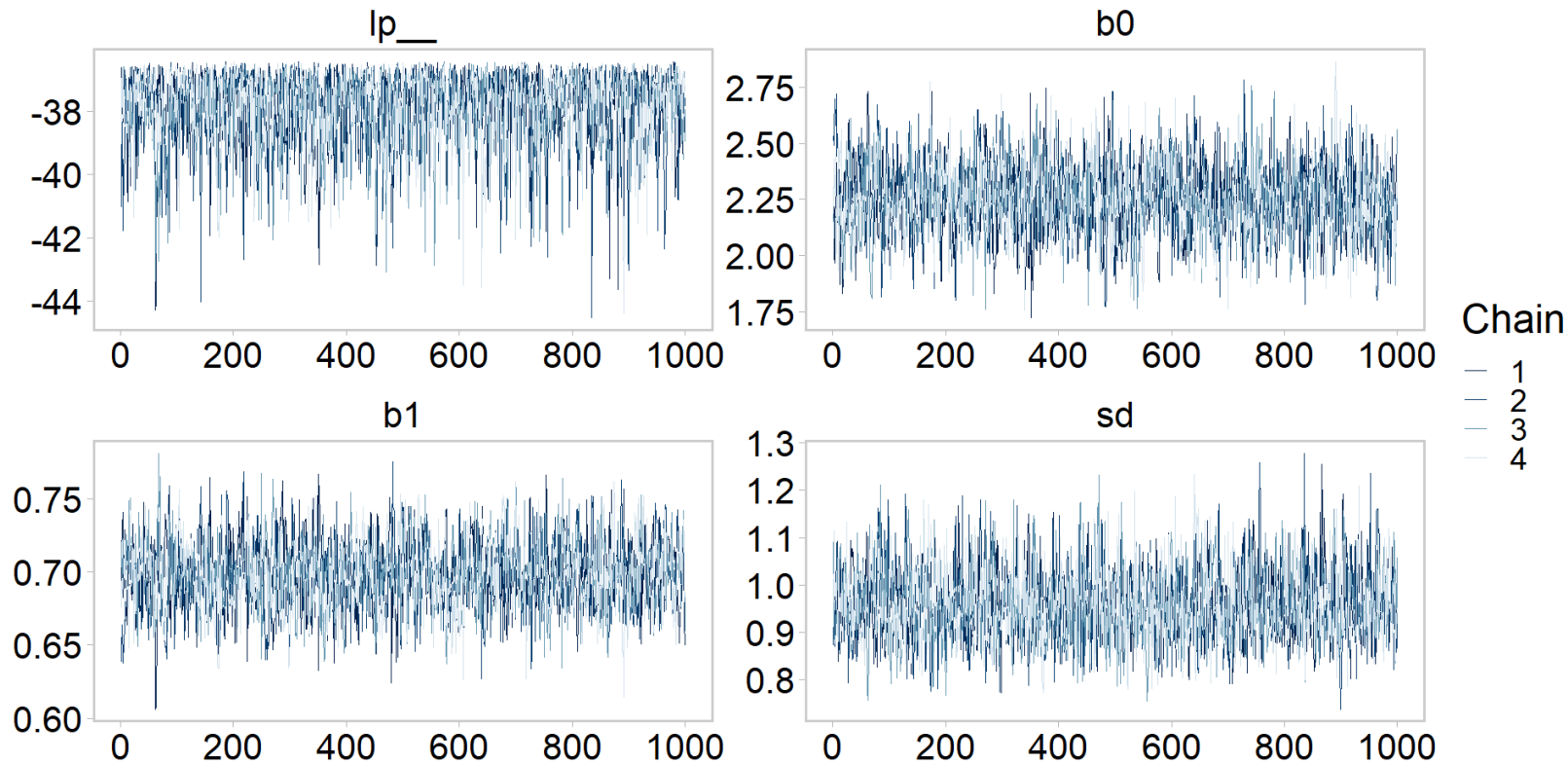
```
1 dim(posterior)
```

```
[1] 4000    7
```

```
1 np <- nuts_params(fit) # get the sampler parameters - useful for debugging
```

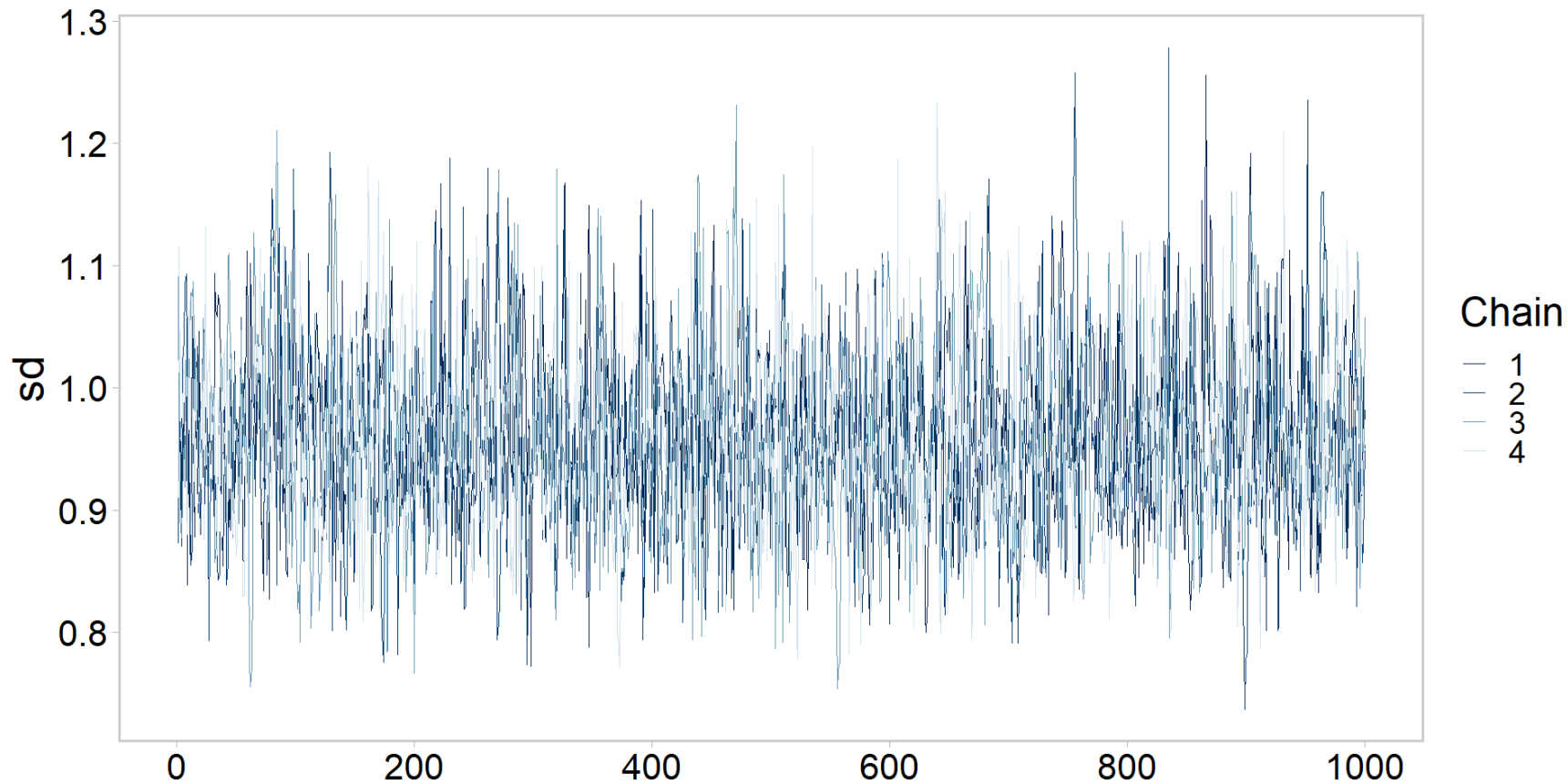
Let's get tidy: visualizing the chains

```
1 color_scheme_set("blue") # bayesplot color themes
2 # plot the chains of all parameters
3 p <- mcmc_trace(posterior, np = np) +
4   theme_qfc() + theme(text = element_text(size = 20))
5 p
```



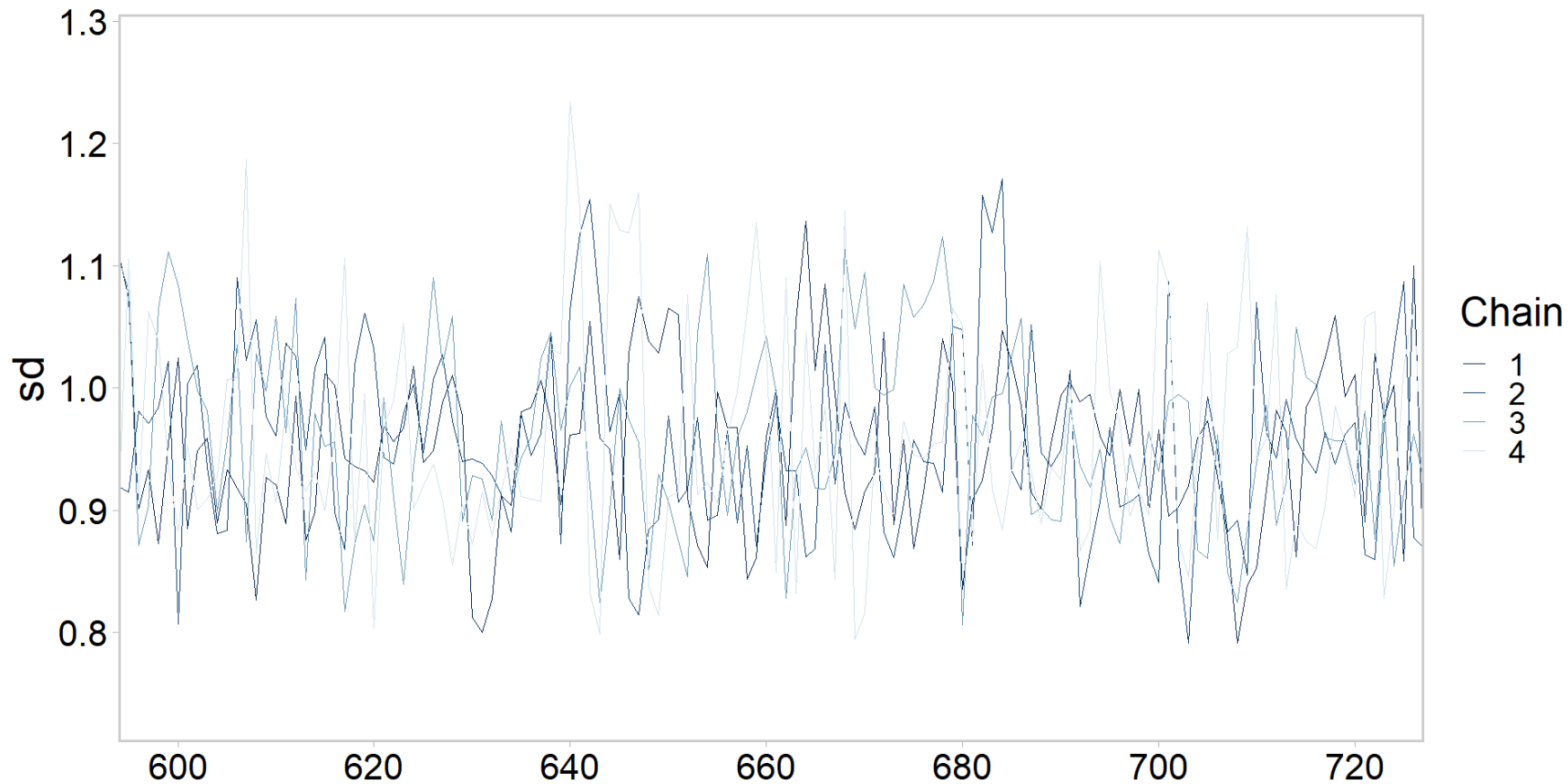
Let's get tidy: visualizing a chain

```
1 # plot chain of one parameter
2 p <- mcmc_trace(posterior, pars = "sd", np = np) +
3   theme_qfc() + theme(text = element_text(size = 20))
4 p
```



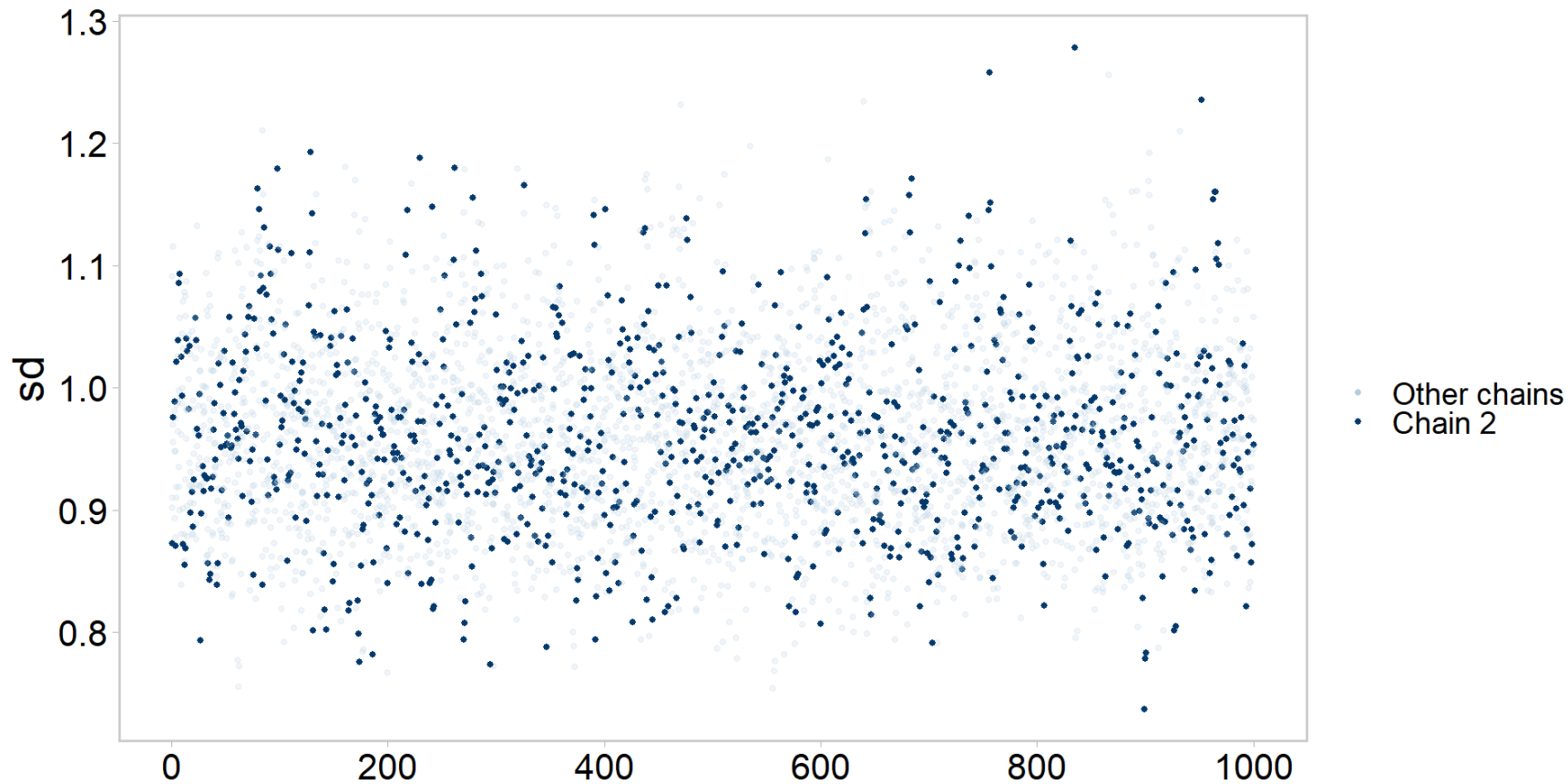
Let's get tidy: zooming in on chains

```
1 # zoom in on one parameter iterations 600-721 (bc why not?)
2 p <- mcmc_trace(posterior, pars = "sd", window = c(600, 721), np = np) +
3   theme_qfc() + theme(text = element_text(size = 20))
4 p
```



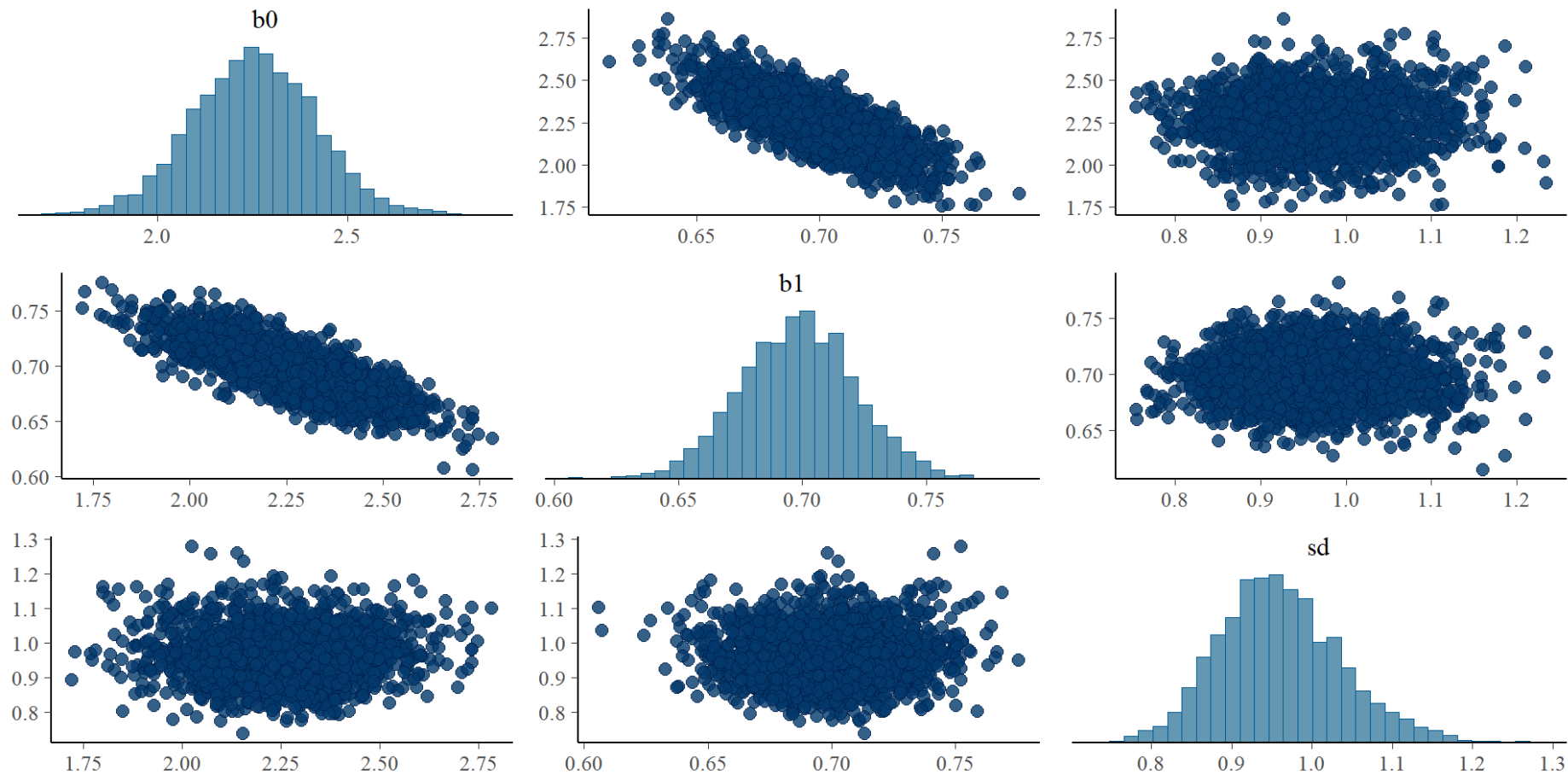
Let's get tidy: highlighting one chain

```
1 # highlight chain 2 vs. other chains:
2 p <- mcmc_trace_highlight(posterior, pars = "sd", highlight = 2) +
3   theme_qfc() + theme(text = element_text(size = 20))
4 p
```



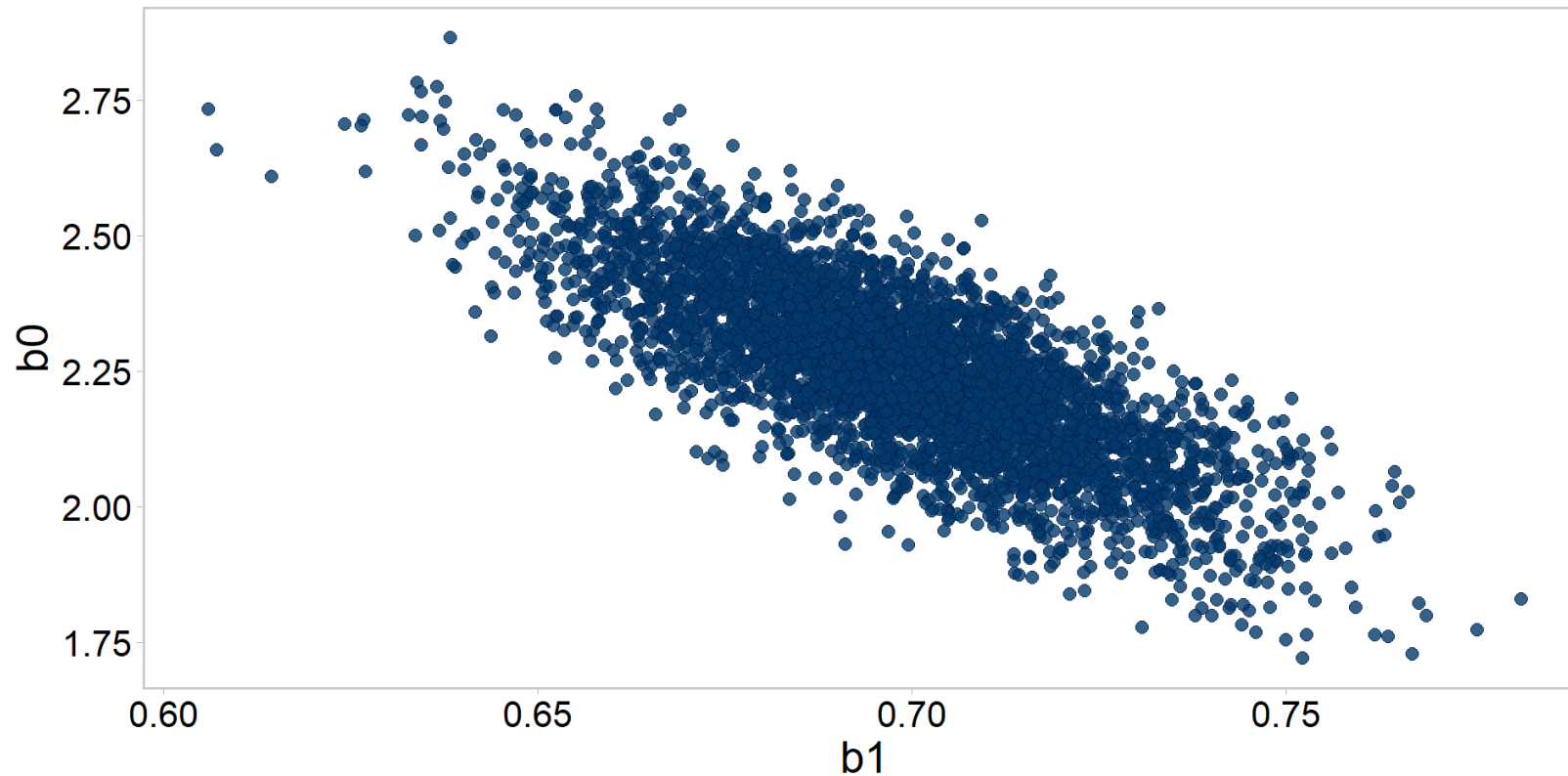
Let's get tidy: pairs plots

```
1 # pairs plots - good modelers almost always use this
2 p <- mcmc_pairs(posterior, pars = c("b0", "b1", "sd"), np = np)
3 p
```



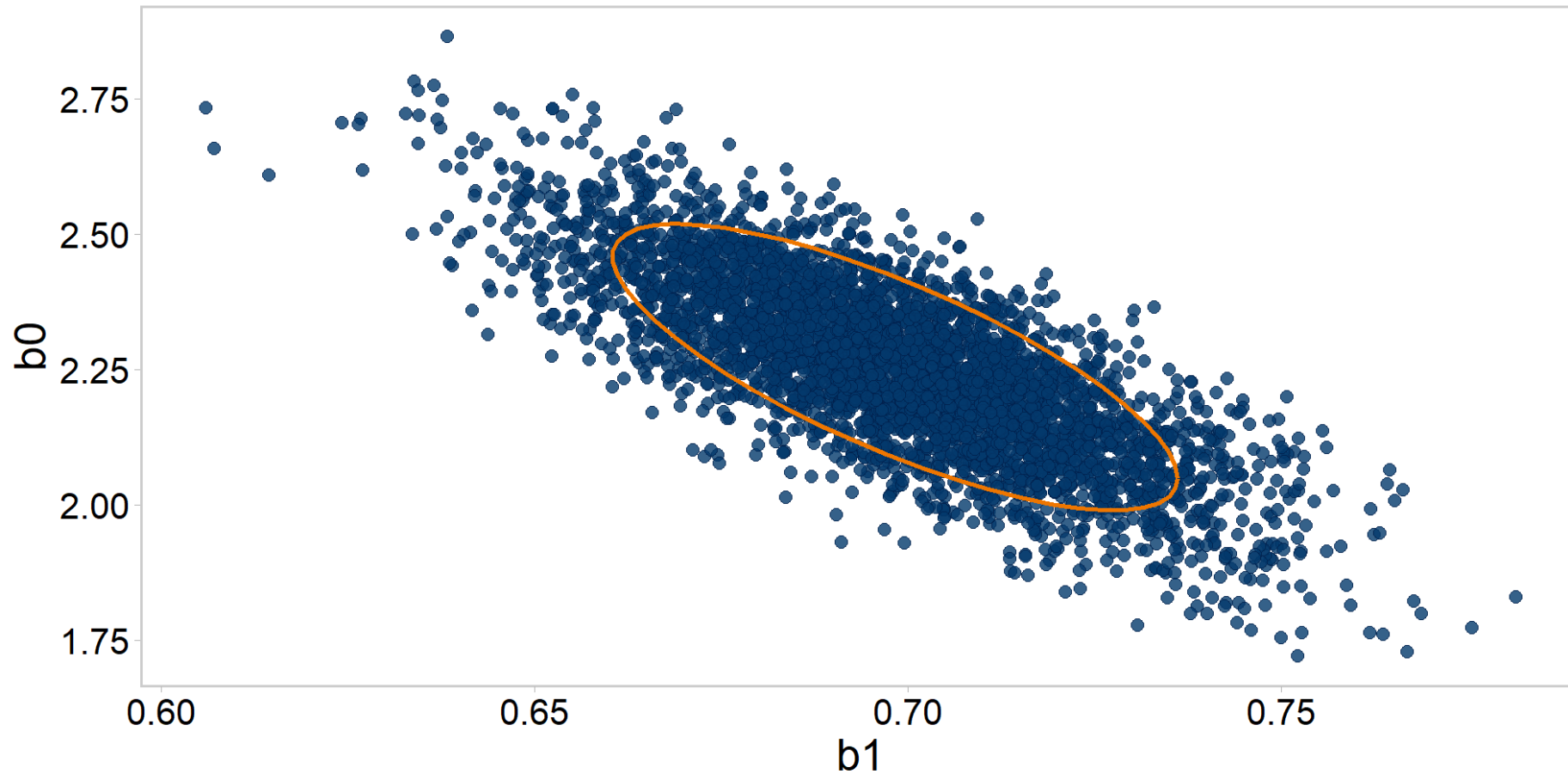
Let's get tidy: pairs plots

```
1 # pairs plots - good modelers almost always use this
2 # two parameters only:
3 p <- mcmc_scatter(posterior, pars = c("b1", "b0"), np = np) +
4   theme_qfc() + theme(text = element_text(size = 20))
5 p # check out that negative correlation!
```



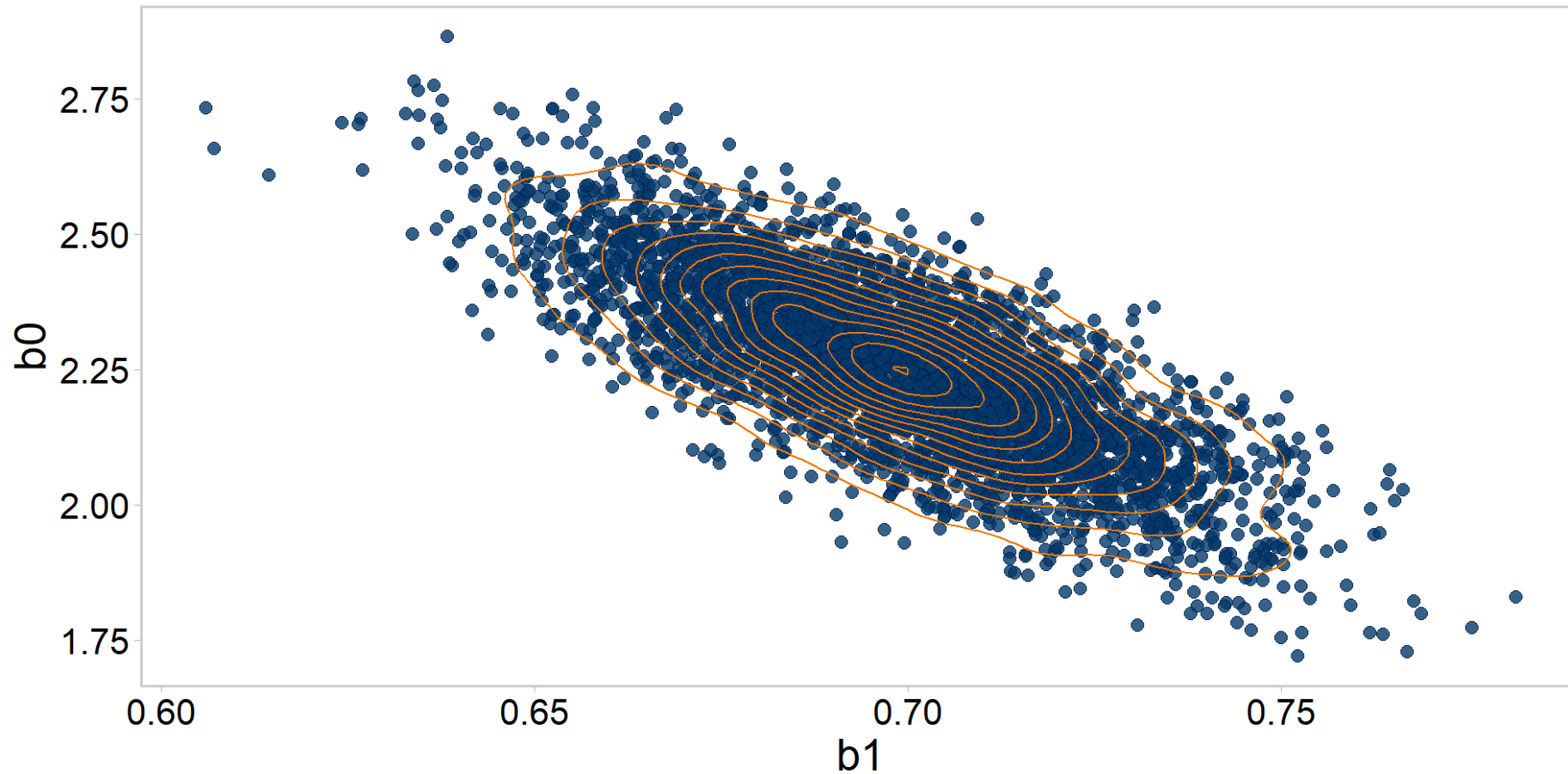
Let's get tidy: pairs plots + quantiles

```
1 # add an 83% (why not, the world is your oyster) ellipse to it
2 p + stat_ellipse(level = 0.83, color = "darkorange2", size = 1) +
3   theme_qfc() + theme(text = element_text(size = 20))
```



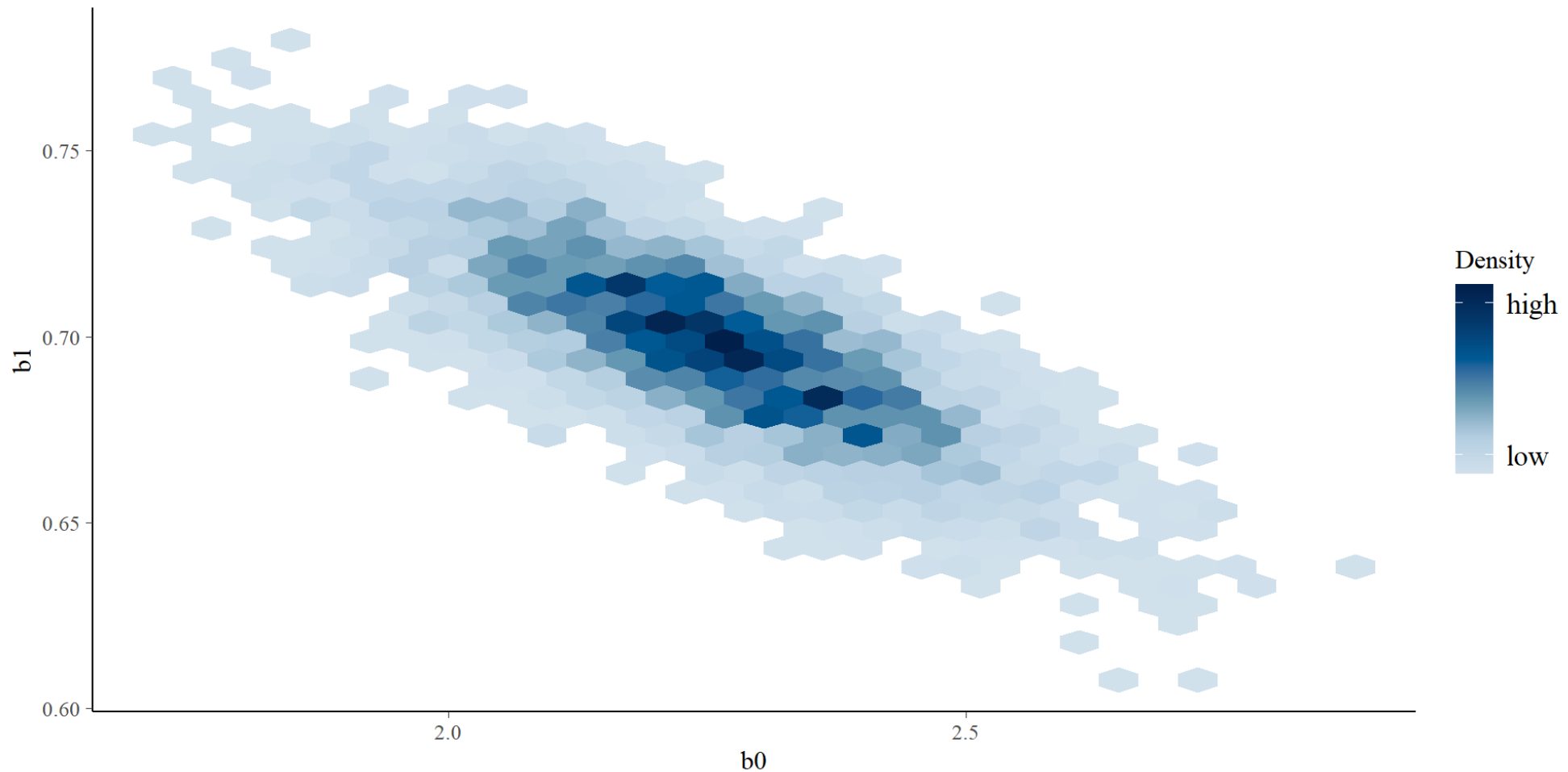
Let's get tidy: pairs plots + contours

```
1 # visualize the posterior distribution's contours for b0, b1
2 p + stat_density_2d(color = "darkorange2", size = .5) +
3   theme_qfc() + theme(text = element_text(size = 20))
```



Let's get tidy: pairs plots + contours

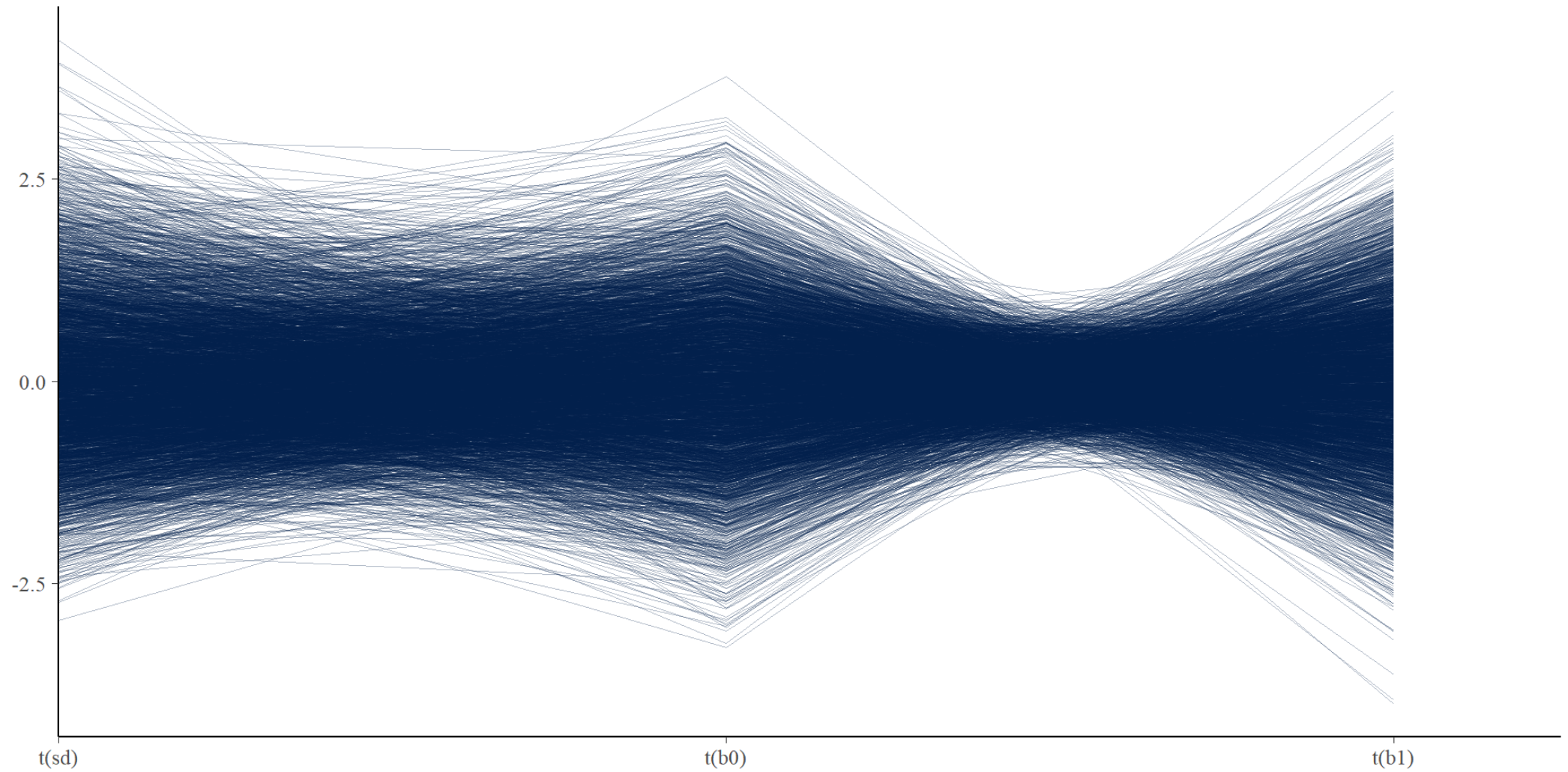
```
1 # view it a different way
2 mcmc_hex(posterior, pars = c("b0", "b1"))
```



Let's get tidy: divergent transitions

```
1 # visualizing divergent transitions
2 # none here, but this plot shows each iteration as a line connecting
3 # parameter values, and divergent iterations will show up as red lines
4 # sometimes this helps you find combinations of parameters that are
5 # leading to divergent transitions
6
7 mcmc_parcoord(posterior,
8   pars = c("sd", "b0", "b1"),
9   transform = function(x) {
10     (x - mean(x)) / sd(x) # mean standardize (easier to compare)
11   },
12   np = np
13 )
```

Let's get tidy: divergent transitions



Moving on to fits vs. data checks

Posterior predictive checks (PPCs)

- Generate replicate datasets based on our posterior draws
- A great way to find discrepancies between your fitted model and the data, critical test for Bayesian models
- Always do them

Posterior predictive checks in R

```
1 head(posterior)
```

```
# A draws_df: 6 iterations, 1 chains, and 4 variables
```

```
  lp__    b0    b1    sd
1  -37  2.3  0.69  0.90
2  -37  2.1  0.71  0.97
3  -38  2.0  0.73  0.95
4  -38  2.0  0.74  0.98
5  -37  2.4  0.68  0.92
6  -37  2.4  0.69  0.99
```

```
# ... hidden reserved variables {'.chain', '.iteration', '.draw'}
```

```
1 b0 <- posterior$b0
```

```
2 b1 <- posterior$b1
```

```
3 sd <- posterior$sd
```

```
4
```

```
5 # generate 1 dataset from the first draw of the posterior:
```

```
6 set.seed(1)
```

```
7 y_rep <- rnorm(length(data$x), b0[1] + b1[1] * data$x, sd[1])
```

Posterior predictive checks in R

```
1 # now do it for the whole posterior
2 # loop through and create replicate datasets based on
3 # each_draw_of_posterior
4 set.seed(1)
5 y_rep <- matrix(NA, nrow = nrow(posterior), ncol = length(data$y))
6 for (i in 1:nrow(posterior)) {
7   y_rep[i, ] <- rnorm(length(data$x), b0[i] + b1[i] * data$x, sd[i])
8 }
9 dim(y_rep)
```

```
[1] 4000    84
```


Posterior predictive checks in Stan

- Can do this in Stan directly via the `generated quantities{ }` section:

```
1 generated quantities {  
2   // replications for the posterior predictive distributions  
3   array[n] real y_rep = normal_rng(b0 + b1*x, sd);  
4 }
```

Posterior predictive checks in Stan

- Recompile and re-run:

```
1 mod <- cmdstan_model("src/linreg_ppc.stan")
2 fit <- mod$sample(
3   data = stan_data,
4   init = inits,
5   seed = 13, # ensure simulations are reproducible
6   chains = 4, # multiple chains
7   iter_warmup = 1000, # how long to warm up the chains
8   iter_sampling = 1000, # how many samples after warmup
9   parallel_chains = 4, # run them in parallel?
10  refresh = 0
11 )
```

Running MCMC with 4 parallel chains...

Chain 1 finished in 0.1 seconds.

Chain 2 finished in 0.1 seconds.

Chain 3 finished in 0.1 seconds.

Chain 4 finished in 0.2 seconds.

All 4 chains finished successfully.

Mean chain execution time: 0.1 seconds.

Total execution time: 0.4 seconds.

Posterior predictive checks in Stan

- Extract the posterior and take a gander at our new `y_reps`

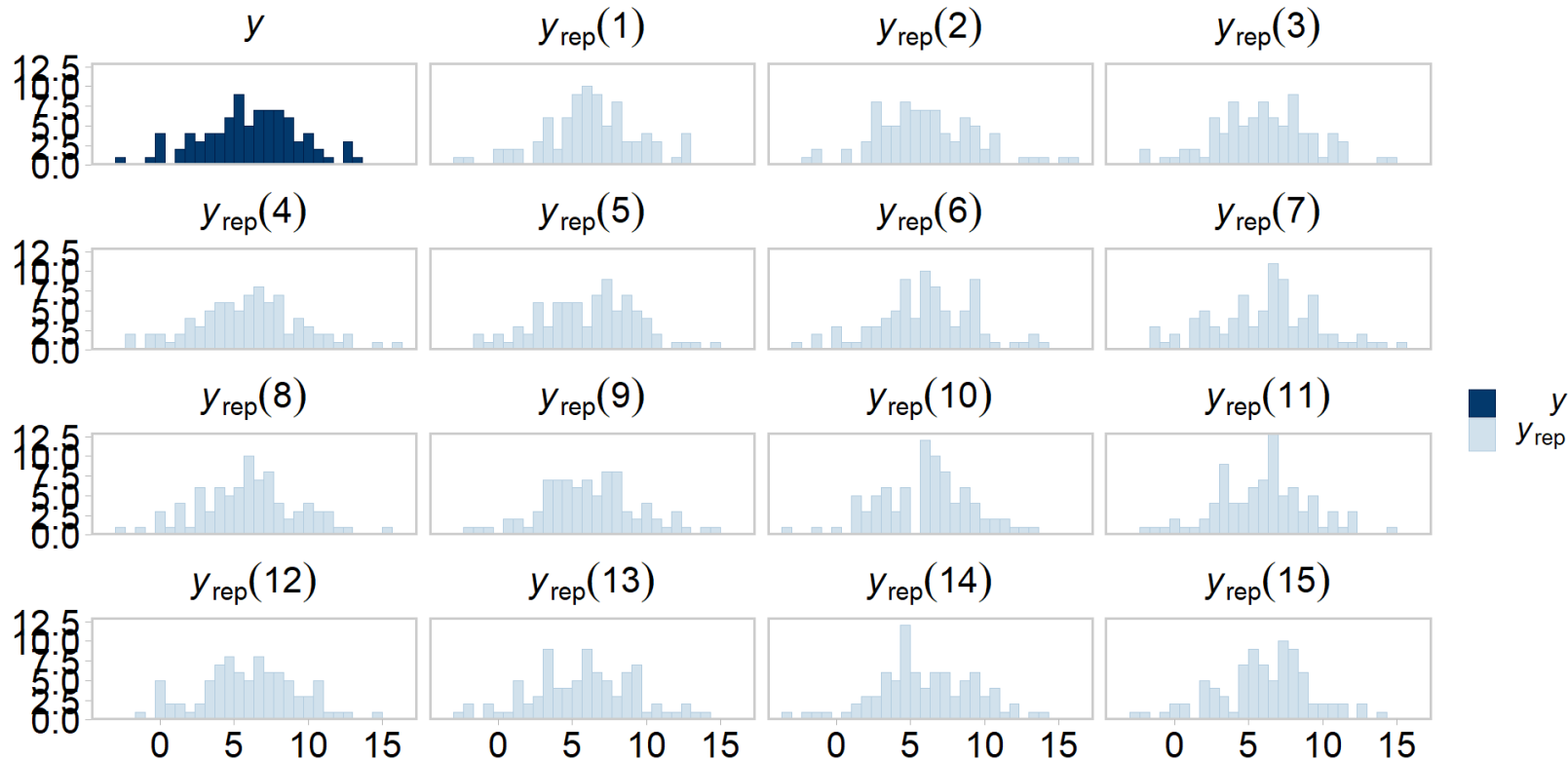
```
1 posterior <- fit$draws(format = "df") # extract draws x variables data frame
2 head(posterior)
```

```
# A draws_df: 6 iterations, 1 chains, and 88 variables
  lp__    b0    b1    sd y_rep[1] y_rep[2] y_rep[3] y_rep[4]
1  -37  2.3  0.71  0.93      3.1      6.5      2.49      11
2  -38  2.3  0.71  1.06      3.7      5.5      2.45      10
3  -40  2.1  0.74  0.87      3.6      4.6      2.56      11
4  -40  1.9  0.73  1.06      4.2      6.1      0.12      12
5  -37  2.4  0.67  0.92      5.7      7.2      2.63      10
6  -37  2.4  0.67  0.98      4.0      6.3      1.63      13
# ... with 80 more variables
# ... hidden reserved variables {'.chain', '.iteration', '.draw'}
```

- Stan generated or simulated replicate `y_rep` “datasets”
 - $n_iter * n_chain$ = number of posterior draws
- Now compare the simulated data to our original (real) dataset

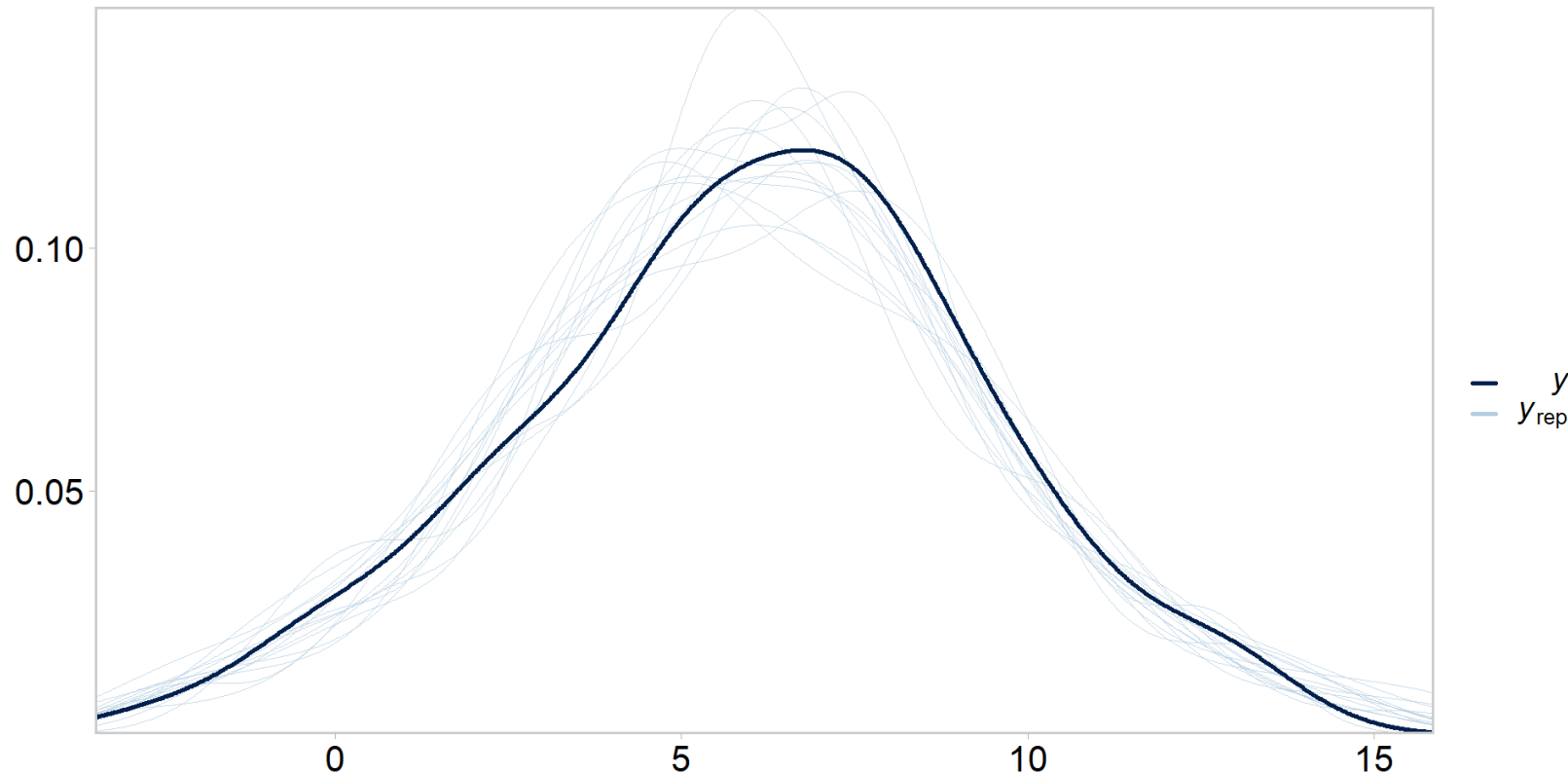
Even more tidy: visualizing PPCs

```
1 y_rep <- posterior[grepl("y_rep", names(posterior))]  
2  
3 # compare original data against 15 simulated datasets:  
4 p <- ppc_hist(y = data$y, yrep = as.matrix(y_rep[1:15, ]))  
5 p + theme_qfc() + theme(text = element_text(size = 20))
```



Even more tidy: visualizing PPCs

```
1 y_rep <- posterior[grepl("y_rep", names(posterior))]  
2  
3 # compare original data against 15 simulated datasets:  
4 p <- ppc_dens_overlay(y = data$y, yrep = as.matrix(y_rep[1:15, ]))  
5 p + theme_qfc() + theme(text = element_text(size = 20))
```



Visualizing PPCs another way

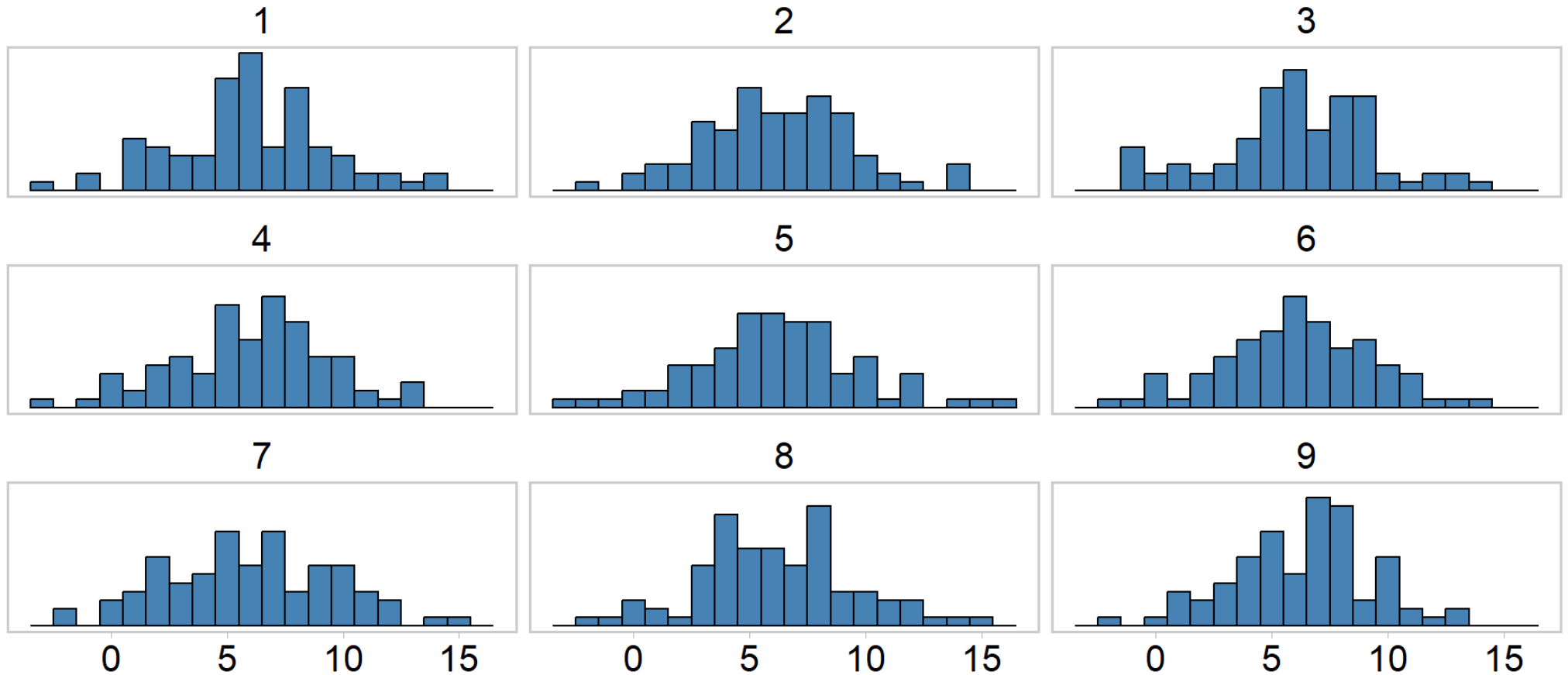
```
1 # ppcs, another way
2 y_reps <- y_rep[sample(nrow(y_rep), 9), ] # draw 9 replicate datasets
3 ind <- sample(9, 1)
4 y_reps[ind, ] <- as.list(data$y) # replace a random y_rep with true y
5
6 yrep_df <- y_reps %>%
7   as.data.frame() %>%
8   pivot_longer(everything()) %>% # use the long format for plotting
9   mutate(name = rep(1:9, each = ncol(y_reps)))
```

Visualizing PPCs another way

```
1 # ppcs, another way
2 yrep_df %>%
3   ggplot() +
4   geom_histogram(aes(x = value),
5     fill = "steelblue",
6     color = "black", binwidth = 1
7   ) +
8   facet_wrap(~name, nrow = 3) +
9   labs(x = "", y = "") +
10  scale_y_continuous(breaks = NULL) +
11  ggtitle("Can you spot the real data?") +
12  theme_qfc() +
13  theme(text = element_text(size = 20))
```

Visualizing PPCs another way

Can you spot the real data?



Prior predictive checks

- Same idea as posterior predictive checks, but we ignore the likelihood
- Simply sample replicate datasets from the prior(s)
- Tries to get at whether our priors + model configuration are consistent with our knowledge of the system
- Usually would do these before posterior predictive checks, but easier to explain once you understand what posterior predictive checks are

Prior predictive checks

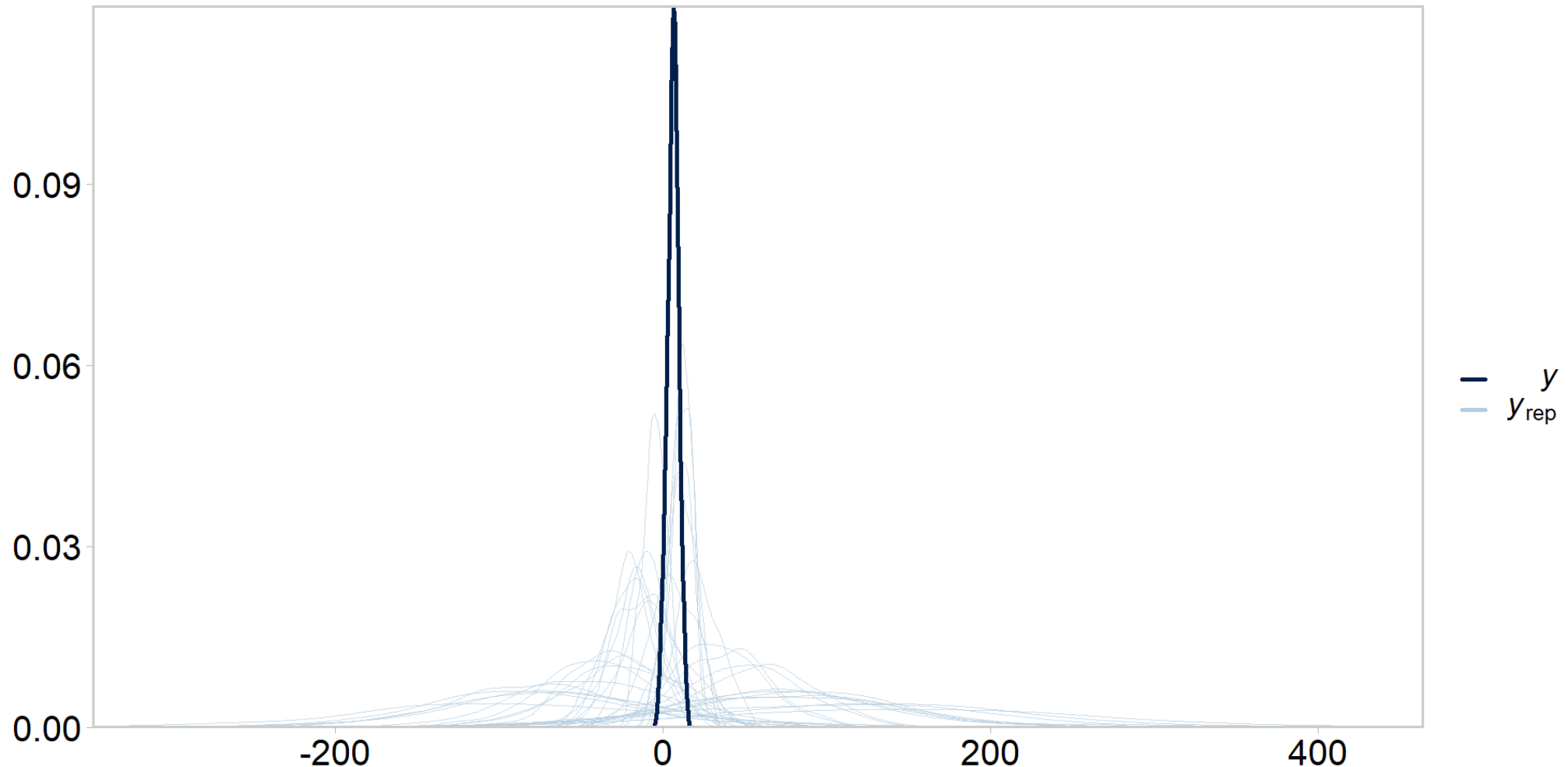
```
1 mod <- cmdstan_model("src/linreg_priorpc.stan")
2
3 fit <- mod$sample(
4   data = stan_data,
5   init = inits,
6   seed = 13,
7   chains = 1,
8   iter_warmup = 1000,
9   iter_sampling = 1000,
10  parallel_chains = 1,
11  refresh = 0
12 )
```

Running MCMC with 1 chain...

Chain 1 finished in 0.1 seconds.

```
1 # extract the draws:
2 prior_draws <- fit$draws(format = "df")
3 y_rep <- prior_draws[grep("y_rep", names(prior_draws))]
4 p <- ppc_dens_overlay(y = data$y, yrep = as.matrix(y_rep[1:35, ]))
```

Prior predictive checks



- Is this reasonable, i.e. consistent with domain expertise?

Data visualization as a key component of a principled Bayesian workflow

- Principled Bayesian workflow is an iterative process of model building (see Betancourt 2018)
- Data visualization is vital to helping us understand the performance of Bayesian models (Gabry et al. 2019)
 - There were some checks we haven't yet done
- In general, need to get good at this stuff if you intend to learn or use Bayesian statistics
- Don't use these tools at your own risk

Summary and outlook

- A presumption...
- We built a linear regression model in Stan, ran it with vague priors
- Spent a substantial amount of time going through diagnostics
 - Split these into MCMC checks and model fit checks
- Spent a great deal of time walking through Bayesian model visualization
 - Touched on how visualization is a core part of a modern Bayesian workflow
- Moving forward, Generalized Linear Models (GLMs)

References

1. Betancourt 2017. [A conceptual introduction to Hamiltonian Monte Carlo](#)
2. Vehtari et al. 2019. [Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC*](#)
3. Gabry et al. 2019. [Visualization in Bayesian workflow](#)
4. Gelman et al. 2021. Bayesian Data Analysis. Edition 3.
5. Kery and Schaub. 2012. Bayesian Population Analysis using WinBUGS.
6. McElreath 2023. Statistical Rethinking. Second Edition.
7. [Stan documentation](#)

Exercises:

1. Calculate the probability that β_1 is < 0.51
2. Calculate the posterior predictive distribution one might expect for y if they went out and collected data at $x_1 = 1.23$
3. Plot the median fit \pm 95% credible intervals from the posterior predictive distribution of y vs. x .
4. Repeat the exercises in this presentation by first simulating your own linear regression dataset to prove to yourself that your code is returning reasonable answers.