Bayesplot

Jonah Gabry

May 05, 2017

Contents

Bayesplot Documentation

bayesplot is an R package providing an extensive library of plotting functions for use after fitting Bayesian models (typically with MCMC). Currently **bayesplot** offers a variety of plots of posterior draws, visual MCMC diagnostics, as well as graphical posterior predictive checking. Additional functionality (e.g. for forecasting/out-of-sample prediction and other inference-related tasks) will be added in future releases.

The plots created by **bayesplot** are ggplot objects, which means that after a plot is created it can be further customized using the various functions for modifying ggplot objects provided by the **ggplot2** package.

The idea behind **bayesplot** is not only to provide convenient functionality for users, but also a common set of functions that can be easily used by developers working on a variety of packages for Bayesian modeling, particularly (but not necessarily) those powered by **RStan**.

RELATED PACKAGES

- RStan
- RStanARM
- Loo

Chapter 1

Posterior Predictive Checking

This vignette focuses on graphical posterior predictive checks (PPC). Plots of parameter estimates from MCMC draws are covered in the separate vignette Plotting MCMC draws using the bayesplot package, and MCMC diagnostics are covered in Visual MCMC diagnostics using the bayesplot package.

1.1 Overview

The **bayesplot** package provides various plotting functions for *graphical posterior predictive checking*, that is, creating graphical displays comparing observed data to simulated data from the posterior predictive distribution.

The idea behind posterior predictive checking is simple: if a model is a good fit then we should be able to use it to generate data that looks a lot like the data we observed.

1.2 Posterior predictive distribution

To generate the data used for posterior predictive checks (PPCs) we simulate from the *posterior predictive distribution* The posterior predictive distribution is the distribution of the outcome variable implied by a model after using the observed data y (a vector of N outcome values) to update our beliefs about unknown model parameters θ. The posterior predictive distribution for observation $\widetilde{y}$ can be written as

$$p(\widetilde{y} \,|\, y) = \int p(\widetilde{y} \,|\, \theta)\, p(\theta \,|\, y)\, d\theta.$$

Typically we will also condition on X (a matrix of predictor variables).

For each draw (simulation) $s = 1, \ldots, S$ of the parameters from the posterior distribution, $\theta^{(s)} \sim p(\theta \,|\, y)$, we draw an entire vector of N outcomes $\widetilde{y}^{(s)}$ from the posterior predictive distribution by simulating from the data model conditional on parameters $\theta^{(s)}$. The result is an $S \times N$ matrix of draws $\widetilde{y}$.

When simulating from the posterior predictive distribution we can use either the same values of the predictors X that we used when fitting the model or new observations of those predictors. When we use the same values of X we denote the resulting simulations by y^{rep}, as they can be thought of as replications of the outcome y rather than predictions for future observations ($\widetilde{y}$ using predictors $\widetilde{X}$). This corresponds to the notation from Gelman et. al. (2013) and is the notation used throughout the package documentation.

1.3 Graphical posterior predictive checks

Using the replicated datasets drawn from the posterior predictive distribution, the functions in the **bayesplot** package create various graphical displays comparing the observed data y to the replications. The names of the **bayesplot** plotting functions for posterior predictive checking all have the prefix `ppc_`.

To demonstrate some of the various PPCs that can be created with the **bayesplot** package we'll use an example of comparing Poisson and Negative binomial regression models from the **rstanarm** package vignette *stan_glm: GLMs for Count Data* (Gabry and Goodrich, 2016).

> We want to make inferences about the efficacy of a certain pest management system at reducing the number of roaches in urban apartments. [...] The regression predictors for the model are the pre-treatment number of roaches `roach1`, the treatment indicator `treatment`, and a variable `senior` indicating whether the apartment is in a building restricted to elderly residents. Because the number of days for which the roach traps were used is not the same for all apartments in the sample, we include it as an exposure [...].

First we fit a Poisson regression model with outcome variable `y` representing the roach count in each apartment at the end of the experiment.

```
library("rstanarm")
head(roaches) # see help("rstanarm-datasets")

roaches$roach1 <- roaches$roach1 / 100 # pre-treatment number of roaches (in 100s)
fit_poisson <- stan_glm(y ~ roach1 + treatment + senior,
                        offset = log(exposure2),
                        family = poisson(link = "log"),
                        data = roaches,
                        seed = 1111)
```

```
print(fit_poisson)
```

```
## stan_glm
##  family:  poisson [log]
##  formula: y ~ roach1 + treatment + senior
## ------
##
## Estimates:
##             Median MAD_SD
## (Intercept)    3.1    0.0
## roach1      6982.9   89.9
## treatment     -0.5    0.0
## senior        -0.4    0.0
##
## Sample avg. posterior predictive
## distribution of y (X = xbar):
##          Median MAD_SD
## mean_PPD 25.7    0.4
##
## ------
## For info on the priors used see help('prior_summary.stanreg').
```

We'll also fit the negative binomial model that we'll compare to the poisson:

```
fit_nb <- update(fit_poisson, family = "neg_binomial_2")
```

```
print(fit_nb)
```

```
## stan_glm
##  family:  neg_binomial_2 [log]
##  formula: y ~ roach1 + treatment + senior
## ------
##
## Estimates:
##                         Median  MAD_SD
## (Intercept)              2.8     0.2
## roach1                13110.0  2416.0
## treatment                -0.8     0.2
## senior                   -0.3     0.2
## reciprocal_dispersion     0.3     0.0
##
## Sample avg. posterior predictive
## distribution of y (X = xbar):
##          Median MAD_SD
## mean_PPD 50.2   29.1
##
## ------
## For info on the priors used see help('prior_summary.stanreg').
```

In order to use the PPC functions from the **bayesplot** package we need a matrix of draws from the posterior predictive distribution. Since we fit the models using **rstanarm** we can use its `posterior_predict` function:

```
yrep_poisson <- posterior_predict(fit_poisson, draws = 500)
yrep_nb <- posterior_predict(fit_nb, draws = 500)
dim(yrep_poisson)
```

```
## [1] 500 262
```

```
dim(yrep_nb)
```

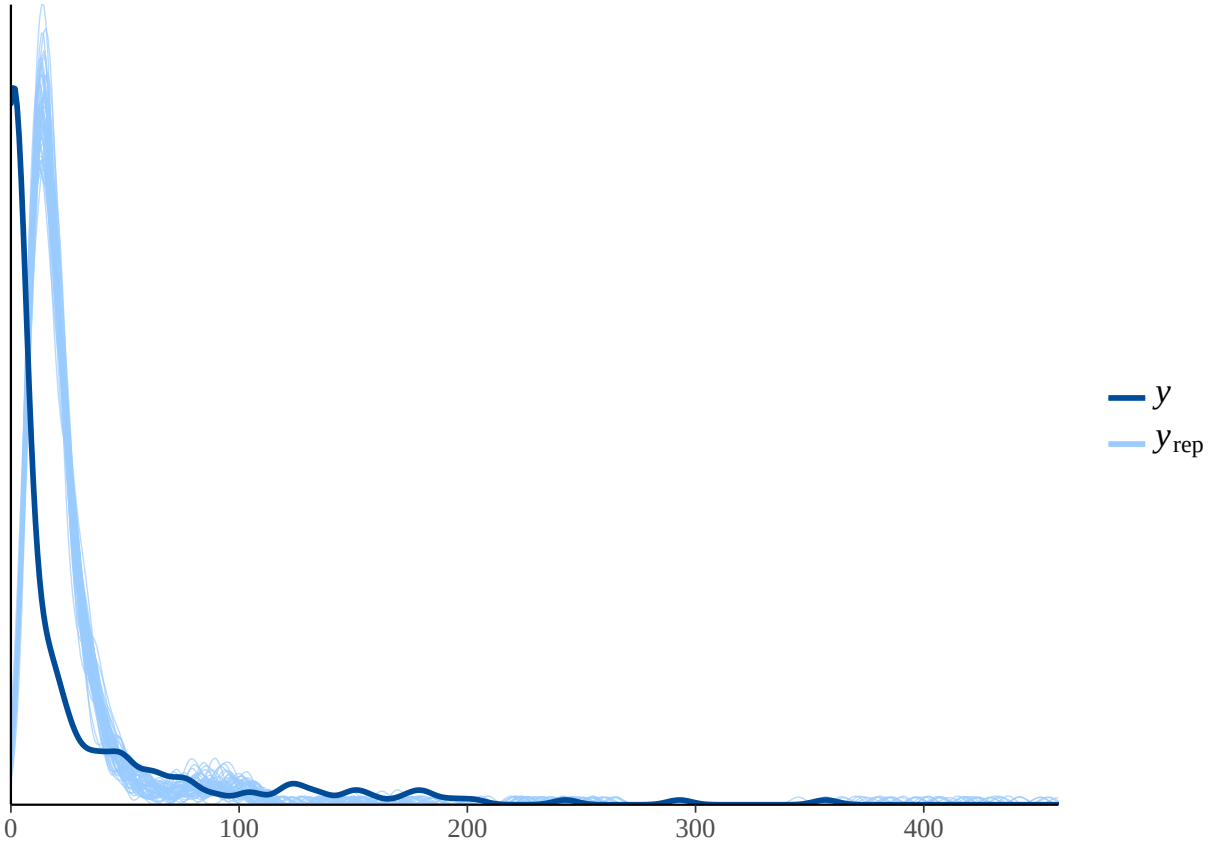
```
## [1] 500 262
```

For each of the `yrep` matrices, every row is a draw from the posterior predictive distribution, i.e. a vector with one element for each of the data points in `y`.

1.3.1 Histograms and density estimates

The first PPC we'll look at is a comparison of the distribution of `y` and the distributions of some of the simulated datasets (rows) in the `yrep` matrix.

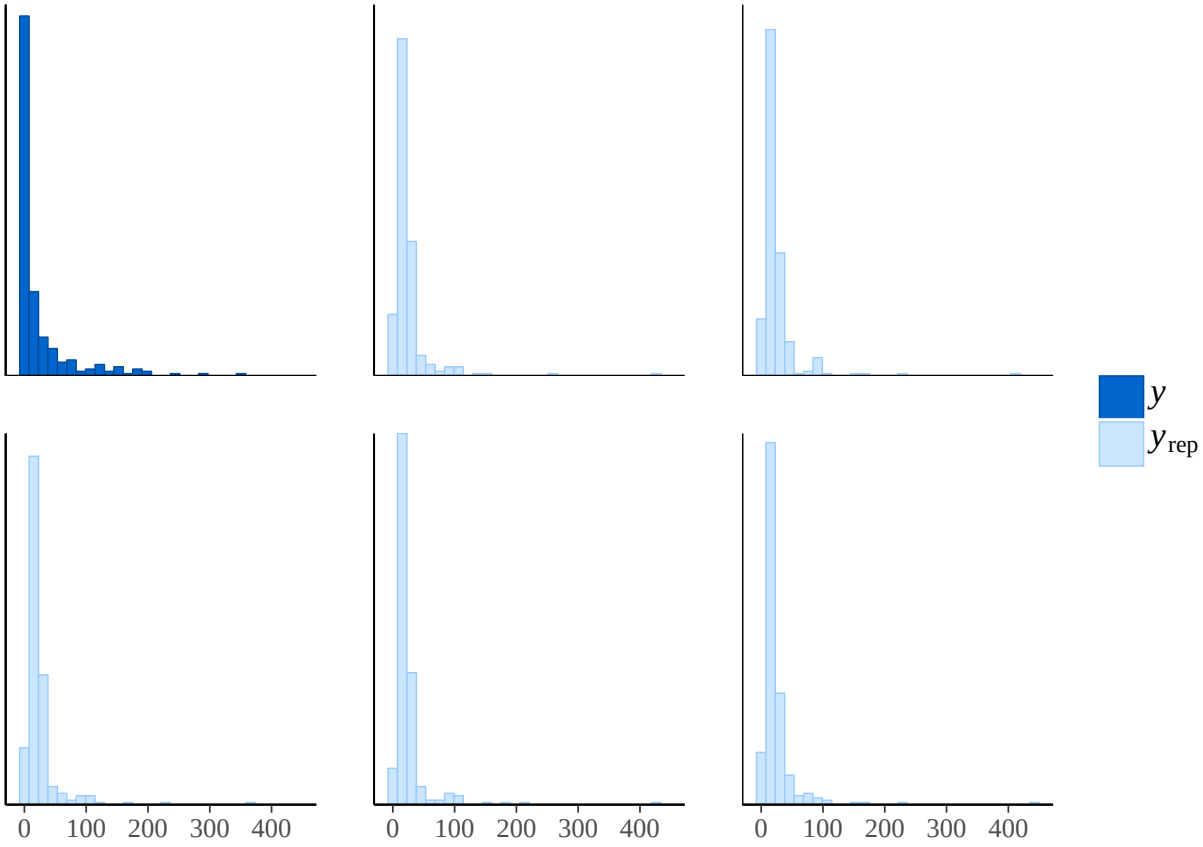
```
library("ggplot2")
library("bayesplot")

color_scheme_set("brightblue") # see help("bayesplot-colors")

y <- roaches$y
ppc_dens_overlay(y, yrep_poisson[1:50, ])
```



In the plot above, the dark line is the distribution of the observed outcomes `y` and each of the 50 lighter lines is the kernel density estimate of one of the replications of `y` from the posterior predictive distribution (i.e., one of the rows in `yrep`). This plot makes it easy to see that this model fails to account for large proportion of zeros in `y`. That is, the model predicts fewer zeros than were actually observed.

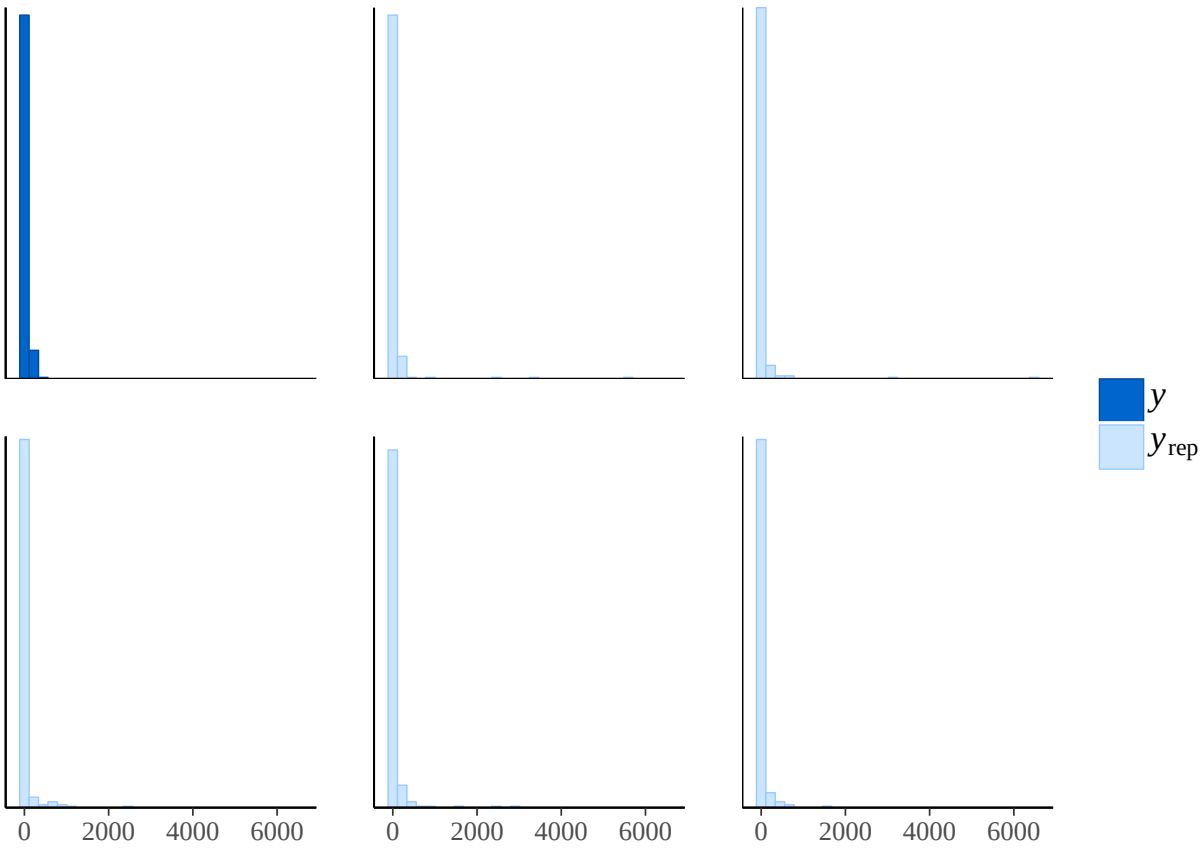
We could see the same thing by looking at separate histograms of `y` and some of the `yrep` datasets using the `ppc_hist` function:

```
ppc_hist(y, yrep_poisson[1:5, ])
```



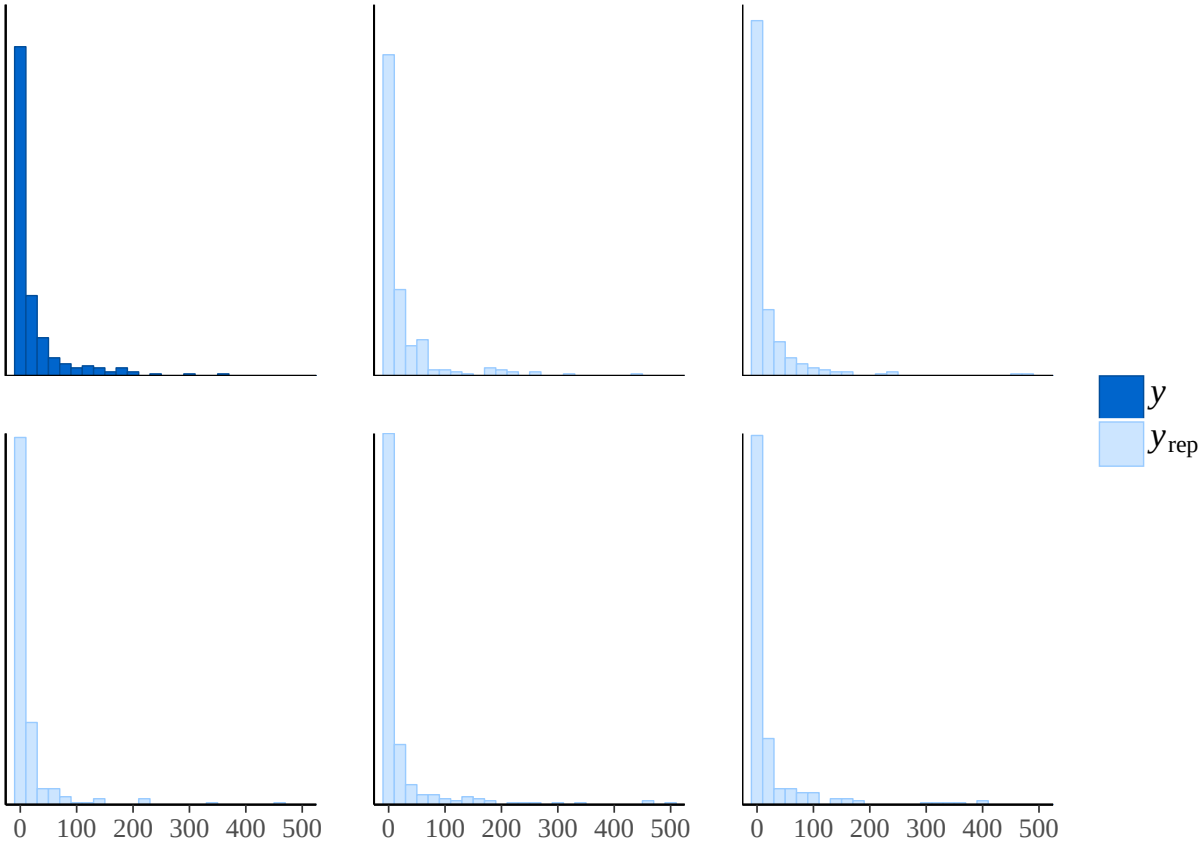
The same plot for the negative binomial model looks much different:

```
ppc_hist(y, yrep_nb[1:5, ])
```



The negative binomial model does better handling the number of zeros in the data, but it occasionally predicts values that are way too large, which is why the x-axes extend to such high values in the plot and make it difficult to read. To see the predictions for the smaller values more clearly we can zoom in:

```
ppc_hist(y, yrep_nb[1:5, ], binwidth = 20) +
  coord_cartesian(xlim = c(-1, 500))
```



1.3.2 Distributions of test statistics

Another way to see that the Poisson model predicts too few zeros is to use the `ppc_stat` function to look at the distribution of the proportion of zeros over the replicated datasets from the posterior predictive distribution in `yrep` and compare to the proportion of observed zeros in `y`.

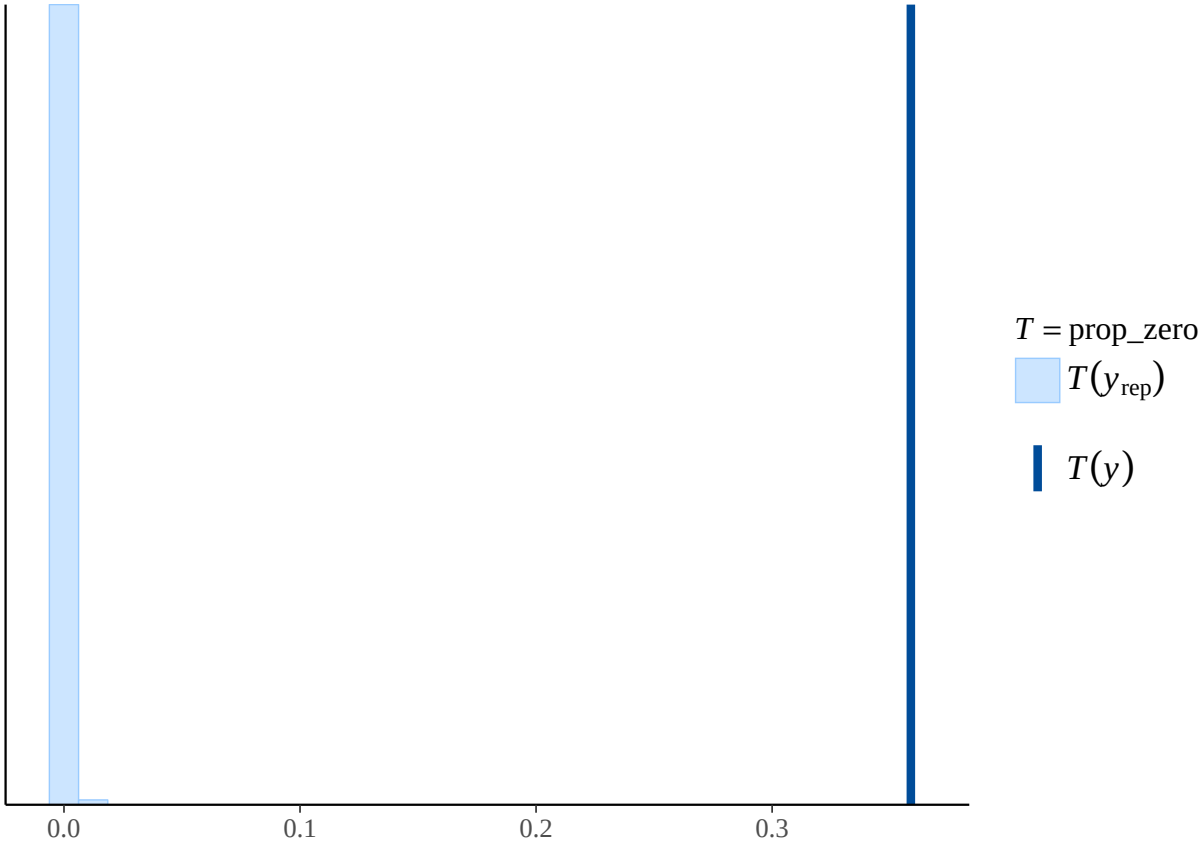
First we define a function that takes a vector as input and returns the proportion of zeros:

```
prop_zero <- function(x) mean(x == 0)
prop_zero(y) # check proportion of zeros in y
```

```
## [1] 0.3587786
```

Then we can use this function as the `stat` argument to `ppc_stat`:

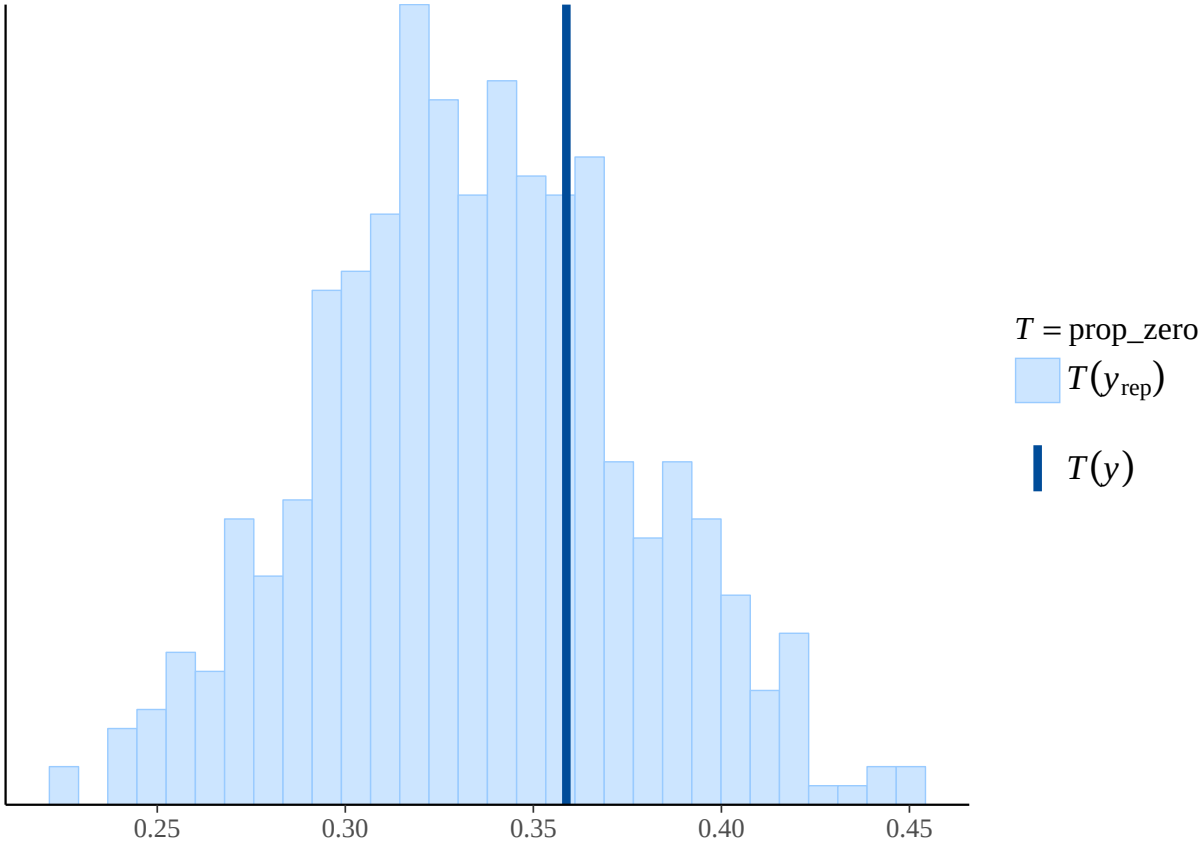
```
ppc_stat(y, yrep_poisson, stat = "prop_zero")
```



In the plot the dark line is at the value $T(y)$, i.e. the value of the test statistic computed from the observed y, in this case `prop_zero(y)`. It's hard to see because almost all the datasets in `yrep` have no zeros, but the lighter bar is actually a histogram of the the proportion of zeros in each of the replicated datasets.

Here's the same plot for the negative binomial model:

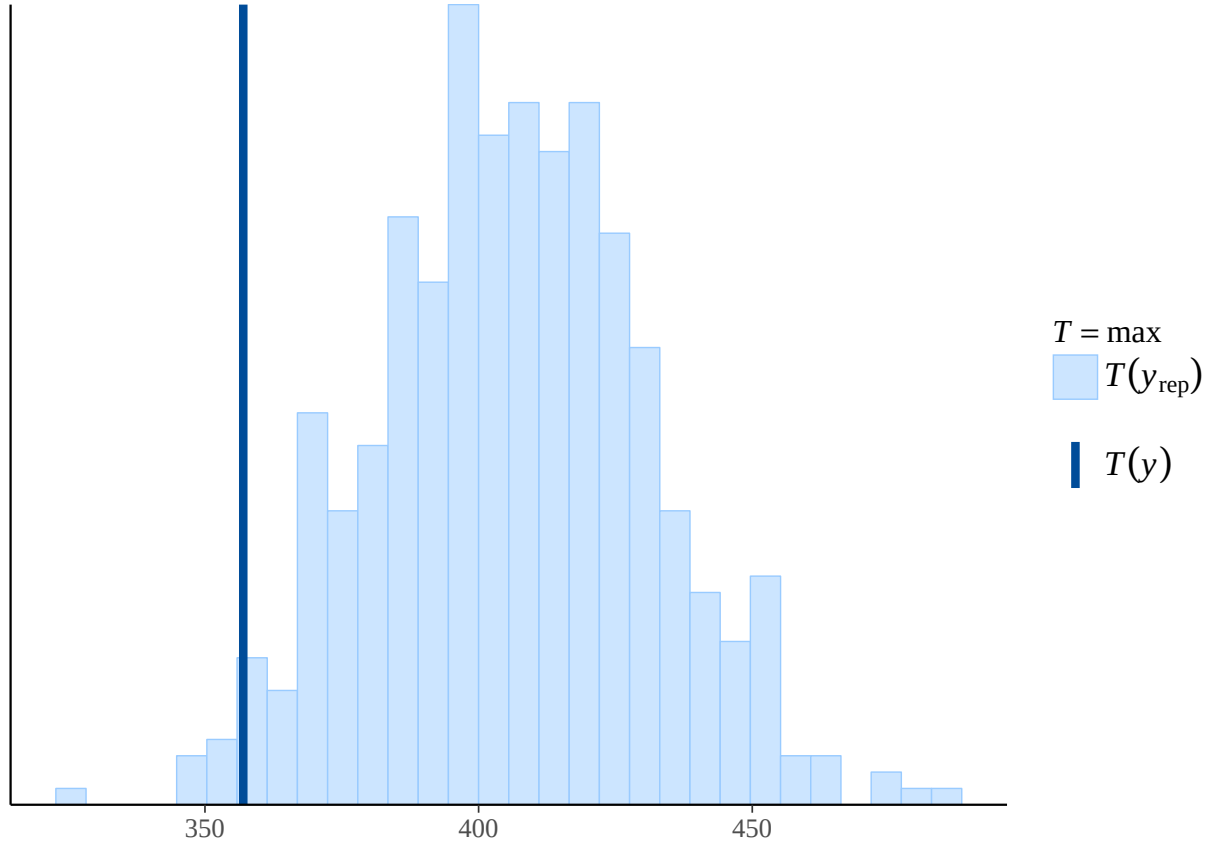
```
ppc_stat(y, yrep_nb, stat = "prop_zero")
```



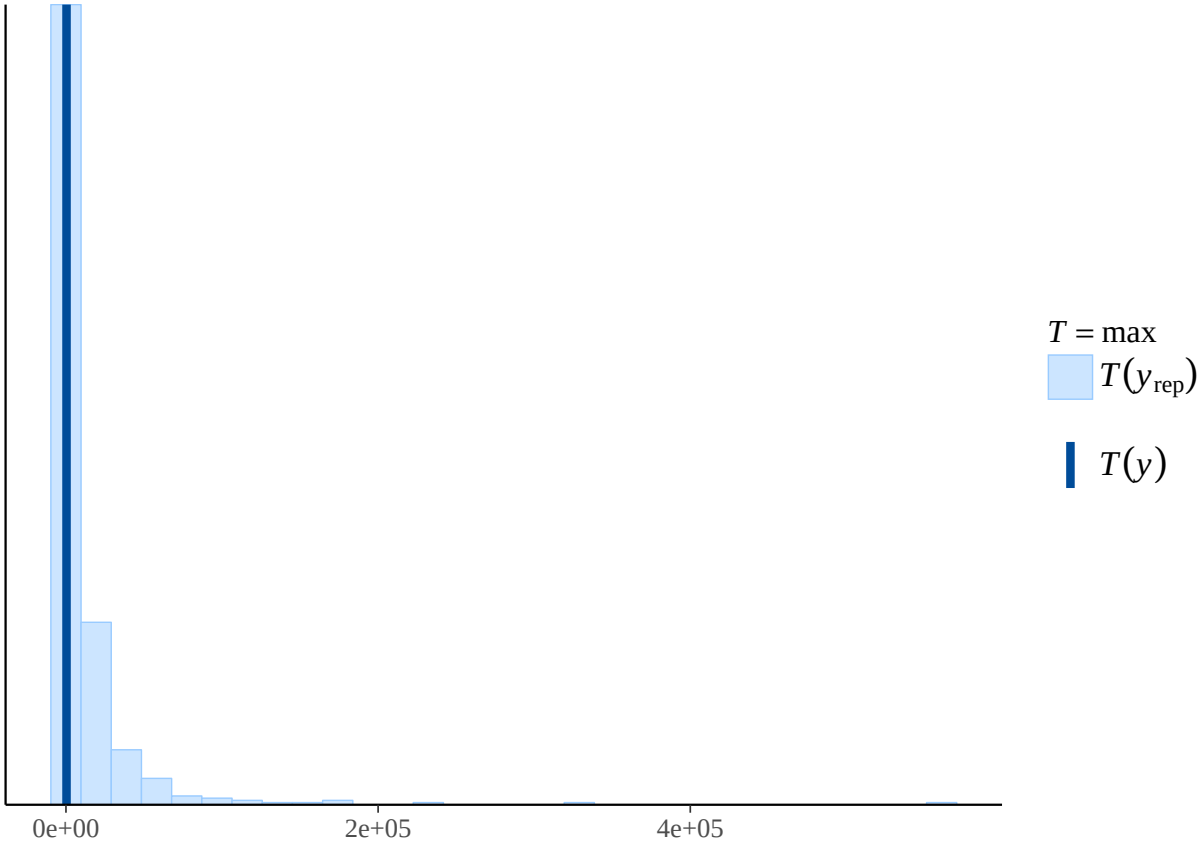
Again we see that the negative binomial model does a much better job predicting the proportion of observed zeros than the Poisson.

However, if we look instead at the distribution of the maximum value in the replications then we can see that the Poisson model makes more realistic predictions than the negative binomial:

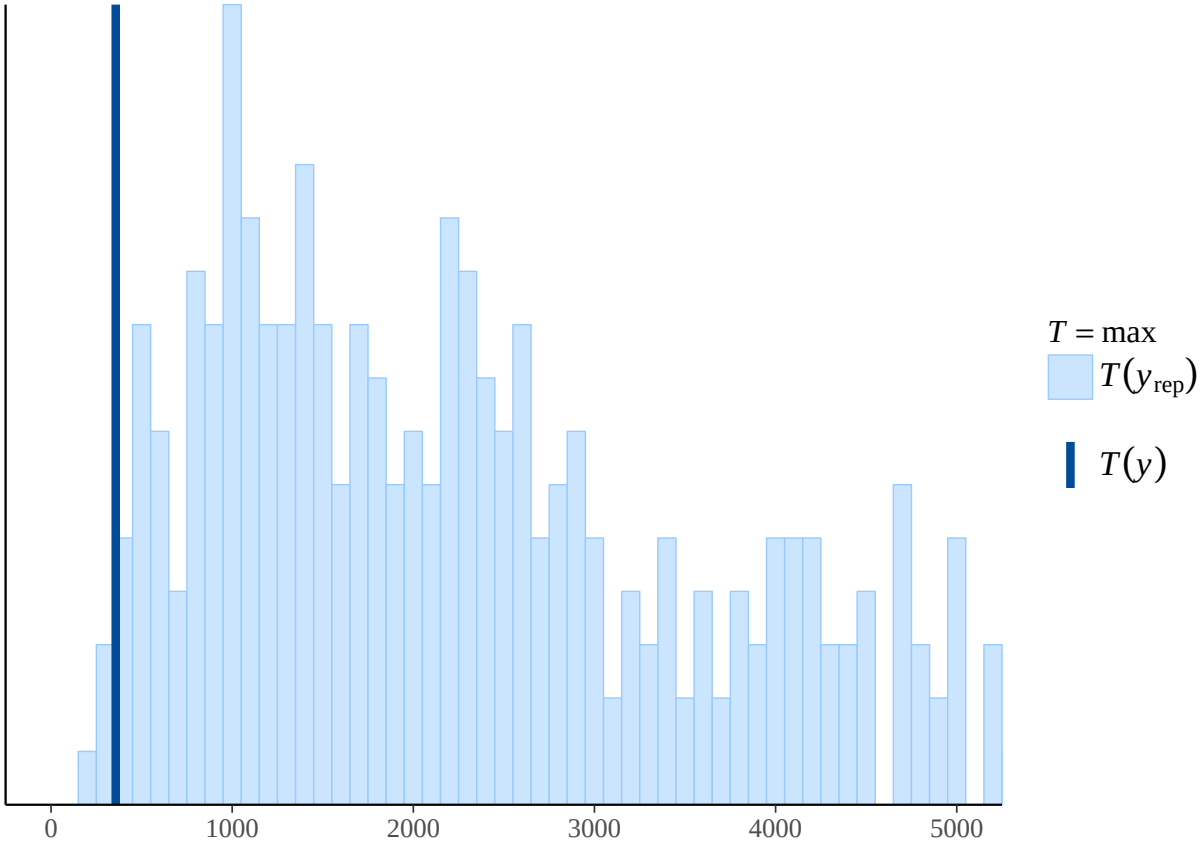
```
ppc_stat(y, yrep_poisson, stat = "max")
```




```
ppc_stat(y, yrep_nb, stat = "max")
```




```
ppc_stat(y, yrep_nb, stat = "max", binwidth = 100) +
  coord_cartesian(xlim = c(-1, 5000))
```



1.3.3 Other PPCs and PPCs by group

There are many additional PPCs available, including plots of predictive intervals, distributions of predictive errors, and more. For links to the documentation for all of the various PPC plots see `help("PPC-overview")`. The `available_ppc` function can also be used to list the names of all PPC plotting functions:

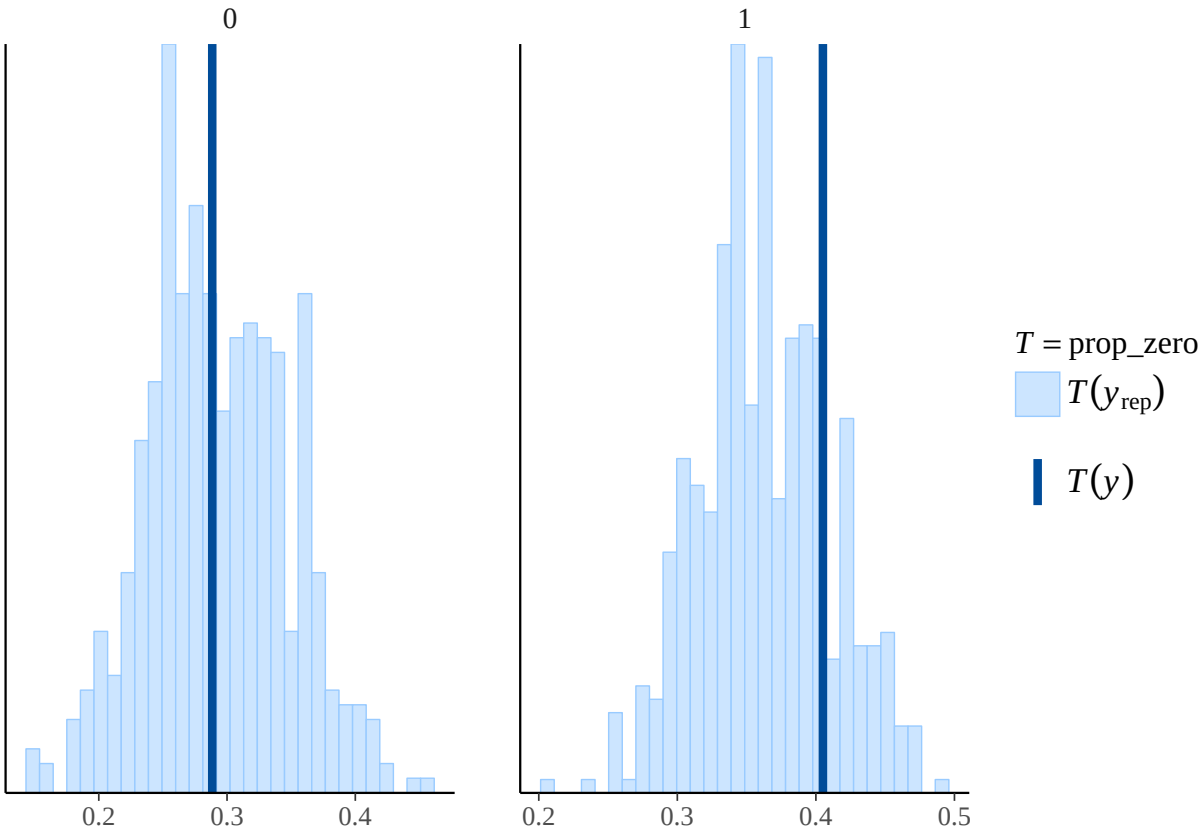
```
available_ppc()
```

```
## bayesplot PPC module:
##   ppc_bars
##   ppc_bars_grouped
##   ppc_boxplot
##   ppc_dens
##   ppc_dens_overlay
##   ppc_ecdf_overlay
##   ppc_error_binned
##   ppc_error_hist
##   ppc_error_hist_grouped
##   ppc_error_scatter
##   ppc_error_scatter_avg
##   ppc_error_scatter_avg_vs_x
##   ppc_freqpoly
##   ppc_freqpoly_grouped
##   ppc_hist
##   ppc_intervals
##   ppc_intervals_grouped
```

```
##   ppc_loo_intervals
##   ppc_loo_pit
##   ppc_loo_ribbon
##   ppc_ribbon
##   ppc_ribbon_grouped
##   ppc_rootogram
##   ppc_scatter
##   ppc_scatter_avg
##   ppc_scatter_avg_grouped
##   ppc_stat
##   ppc_stat_2d
##   ppc_stat_freqpoly_grouped
##   ppc_stat_grouped
##   ppc_violin_grouped
```

Many of the available PPCs can also be carried out within levels of a grouping variable. Any function for PPCs by group will have a name ending in `_grouped` and will accept an additional argument `group`. For example, `ppc_stat_grouped` is the same as `ppc_stat` except that the test statistics are computed within levels of the grouping variable and a separate plot is made for each level:

```
ppc_stat_grouped(y, yrep_nb, group = roaches$treatment, stat = "prop_zero")
```



The full list of currently available `_grouped` functions is:

```
available_ppc(pattern = "_grouped")
```

```
## bayesplot PPC module:
## (matching pattern '_grouped')
##   ppc_bars_grouped
```

```
##   ppc_error_hist_grouped
##   ppc_freqpoly_grouped
##   ppc_intervals_grouped
##   ppc_ribbon_grouped
##   ppc_scatter_avg_grouped
##   ppc_stat_freqpoly_grouped
##   ppc_stat_grouped
##   ppc_violin_grouped
```

1.4 Providing an interface to bayesplot PPCs from another package

The **bayesplot** package provides the S3 generic function `pp_check`. Authors of R packages for Bayesian inference are encouraged to define methods for the fitted model objects created by their packages. This will hopefully be convenient for both users and developers and contribute to the use of the same naming conventions across many of the R packages for Bayesian data analysis.

To provide an interface to **bayesplot** from your package, you can very easily define a `pp_check` method (or multiple `pp_check` methods) for the fitted model objects created by your package. All a `pp_check` method needs to do is provide the `y` vector and `yrep` matrix arguments to the various plotting functions included in **bayesplot**.

1.4.1 Defining a `pp_check` method

Here is an example for how to define a simple `pp_check` method in a package that creates fitted model objects of class `"foo"`. We will define a method `pp_check.foo` that extracts the data `y` and the draws from the posterior predictive distribution `yrep` from an object of class `"foo"` and then calls one of the plotting functions from **bayesplot**.

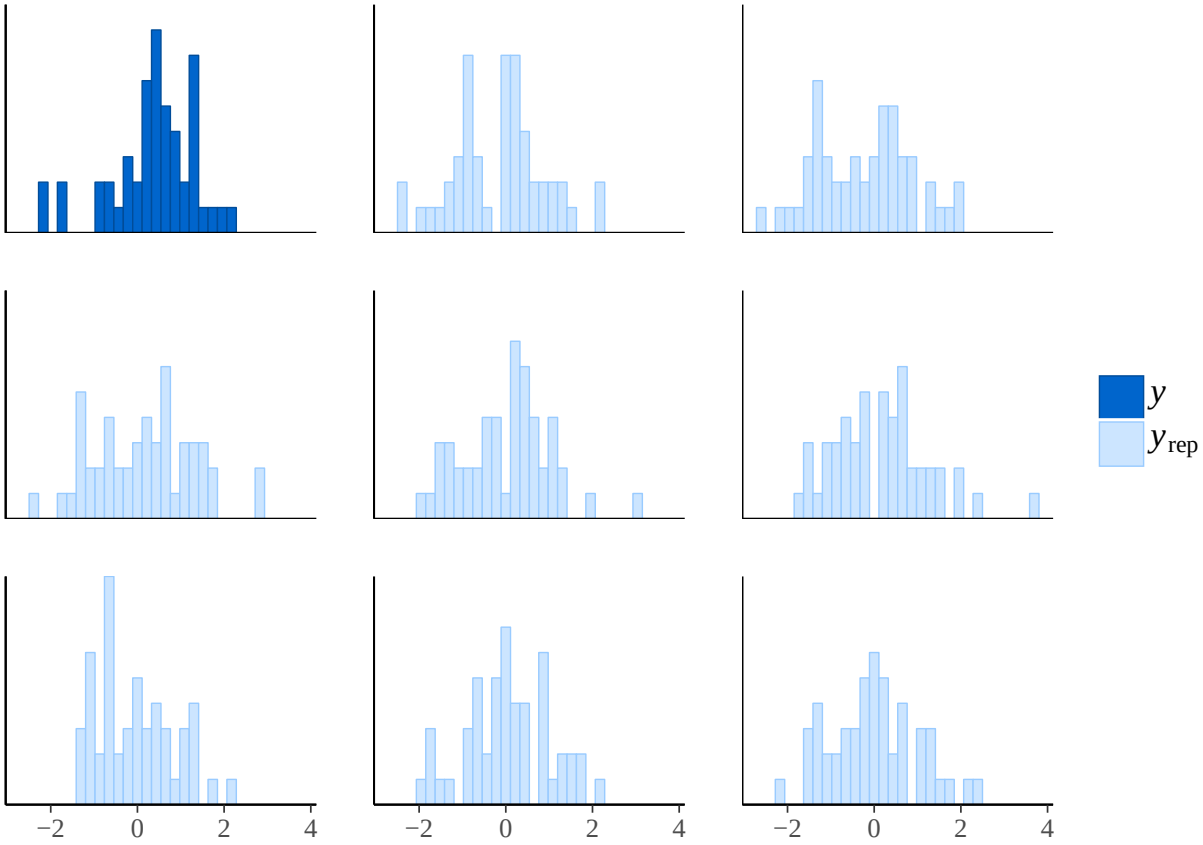
Suppose that objects of class `"foo"` are lists with named components, two of which are `y` and `yrep`. Here's a simple method `pp_check.foo` that offers the user the option of two different plots:

```
pp_check.foo <- function(object, ..., type = c("multiple", "overlaid")) {
  y <- object[["y"]]
  yrep <- object[["yrep"]]
  switch(
    match.arg(type),
    multiple = ppc_hist(y, yrep[1:min(8, nrow(yrep)),, drop = FALSE]),
    overlaid = ppc_dens_overlay(y, yrep)
  )
}
```

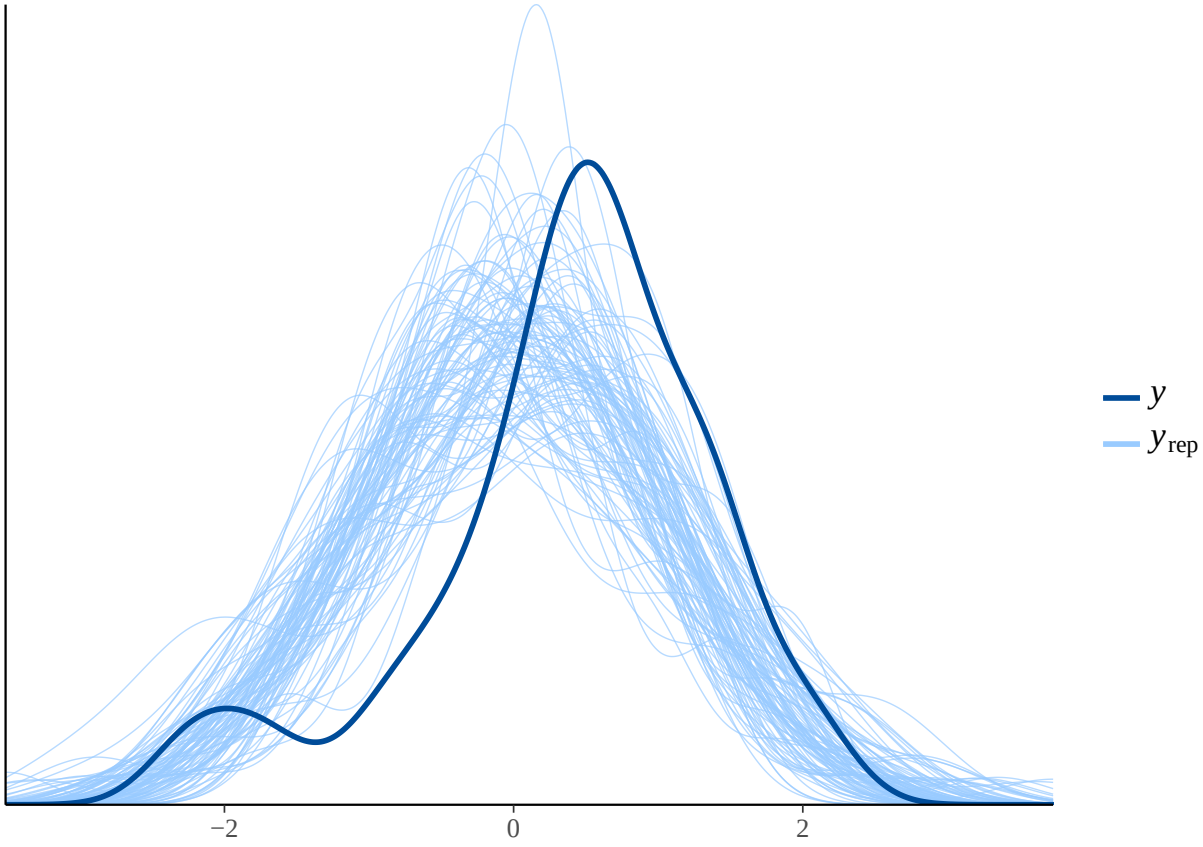
To try out `pp_check.foo` we can just make a list with `y` and `yrep` components and give it class `foo`:

```
x <- list(y = rnorm(50), yrep = matrix(rnorm(5000), nrow = 100, ncol = 50))
class(x) <- "foo"
```

```
pp_check(x)
```




```
pp_check(x, type = "overlaid")
```



1.4.2 Examples of `pp_check` methods in other packages

Several packages currently (or will soon) use this approach to provide an interface to **bayesplot**'s graphical posterior predictive checks. See, for example, the `pp_check` methods in the **rstanarm** and **brms** packages.

Chapter 2

Plotting MCMC Draws

This vignette focuses on plotting parameter estimates from MCMC draws. MCMC diagnostic plots are covered in the separate vignette Visual MCMC diagnostics using the bayesplot package, and graphical posterior predictive checks are covered in Graphical posterior predictive checks using the bayesplot package.

2.1 Plots for MCMC draws

The **bayesplot** package provides various plotting functions for visualizing Markov chain Monte Carlo (MCMC) draws from the posterior distribution of the parameters of a Bayesian model.

In this vignette we'll use draws obtained using the `stan_glm` function in the **rstanarm** package (Gabry and Goodrich, 2016), but MCMC draws from using any package can be used with the functions in the **bayesplot** package. See, for example, **brms** (which, like **rstanarm**, calls the **rstan** package internally to use Stan's MCMC sampler).

```
library("rstanarm")
fit <- stan_glm(
  mpg ~ ., # ~ . includes all other variables in dataset
  data = mtcars,
  chains = 4,
  iter = 2000,
  seed = 1111
)
print(fit)
```

```
## stan_glm
##  family:  gaussian [identity]
##  formula: mpg ~ .
## ------
##
## Estimates:
##             Median MAD_SD
## (Intercept) 12.9   19.9
## cyl         -0.1    1.1
## disp         0.0    0.0
## hp           0.0    0.0
## drat         0.8    1.6
## wt          -3.7    1.9
```

```
## qsec         0.8    0.8
## vs           0.3    2.1
## am           2.5    2.1
## gear         0.7    1.6
## carb        -0.2    0.8
## sigma        2.7    0.4
##
## Sample avg. posterior predictive
## distribution of y (X = xbar):
##          Median MAD_SD
## mean_PPD 20.1    0.7
##
## ------
## For info on the priors used see help('prior_summary.stanreg').
```

To use the posterior draws with the functions in the **bayesplot** package we'll extract them from the fitted model object:

```
posterior <- as.array(fit)
dim(posterior)
```

```
## [1] 1000    4   12
```

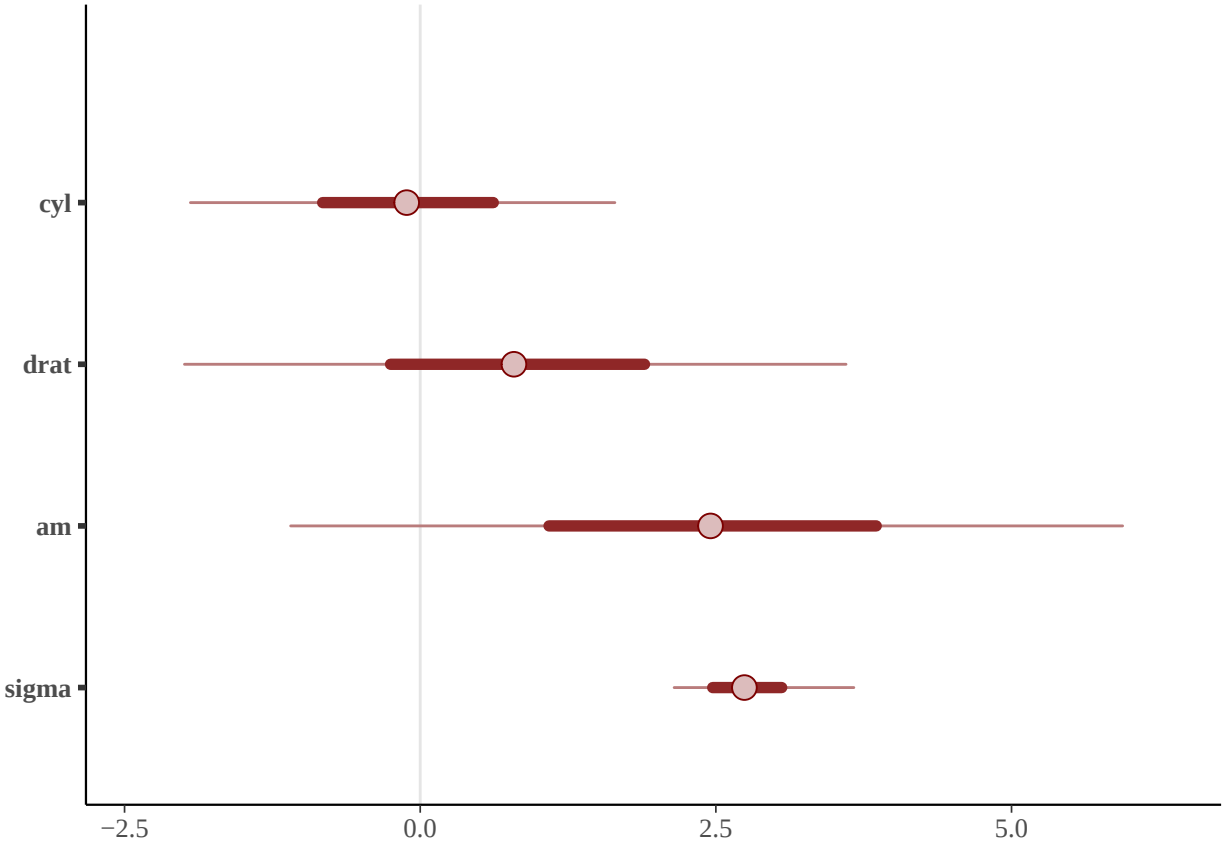
```
dimnames(posterior)
```

```
## $iterations
## NULL
##
## $chains
## [1] "chain:1" "chain:2" "chain:3" "chain:4"
##
## $parameters
##  [1] "(Intercept)" "cyl"         "disp"        "hp"          "drat"
##  [6] "wt"          "qsec"        "vs"          "am"          "gear"
## [11] "carb"        "sigma"
```

2.1.1 Interval estimates

Posterior intervals for the parameters can be plotted using the `mcmc_intervals` function.

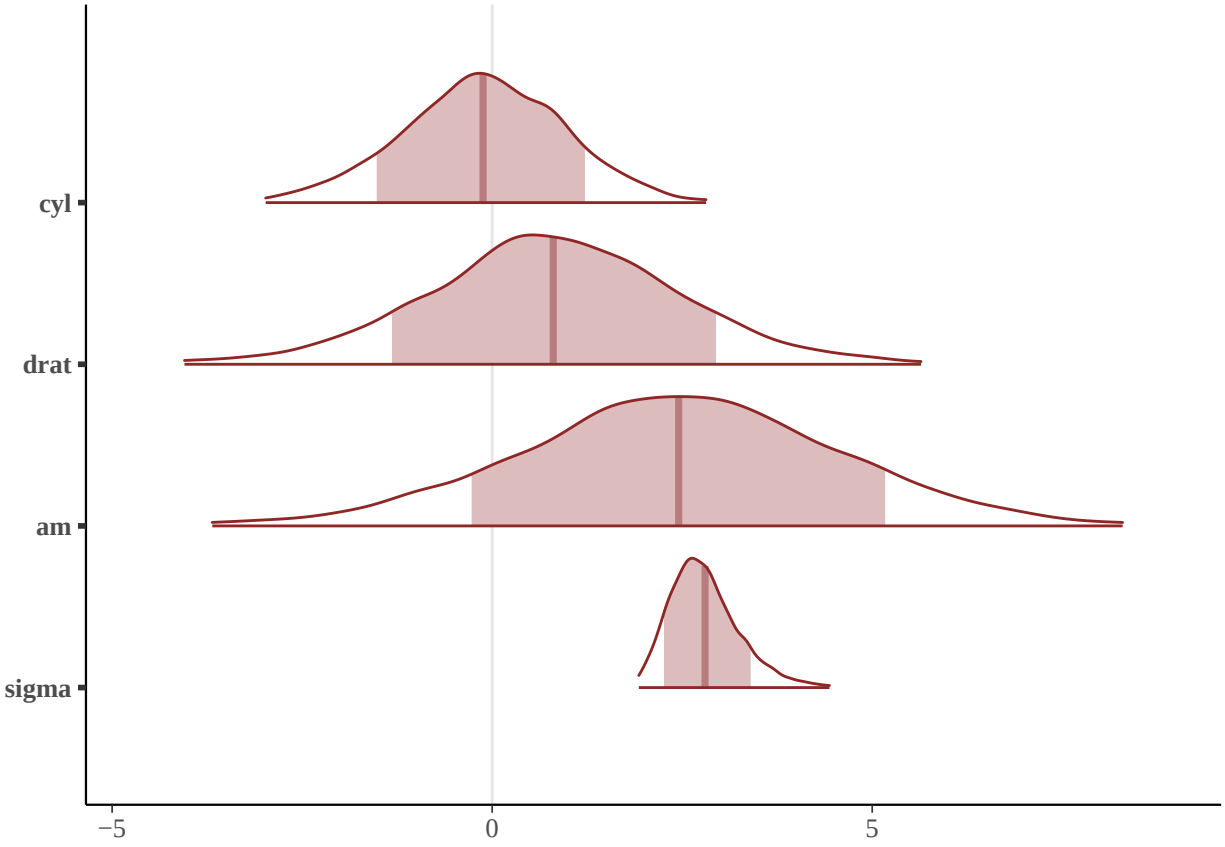
```
library("bayesplot")
color_scheme_set("red")
mcmc_intervals(posterior, pars = c("cyl", "drat", "am", "sigma"))
```



The default is to show 50% intervals (the thick lines) and 90% intervals (the thinner outer lines). These defaults can be changed using the `prob` and `prob_outer` arguments, respectively. The points in the above plot are posterior medians. The `point_est` argument can be used to select posterior means instead or to omit the point estimates.

To show the uncertainty intervals as shaded areas under the estimated posterior density curves we can use the `mcmc_areas` function:

```
mcmc_areas(
  posterior,
  pars = c("cyl", "drat", "am", "sigma"),
  prob = 0.8, # 80% intervals
  prob_outer = 0.99, # 99%
  point_est = "mean"
)
```



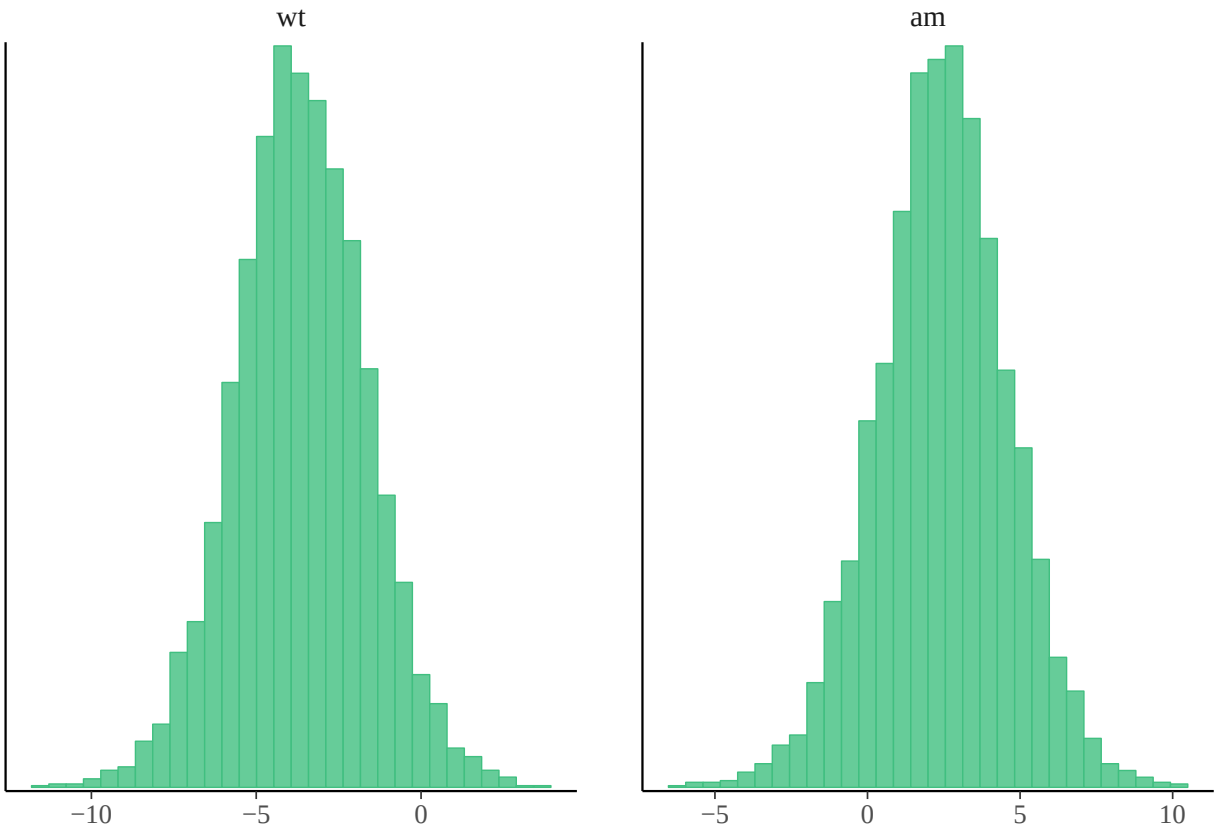
2.1.2 Histograms and density estimates

Histograms or kernel density estimates of posterior distributions of the various model parameters can be visualized using the functions described on the `MCMC-distributions` page in the **bayesplot** documentation.

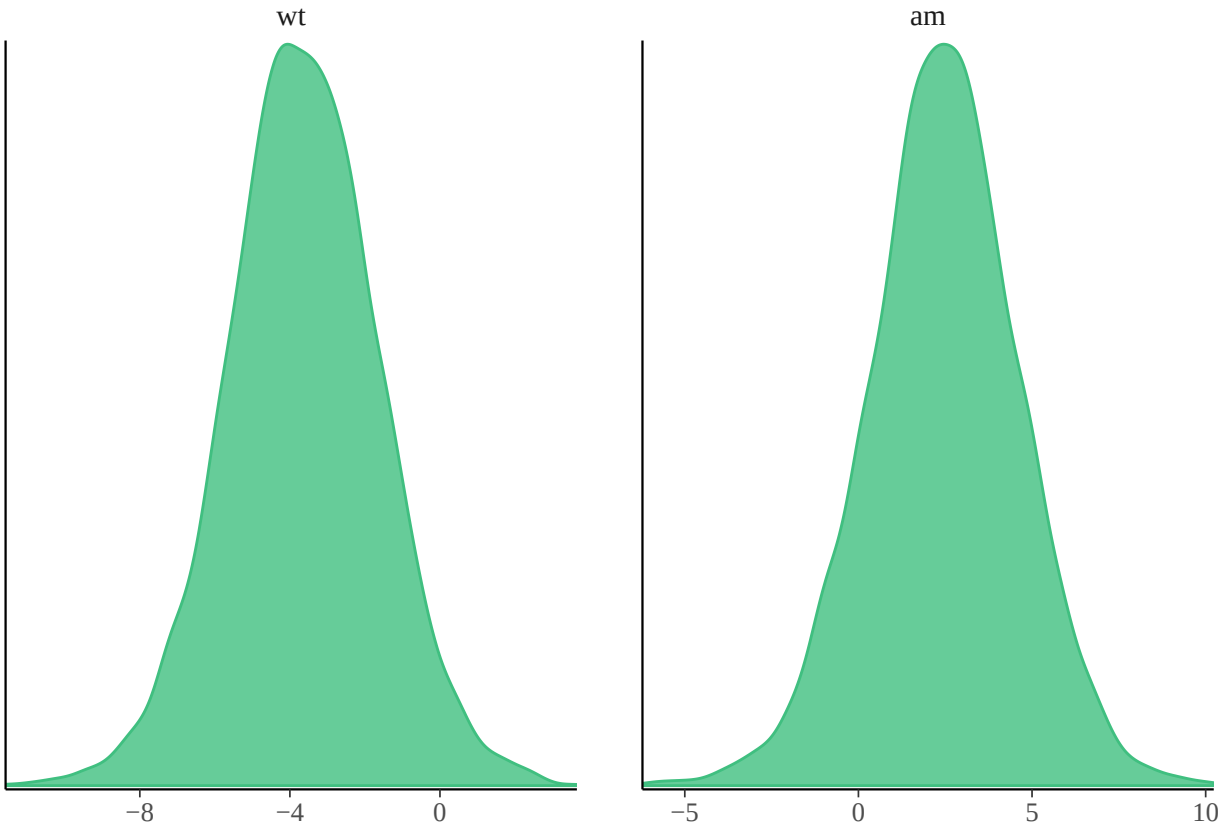
2.1.2.1 Histograms

The `mcmc_hist` and `mcmc_dens` functions plot posterior distributions (combining all chains):

```
color_scheme_set("green")
mcmc_hist(posterior, pars = c("wt", "am"))
```

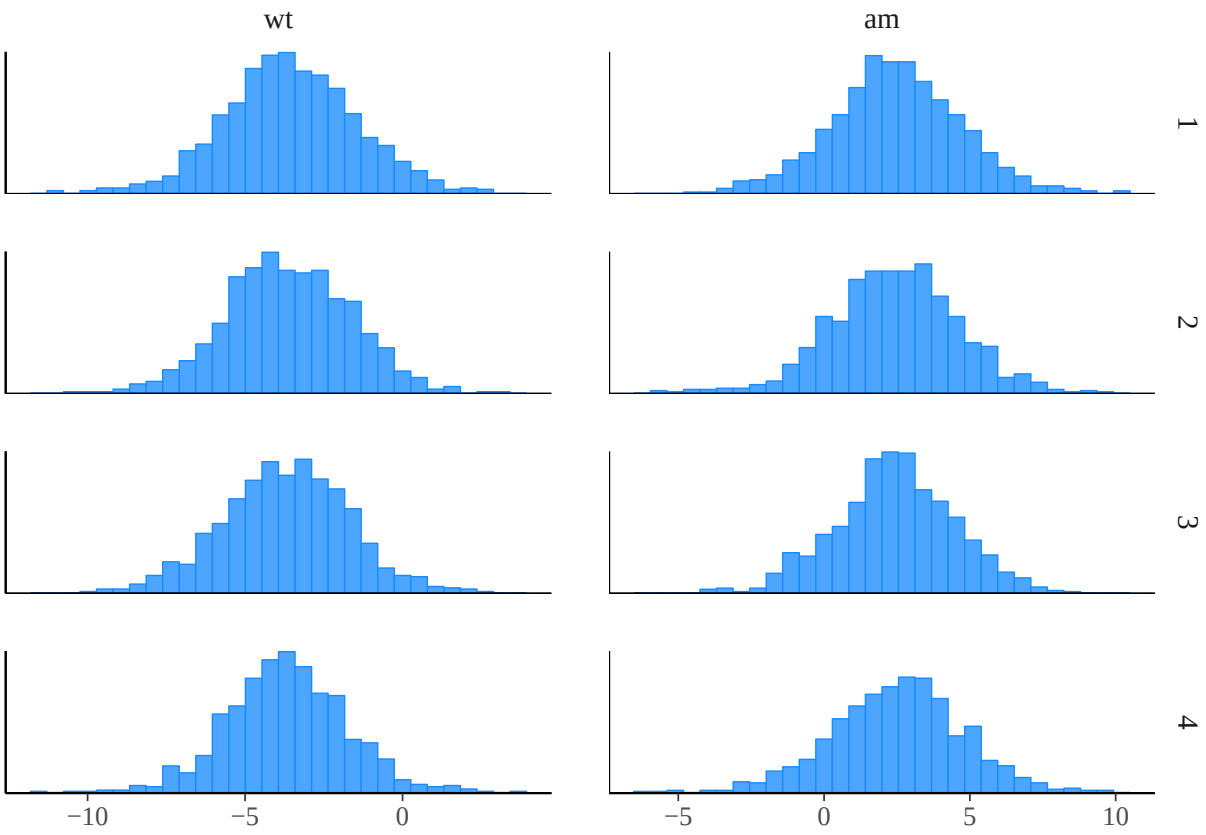



```
mcmc_dens(posterior, pars = c("wt", "am"))
```

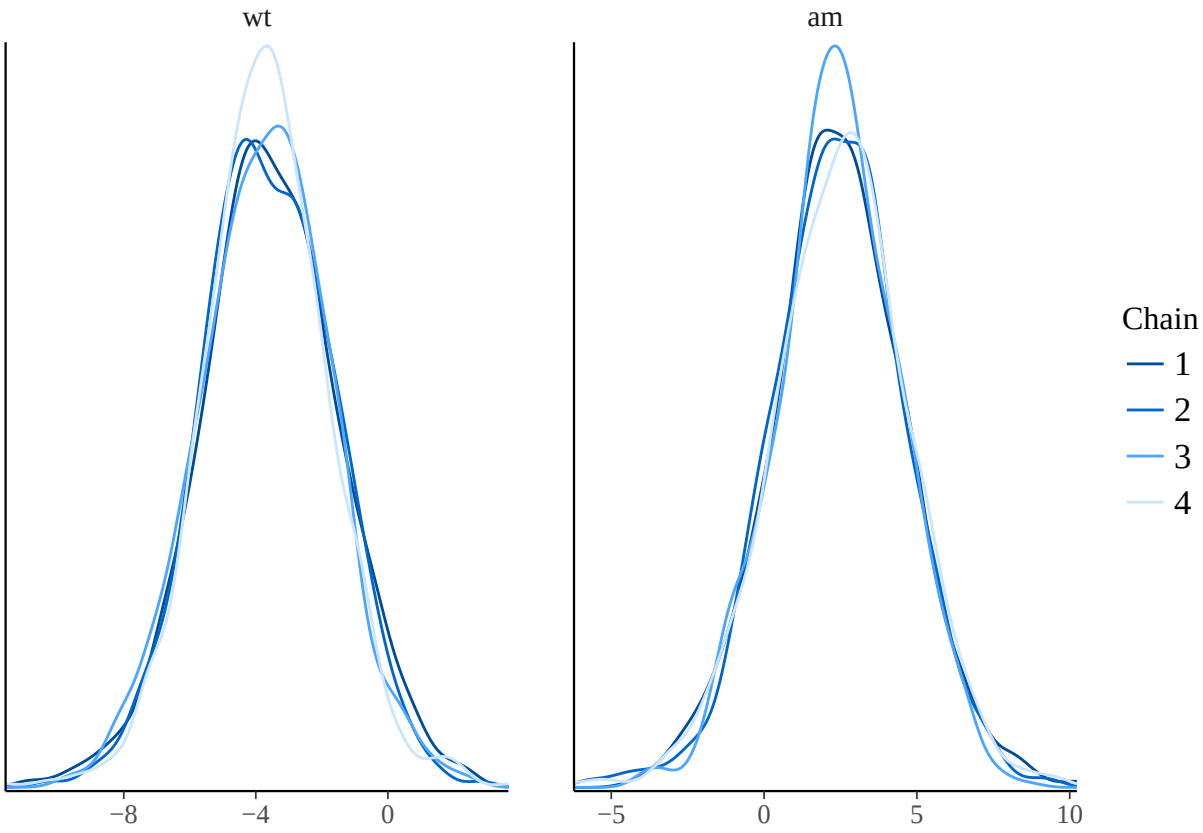


To view the four Markov chain separately we can use `mcmc_hist_by_chain`, `mcmc_dens_overlay`:

```
color_scheme_set("brightblue")
mcmc_hist_by_chain(posterior, pars = c("wt", "am"))
```

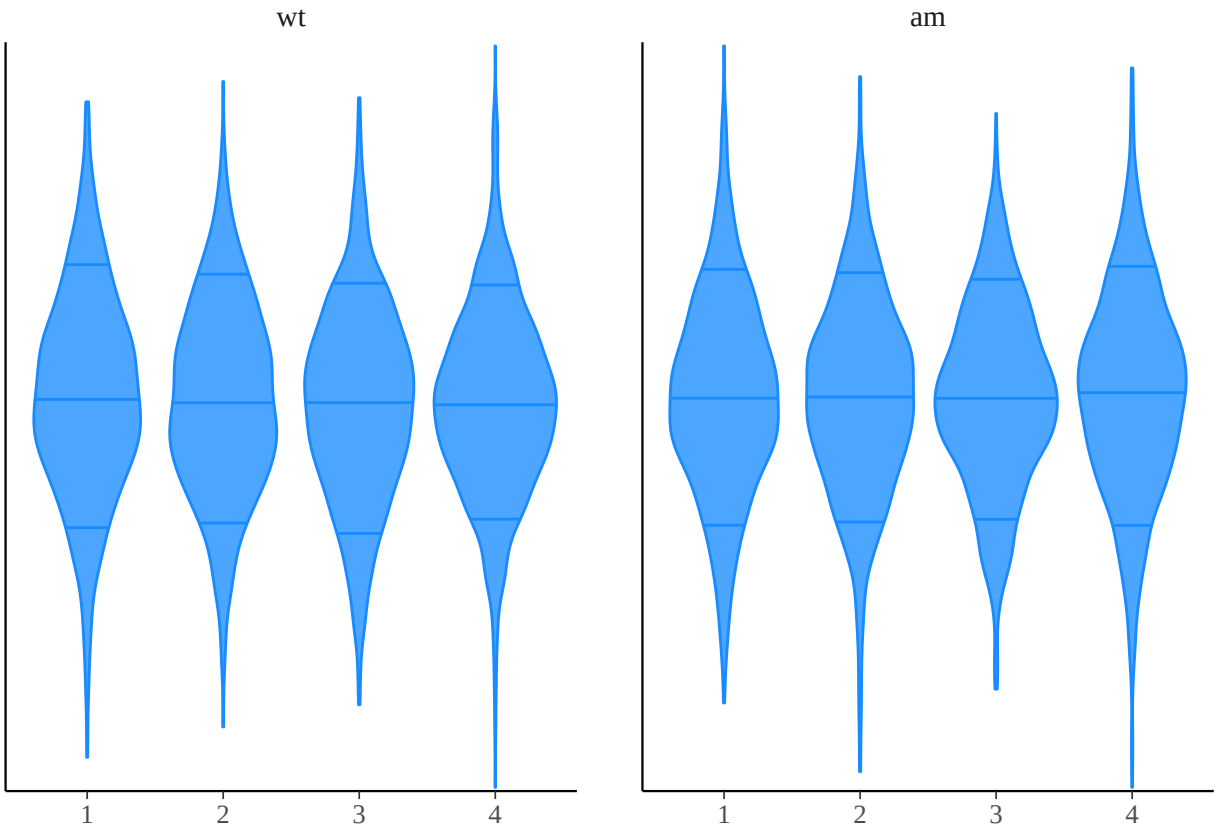



```
mcmc_dens_overlay(posterior, pars = c("wt", "am"))
```



The `mcmc_violin` function also plots the density estimates of each chain as violins with horizontal lines at user-specified quantiles:

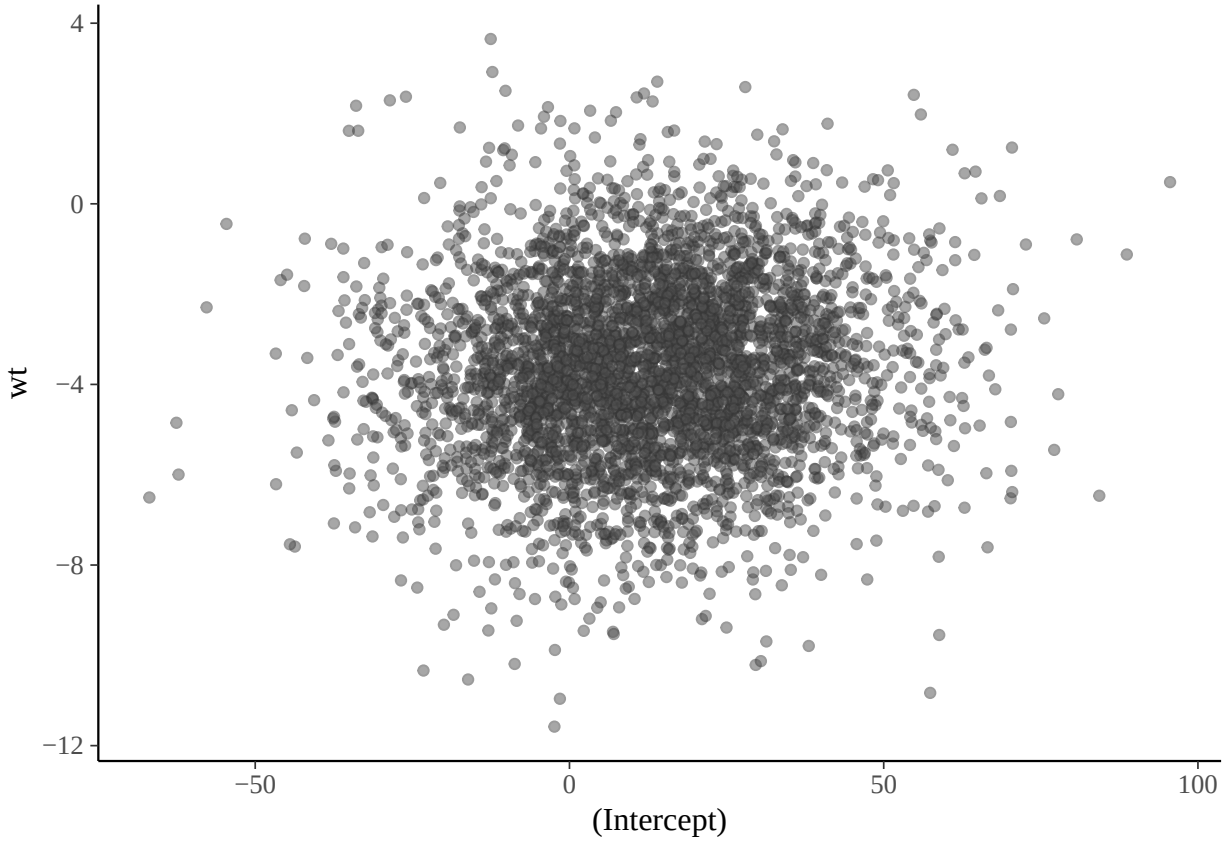
```
mcmc_violin(posterior, pars = c("wt", "am"), probs = c(0.1, 0.5, 0.9))
```



2.1.3 Scatterplots

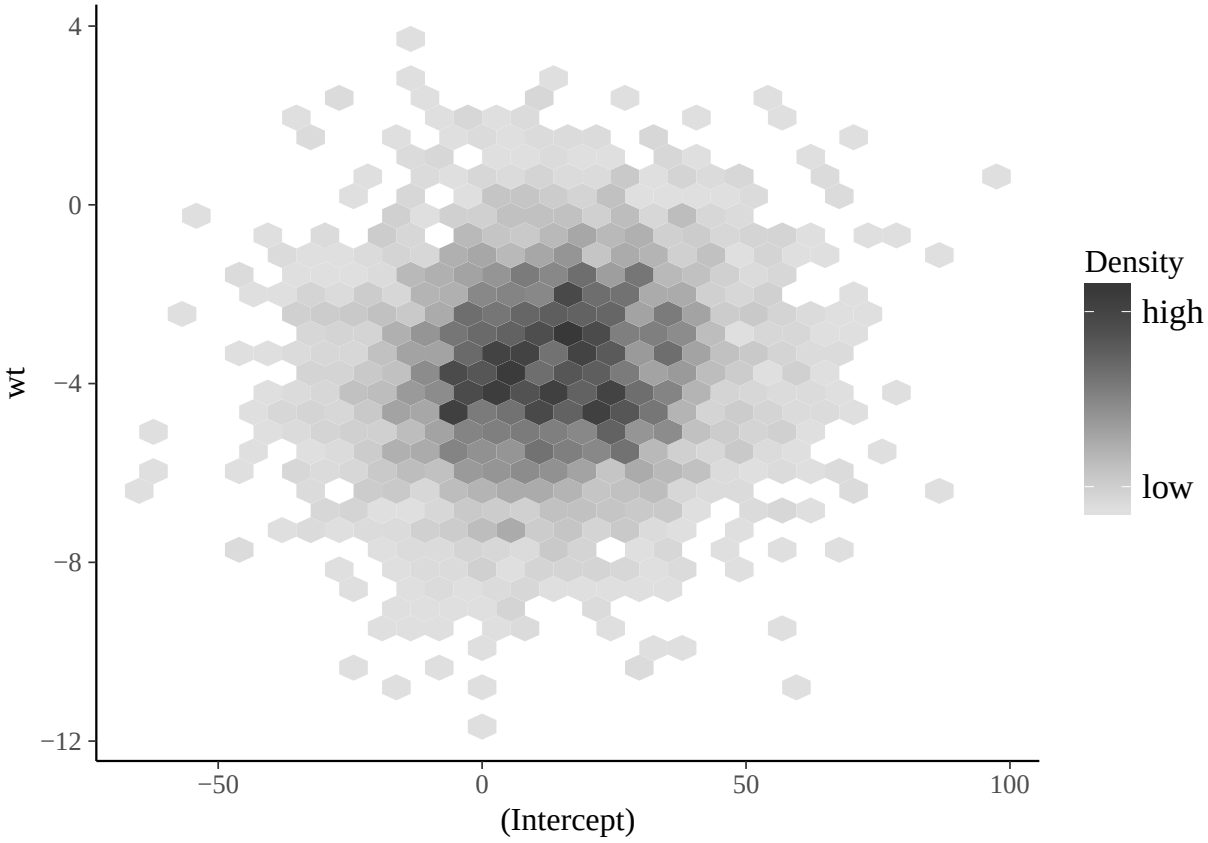
The `mcmc_scatter` function creates a scatterplot with two parameters:

```
color_scheme_set("gray")
mcmc_scatter(posterior, pars = c("(Intercept)", "wt"), size = 1.5, alpha = 0.5)
```



The `mcmc_hex` function creates a similar plot but using hexagonal binning, which can be useful to avoid overplotting:

```
mcmc_hex(posterior, pars = c("(Intercept)", "wt"))
```

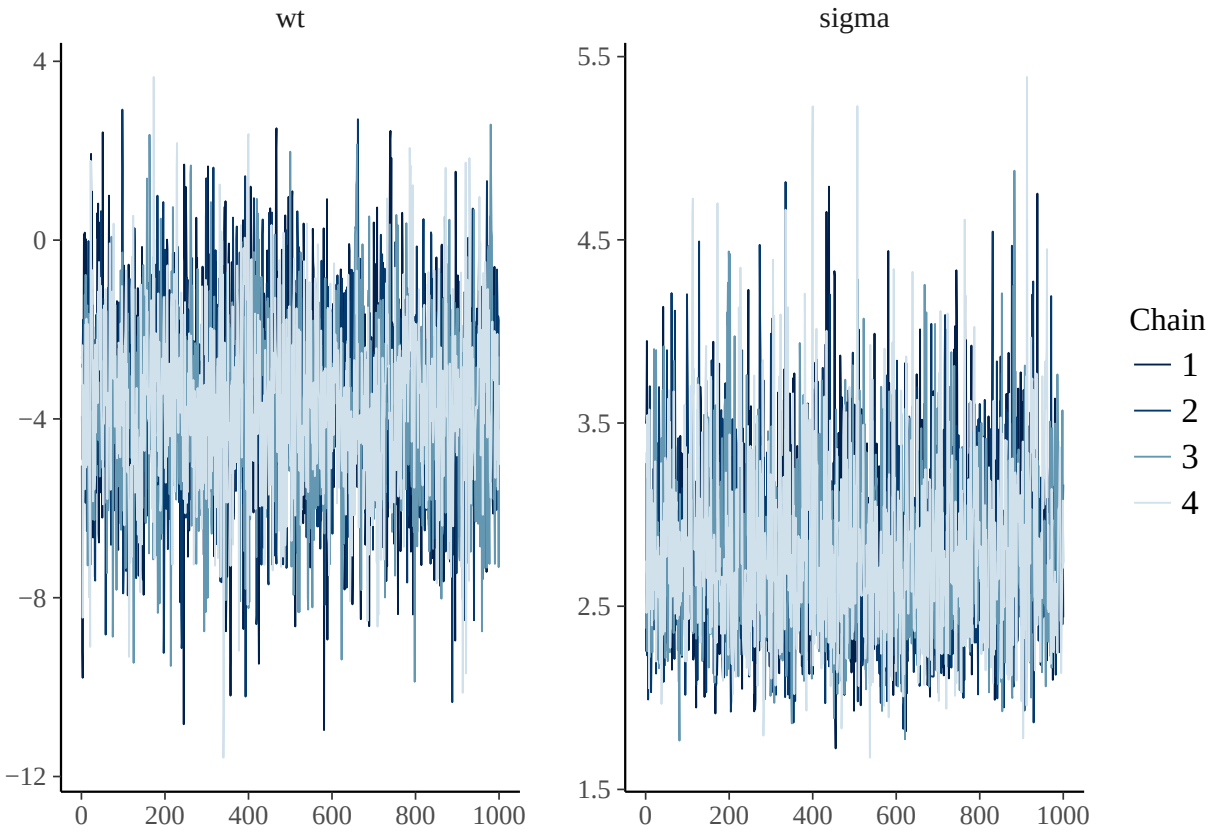


In addition to `mcmc_scatter` and `mcmc_hex`, as of **bayesplot** version 1.2.0 an `mcmc_pairs` function for creating pairs plots with more than two parameters is available. Examples will eventually be included in this vignette. For now see `help("mcmc_pairs")`.

2.1.4 Traceplots

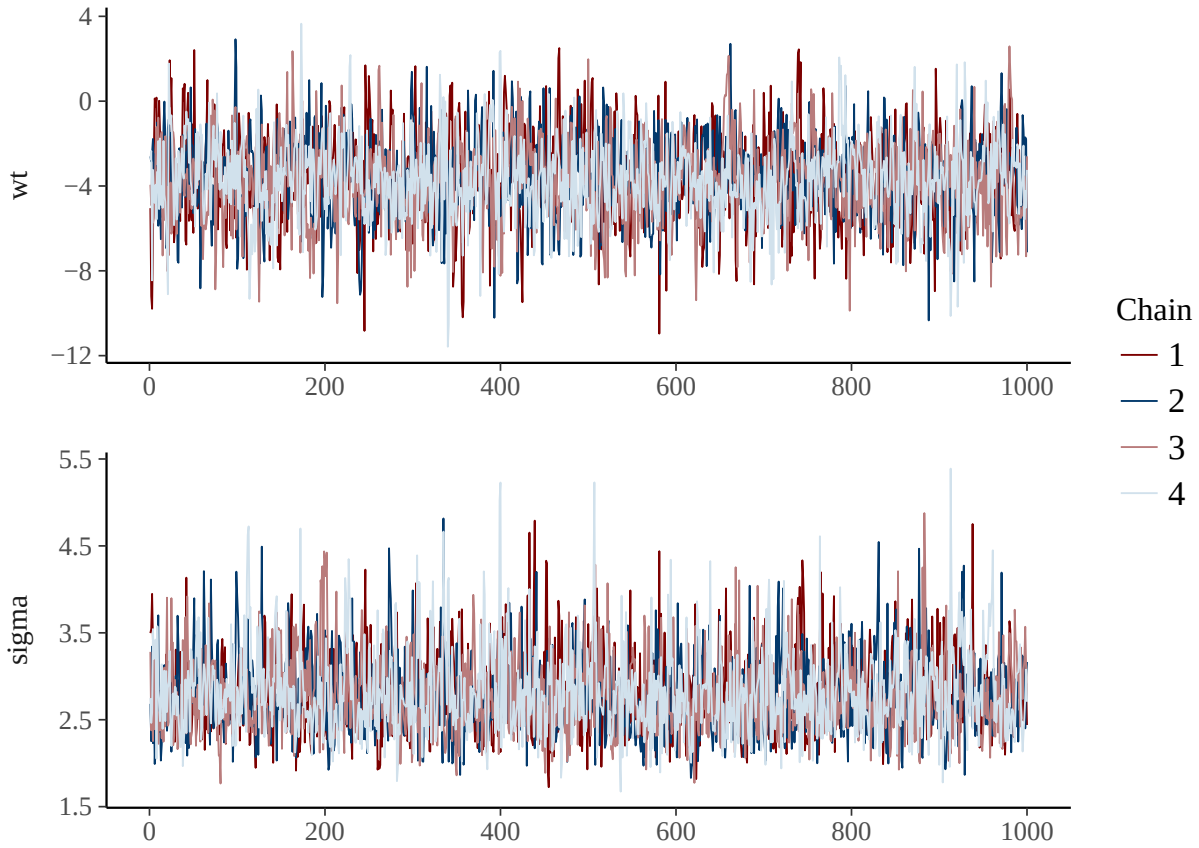
The `mcmc_trace` function creates standard traceplots:

```
color_scheme_set("blue")
mcmc_trace(posterior, pars = c("wt", "sigma"))
```



If it's hard to see the difference between the chains we can change to a mixed color scheme, for example:

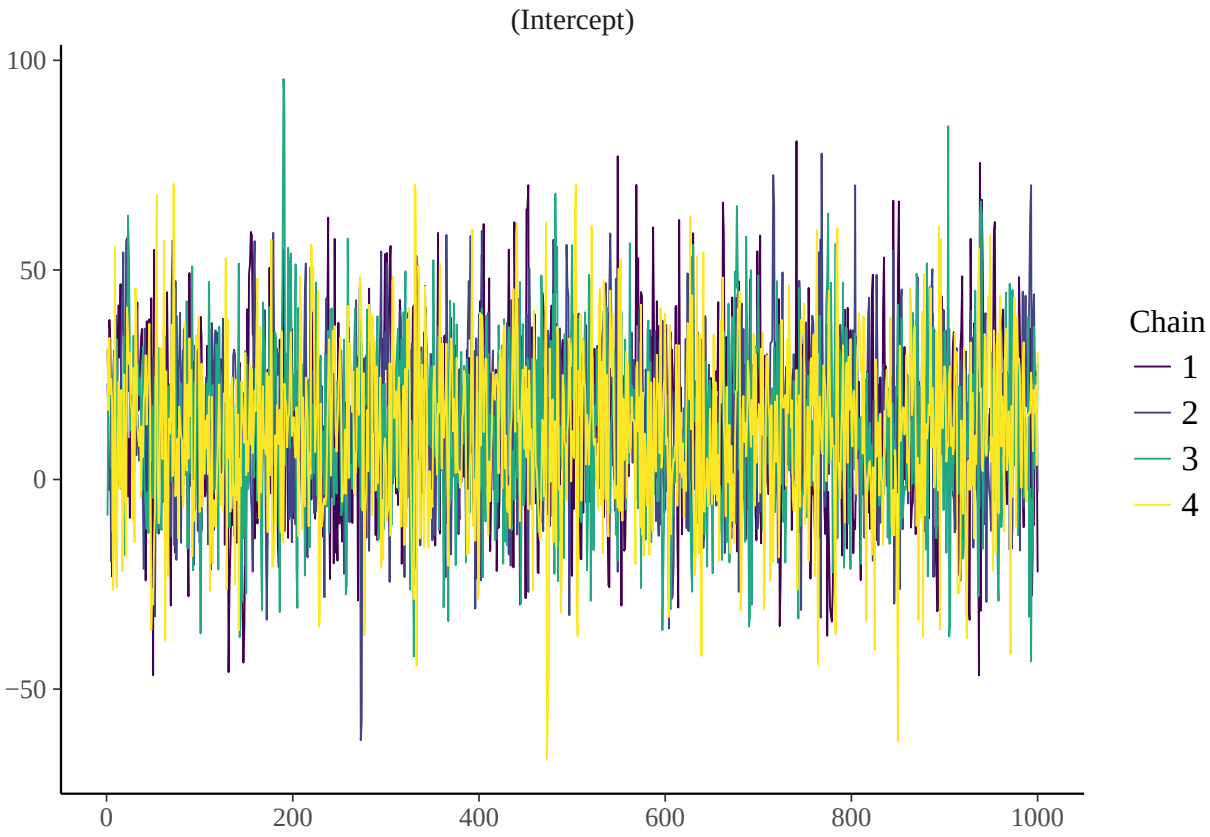
```
color_scheme_set("mix-blue-red")
mcmc_trace(posterior, pars = c("wt", "sigma"),
           facet_args = list(ncol = 1, strip.position = "left"))
```



The code above also illustrates the use of the `facet_args` argument, which is a list of parameters passed to `facet_wrap` in **ggplot2**. Specifying `ncol=1` means the traceplots will be stacked in a single column rather than placed side by side, and `strip.position="left"` moves the facet labels to the y-axis (instead of above each facet).

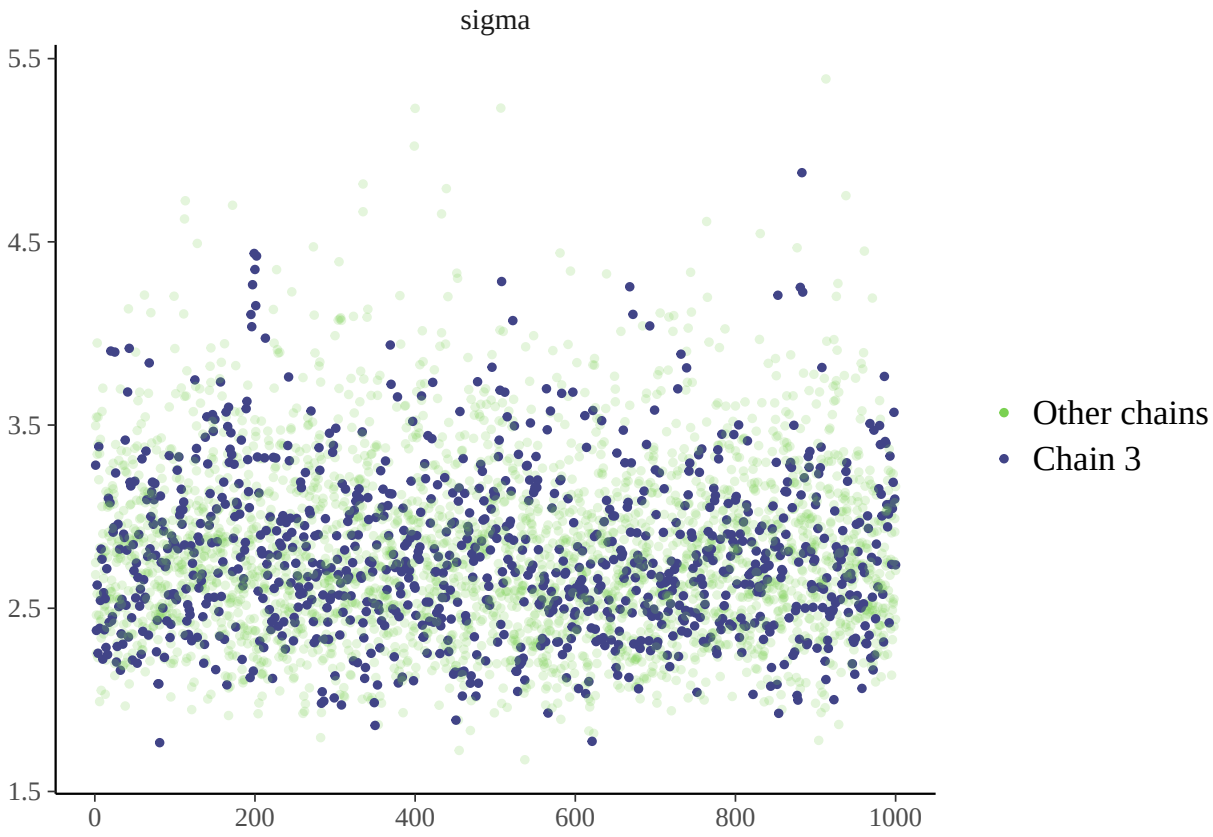
The `"viridis"` color scheme is also useful for traceplots because it is comprised of very distinct colors:

```
color_scheme_set("viridis")
mcmc_trace(posterior, pars = "(Intercept)")
```



The `mcmc_trace_highlight` function uses points instead of lines and reduces the opacity of all but a single chain (which is specified using the `highlight` argument).

```
mcmc_trace_highlight(posterior, pars = "sigma", highlight = 3)
```


sigma
5.5
4.5
3.5
2.5
1.5
0
200
400
600
800
1000
Other chains
Chain 3

Chapter 3

MCMC Diagnostics

This vignette focuses on MCMC diagnostic plots. Plots of parameter estimates from MCMC draws are covered in the separate vignette Plotting MCMC draws using the bayesplot package, and graphical posterior predictive checks are covered in Graphical posterior predictive checks using the bayesplot package.

In addition to **bayesplot** we'll load the following packages:

- **ggplot2** for customizing the ggplot objects created by **bayesplot**
- **rstan** for fitting the example models used throughout the vignette

```
library("bayesplot")
library("ggplot2")
library("rstan")
```

3.1 General MCMC diagnostics

A Markov chain generates draws from the target distribution only after it has converged to equilibrium. Unfortunately, this is only guaranteed in the limit in theory. In practice, diagnostics must be applied to monitor whether the Markov chain(s) have converged. The **bayesplot** package provides various plotting functions for visualizing Markov chain Monte Carlo (MCMC) diagnostics after fitting a Bayesian model. MCMC draws from any package can be used with **bayesplot**, although there are a few diagnostic plots that are specifically intended to be used for Stan models.

To demonstrate, in this vignette we'll use the eight schools example discussed in Rubin (1981), Gelman et al (2013), and the RStan Getting Started wiki. This is a simple hierarchical meta-analysis model with data consisting of point estimates `y` and standard errors `sigma` from analyses of test prep programs in `J=8` schools:

```
schools_dat <- list(
  J = 8,
  y = c(28,  8, -3,  7, -1,  1, 18, 12),
  sigma = c(15, 10, 16, 11,  9, 11, 10, 18)
)
```

The model is:

$$
\begin{aligned}
y_j &\sim \text{Normal}(\theta_j, \sigma_j), \quad j = 1, \ldots, J \\
\theta_j &\sim \text{Normal}(\mu, \tau), \quad j = 1, \ldots, J \\
\mu &\sim \text{Normal}(0, 10) \\
\tau &\sim \text{half} - \text{Cauchy}(0, 10),
\end{aligned}
$$

with the normal distribution parameterized by the mean and standard deviation, not the variance or precision. In Stan code:

```
// Saved in 'schools_mod1.stan'
data {
  int<lower=0> J;
  vector[J] y;
  vector<lower=0>[J] sigma;
}
parameters {
  real mu;
  real<lower=0> tau;
  vector[J] theta;
}
model {
  mu ~ normal(0, 10);
  tau ~ cauchy(0, 10);
  theta ~ normal(mu, tau);
  y ~ normal(theta, sigma);
}
```

This parameterization of the model is referred to as the centered parameterization (CP). We'll also fit the same statistical model but using the so-called non-centered parameterization (NCP), which replaces the vector θ with a vector η of a priori *i.i.d.* standard normal parameters and then contructs θ deterministically from η by scaling by τ and shifting by μ:

$$
\begin{aligned}
\theta_j &= \mu + \tau\,\eta_j, \quad j = 1, \ldots, J \\
\eta_j &\sim N(0, 1), \quad j = 1, \ldots, J.
\end{aligned}
$$

The Stan code for this model is:

```
// Saved in 'schools_mod2.stan'
data {
  int<lower=0> J;
  vector[J] y;
  vector<lower=0>[J] sigma;
}
parameters {
  real mu;
  real<lower=0> tau;
  vector[J] eta;
}
transformed parameters {
  vector[J] theta;
  theta = mu + tau * eta;
}
```

```
model {
  mu ~ normal(0, 10);
  tau ~ cauchy(0, 10);
  eta ~ normal(0, 1); // implies theta ~ normal(mu, tau)
  y ~ normal(theta, sigma);
}
```

The centered and non-centered are two parameterizations of the same statistical model, but they have very different practical implications for MCMC. Using the **bayesplot** diagnostic plots, we'll see that, for this data, the NCP is required in order to properly explore the posterior distribution.

To fit both models we first translate the Stan code to C++ and compile it using the `stan_model` function.

```
schools_mod1 <- stan_model("schools_mod1.stan")
schools_mod2 <- stan_model("schools_mod2.stan")
```

We then fit the model by calling Stan's MCMC algorithm using the `sampling` function,

```
fit1 <- sampling(schools_mod1, data = schools_dat)
```

```
## Warning: There were 113 divergent transitions after warmup. Increasing adapt_delta above 0.8 may hel
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
```

```
## Warning: There were 2 chains where the estimated Bayesian Fraction of Missing Information was low. S
## http://mc-stan.org/misc/warnings.html#bfmi-low
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
fit2 <- sampling(schools_mod2, data = schools_dat)
```

and extract a `iterations x chains x parameters` array of posterior draws with `as.array`,

```
# Extract posterior draws for later use
posterior1 <- as.array(fit1)
posterior2 <- as.array(fit2)
```

For now ignore any warnings about divergent transitions after warmup. We will come back to those later in the vignette in the Diagnostics for the No-U-Turn Sampler section.

3.1.1 Rhat: potential scale reduction statistic

> One way to monitor whether a chain has converged to the equilibrium distribution is to compare its behavior to other randomly initialized chains. This is the motivation for the Gelman and Rubin (1992) potential scale reduction statistic, $\hat{R}$. The $\hat{R}$ statistic measures the ratio of the average variance of samples within each chain to the variance of the pooled samples across chains; if all chains are at equilibrium, these will be the same and $\hat{R}$ will be one. If the chains have not converged to a common distribution, the $\hat{R}$ statistic will be greater than one. (Stan Development Team, 2016).

The **bayesplot** package provides the functions `mcmc_rhat` and `mcmc_rhat_hist` for visualizing $\hat{R}$ estimates.

First we'll quickly fit one of the models above again, this time intentionally using too few MCMC iterations. This should lead to some high $\hat{R}$ values.

```
fit1_50iter <- sampling(schools_mod1, data = schools_dat, chains = 2, iter = 50)
```

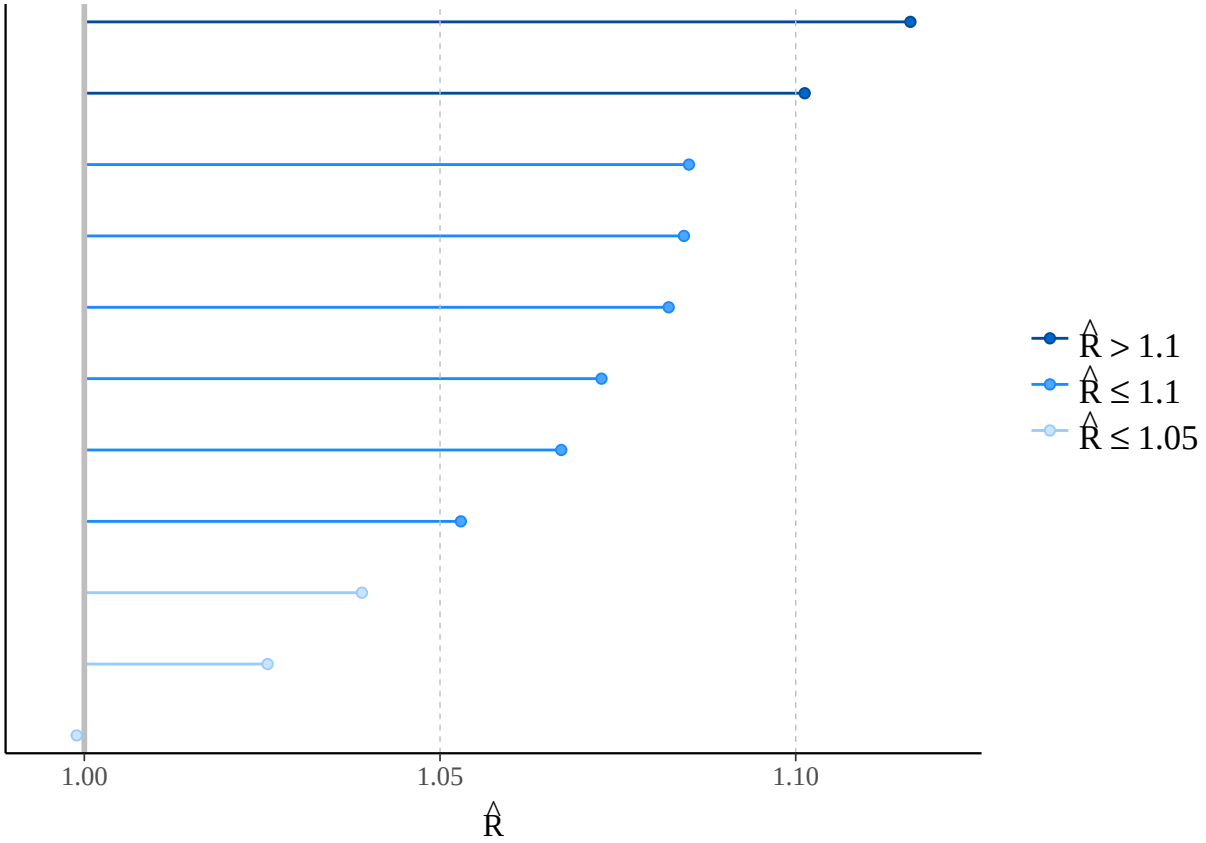
bayesplot also provides a generic `rhat` extractor function, currently with methods defined for models fit using the **rstan** and **rstanarm** packages. But regardless of how you fit your model, all **bayesplot** needs is a vector of $\hat{R}$ values.

```
library("bayesplot")
rhats <- rhat(fit1_50iter)
print(rhats)
```

```
##        mu       tau  theta[1]  theta[2]  theta[3]  theta[4]  theta[5]
## 1.1161176 1.0821440 1.0390174 1.0726987 1.0257599 1.0529287 1.0670428
##  theta[6]  theta[7]  theta[8]      lp__
## 0.9989288 1.1012643 1.0849883 1.0842890
```

We can visualize the $\hat{R}$ values with the `mcmc_rhat` function:

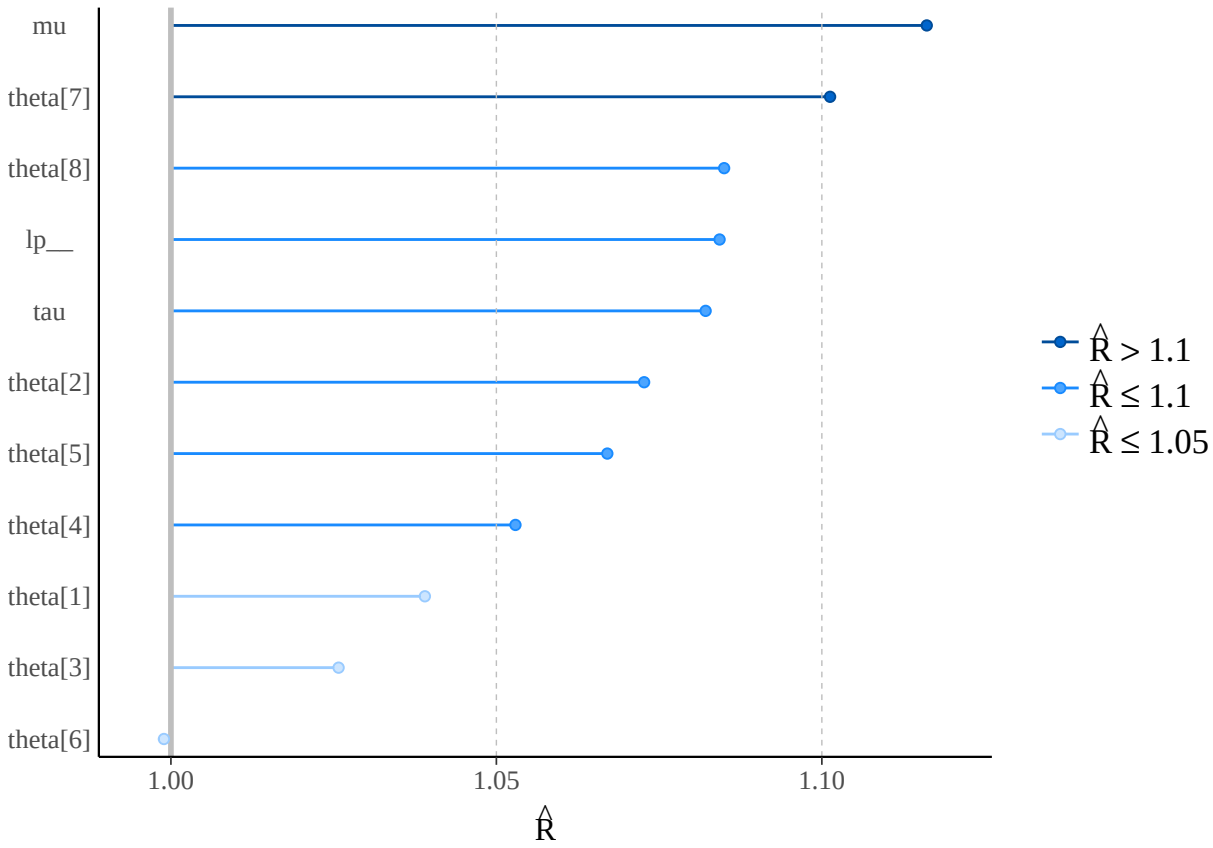
```
color_scheme_set("brightblue") # see help("color_scheme_set")
mcmc_rhat(rhats)
```



In the plot, the points representing the $\hat{R}$ values are colored based on whether they are less than 1.05, between 1.05 and 1.1, or greater than 1.1.

We can see the names of the parameters with the concerning $\hat{R}$ values by turning on the y-axis text using the `yaxis_text` convenience function:

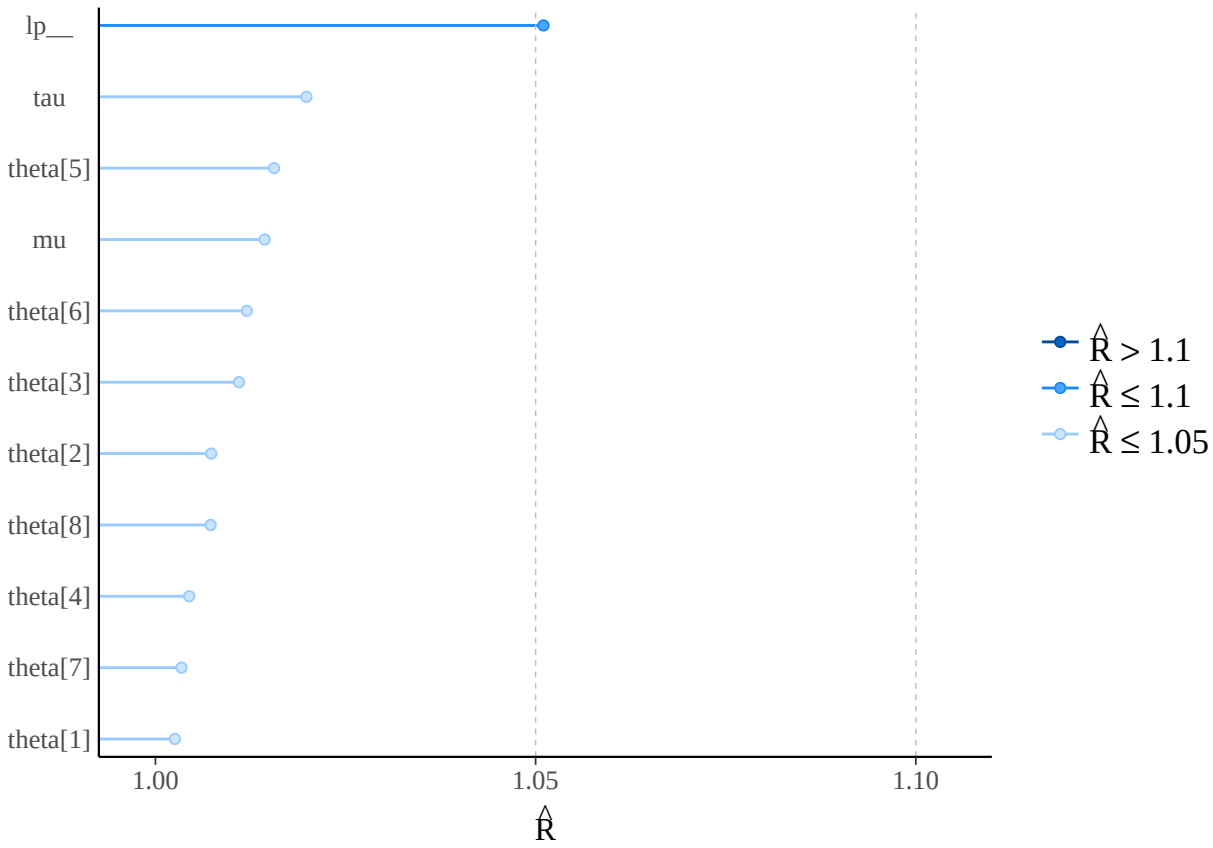
```
mcmc_rhat(rhats) + yaxis_text()
```



The axis y-axis text is off by default for this plot because it's only possible to see the labels clearly for models with very few parameters.

If we look at the same model fit using longer Markov chains we should see all $\hat{R} < 1.1$, and all points in the plot the same (light) color:

```
mcmc_rhat(rhat = rhat(fit1)) + yaxis_text()
```



We can see the same information shown by `mcmc_rhat` but in histogram form using the `mcmc_rhat_hist` function. See the **Examples** section in `help("mcmc_rhat_hist")` for examples.

3.1.2 Effective sample size

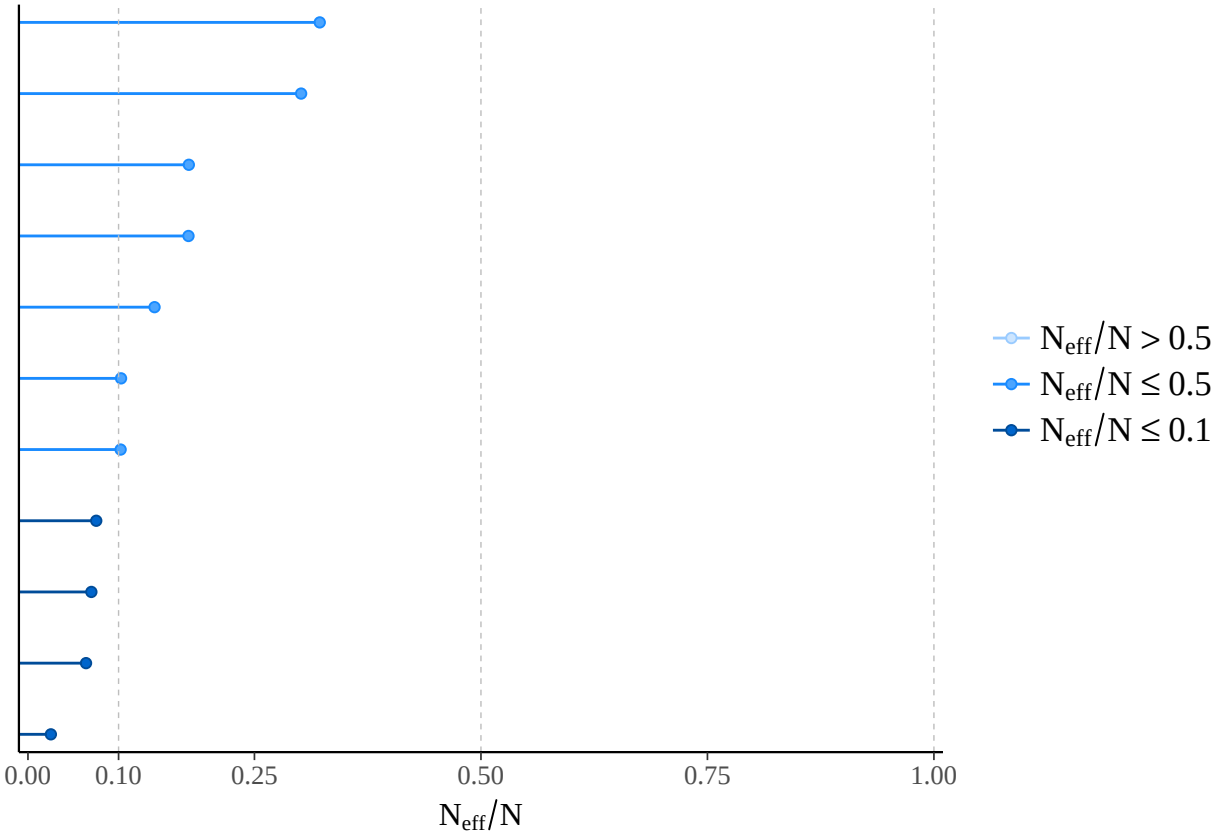
The effective sample size is an estimate of the number of independent draws from the posterior distribution of the estimand of interest. Because the draws within a Markov chain are *not* independent if there is autocorrelation, the effective sample size, n_{eff}, will be smaller than the total sample size, N. The larger the ratio of n_{eff} to N the better.

The **bayesplot** package provides a generic `neff_ratio` extractor function, currently with methods defined for models fit using the **rstan** and **rstanarm** packages. But regardless of how you fit your model, all **bayesplot** needs is a vector of n_{eff}/N values. The `mcmc_neff` and `mcmc_neff_hist` can then be used to plot the ratios.

```
ratios1 <- neff_ratio(fit1)
print(ratios1)
```

```
##         mu        tau   theta[1]   theta[2]   theta[3]   theta[4]
## 0.06997981 0.06409421 0.32206367 0.13971662 0.10274815 0.17756532
##   theta[5]   theta[6]   theta[7]   theta[8]       lp__
## 0.07536029 0.10225037 0.30151712 0.17718197 0.02528420
```

```
mcmc_neff(ratios1)
```

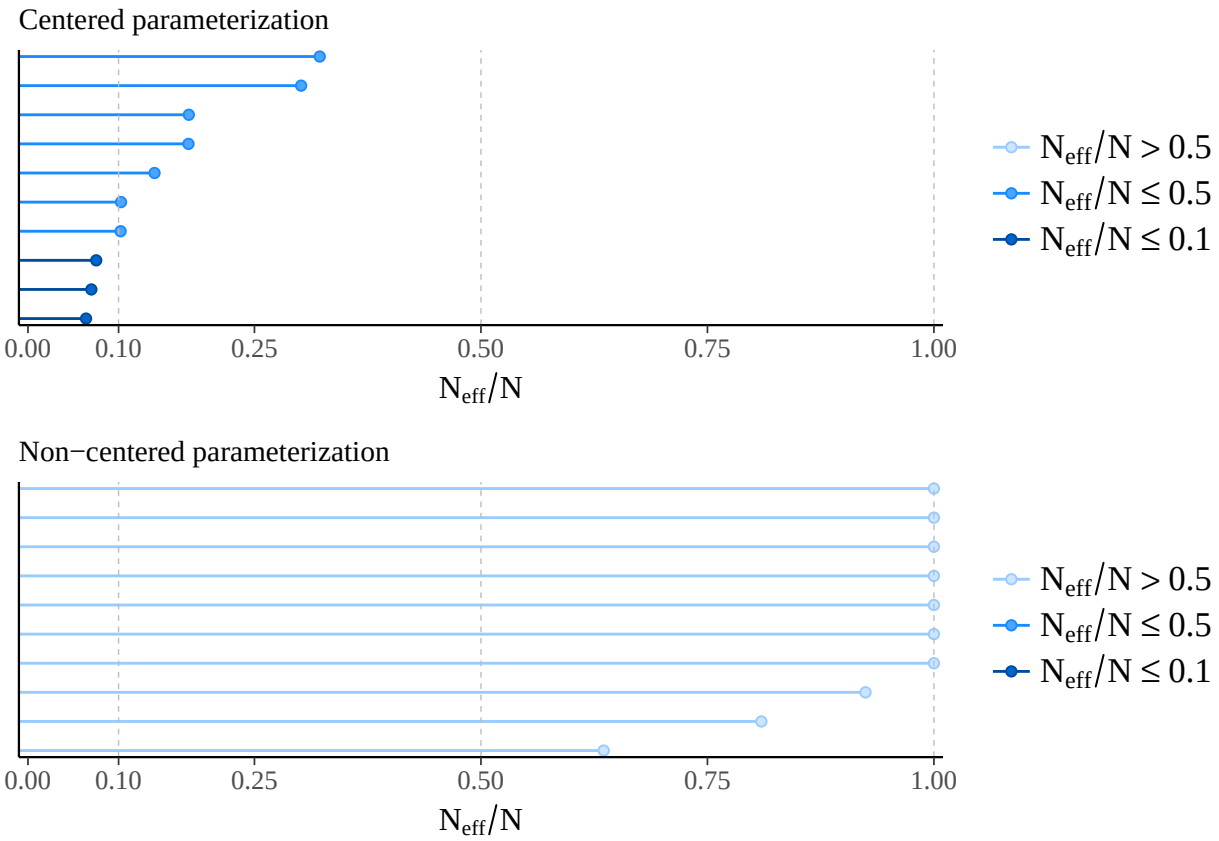


In the plot, the points representing the values of n_{eff}/N are colored based on whether they are less than 0.1, between 0.1 and 0.5, or greater than 0.5. These particular values are arbitrary in that they have no particular theoretical meaning, but a useful heuristic is to worry about any n_{eff}/N less than 0.1.

One important thing to keep in mind is that these ratios will depend not only on the model being fit but also on the particular MCMC algorithm used. One reason why we have such high ratios of n_{eff} to N is that the No-U-Turn sampler used by **rstan** generally produces draws from the posterior distribution with much lower autocorrelations compared to draws obtained using other MCMC algorithms (e.g., Gibbs).

Even for models fit using **rstan** the parameterization can make a big difference. Here are the n_{eff}/N plots for `fit1` and `fit2` side by side.

```
# A function we'll use several times to plot comparisons of the centered
# parameterization (cp) and the non-centered parameterization (ncp). See
# help("bayesplot_grid") for details on the bayesplot_grid function used here.
compare_cp_ncp <- function(cp_plot, ncp_plot, ncol = 1) {
  txt <- c("Centered parameterization", "Non-centered parameterization")
  bayesplot_grid(cp_plot, ncp_plot, subtitles = txt,
                 grid_args = list(ncol = ncol))
}

neff1 <- neff_ratio(fit1, pars = c("theta", "mu", "tau"))
neff2 <- neff_ratio(fit2, pars = c("theta", "mu", "tau"))
compare_cp_ncp(mcmc_neff(neff1), mcmc_neff(neff2))
```



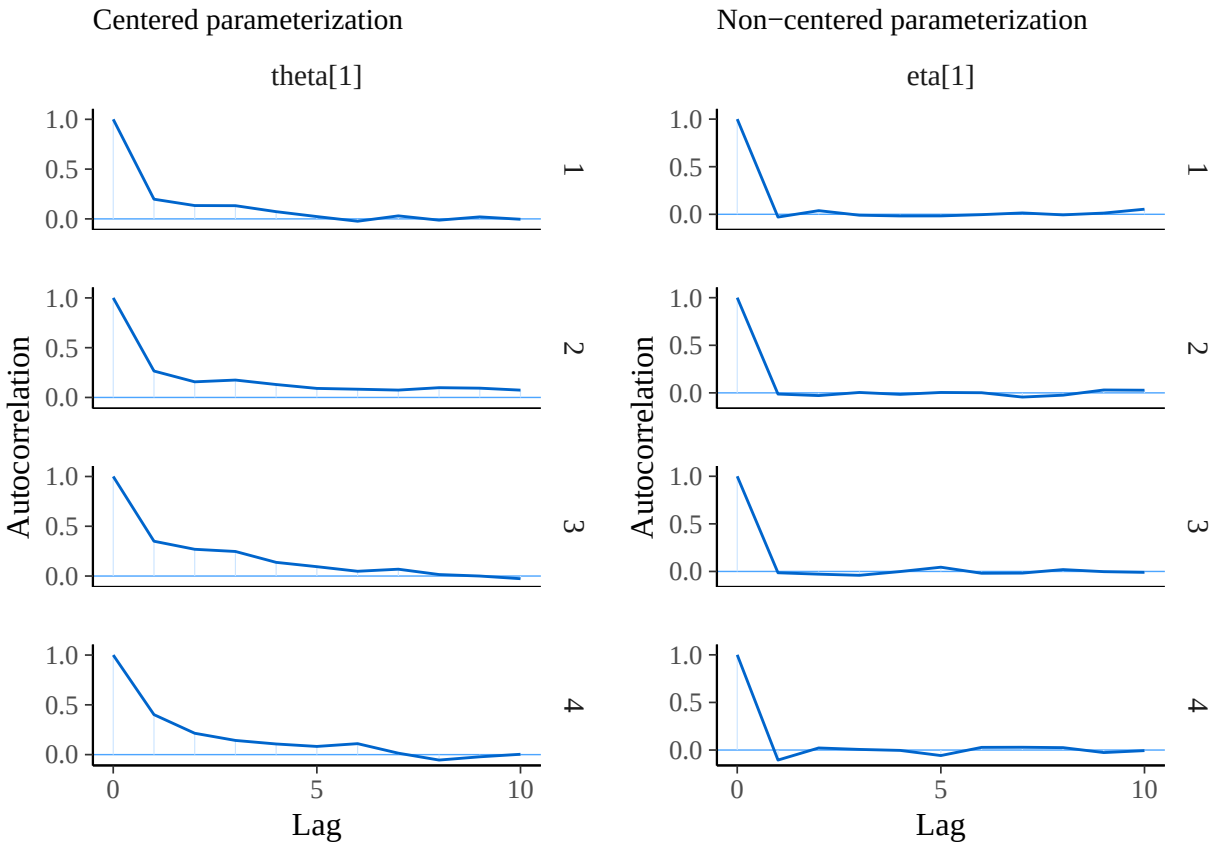
Because of the difference in parameterization, the effective sample sizes are much better for the second model, the non-centered parameterization.

3.1.3 Autocorrelation

As mentioned above, n_{eff}/N decreases as autocorrelation becomes more extreme. We can visualize the autocorrelation using the `mcmc_acf` (line plot) or `mcmc_acf_bar` (bar plot) functions. For the selected parameters, these functions show the autocorrelation for each Markov chain separately up to a user-specified number of lags.

Here we can again see a difference when comparing the two parameterizations of the same model. For model 1, θ_1 is the primitive parameter for school 1, whereas for the non-centered parameterization in model 2 the primitive parameter is η_1 (and θ_1 is later constructed from η_1, μ, and τ):

```
compare_cp_ncp(
  mcmc_acf(posterior1, pars = "theta[1]", lags = 10),
  mcmc_acf(posterior2, pars = "eta[1]", lags = 10),
  ncol = 2
)
```



3.2 Diagnostics for the No-U-Turn Sampler

The No-U-Turn Sampler (NUTS, Hoffman and Gelman, 2014) is the variant of Hamiltonian Monte Carlo (HMC) used by Stan and the various R packages that depend on Stan for fitting Bayesian models.

The **bayesplot** package has special functions for visualizing some of the unique diagnostics permitted by HMC, and NUTS in particular. See Betancourt (2017), Betancourt and Girolami (2013), and Stan Development Team (2016) for more details on the concepts.

The special **bayesplot** functions for NUTS diagnostics are

```
available_mcmc(pattern = "_nuts_")
```

```
## bayesplot MCMC module:
## (matching pattern '_nuts_')
##   mcmc_nuts_acceptance
##   mcmc_nuts_divergence
##   mcmc_nuts_energy
##   mcmc_nuts_stepsize
##   mcmc_nuts_treedepth
```

The **bayesplot** package also provides generic functions `log_posterior` and `nuts_params` for extracting the required information for the plots from fitted model objects. Currently methods are provided for models fit using the **rstan** and **rstanarm** packages, although it is not difficult to define additional methods for the objects returned by other R packages.

For the Stan models we fit above we can use the `log_posterior` and `nuts_params` methods for stanfit objects:

```
lp1 <- log_posterior(fit1)
head(lp1)
```

```
##   Iteration     Value Chain
## 1         1 -8.382750     1
## 2         2 -6.188724     1
## 3         3 -9.237994     1
## 4         4 -5.509421     1
## 5         5 -5.111442     1
## 6         6 -5.111442     1
```

```
np1 <- nuts_params(fit1)
head(np1)
```

```
##   Iteration     Parameter        Value Chain
## 1         1 accept_stat__ 0.1838466550     1
## 2         2 accept_stat__ 0.9549609100     1
## 3         3 accept_stat__ 0.3343947756     1
## 4         4 accept_stat__ 0.2857634106     1
## 5         5 accept_stat__ 0.5714804380     1
## 6         6 accept_stat__ 0.0001527055     1
```

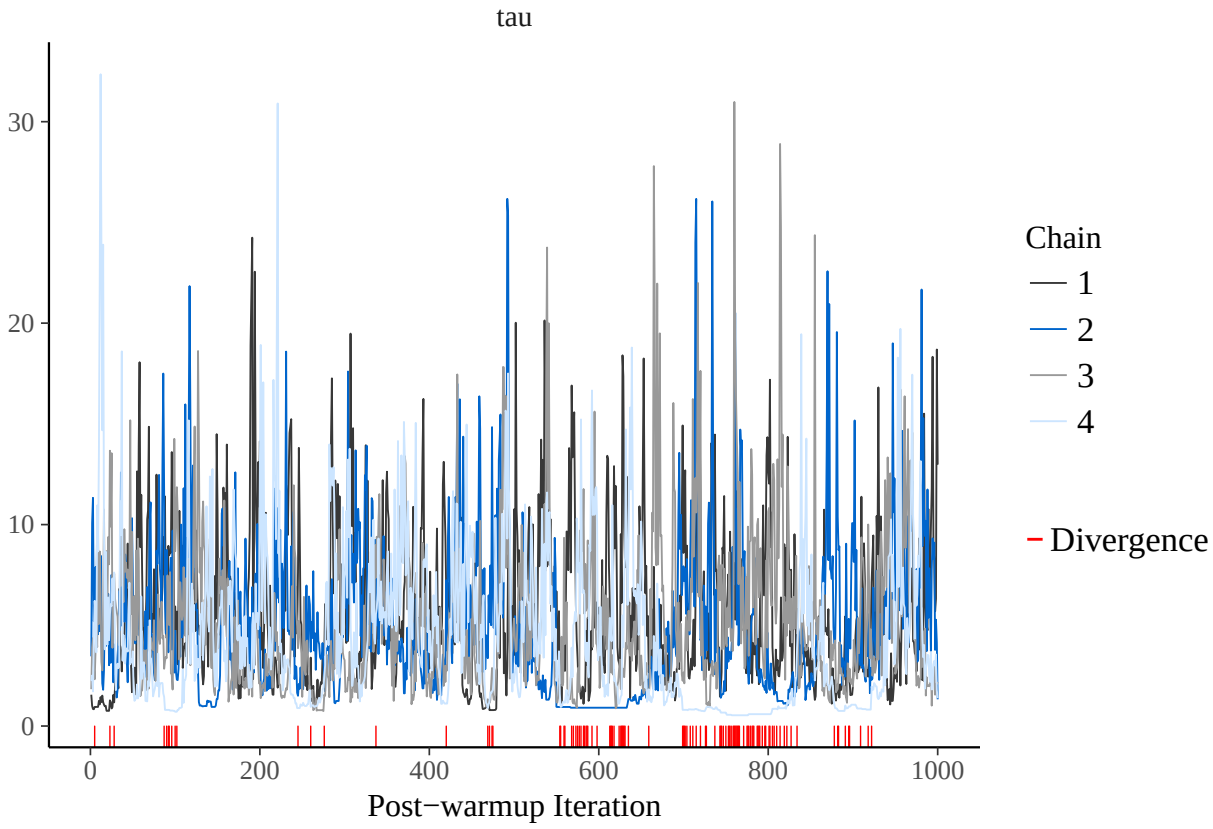
```
# for the second model
lp2 <- log_posterior(fit2)
np2 <- nuts_params(fit2)
```

3.2.1 Divergent transitions

When running the Stan models at the beginning of this vignette there were warnings about divergent transitions. For an explanation of these warnings see Divergent transitions after warmup.

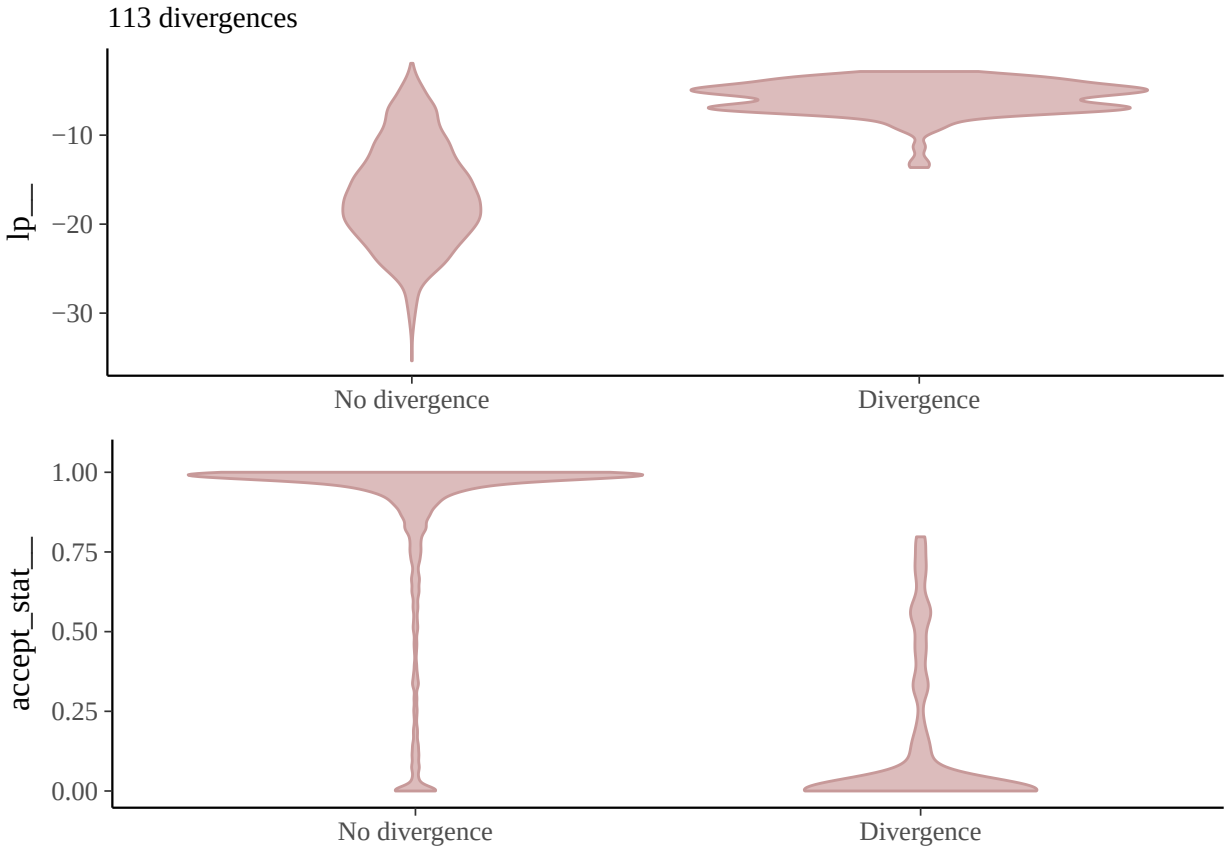
The `divergences` argument to the `mcmc_trace` function can be used to add a rug plot (in red) of the divergences to a trace plot of parameter draws. Typically we can see that at least one of the chains is getting stuck wherever there is a cluster of many red marks:

```
color_scheme_set("mix-brightblue-gray")
mcmc_trace(posterior1, pars = "tau", divergences = np1) +
  xlab("Post-warmup Iteration")
```



To look deeper at the information conveyed by the divergences we can use the `mcmc_nuts_divergence` function:

```
color_scheme_set("red")
mcmc_nuts_divergence(np1, lp1)
```

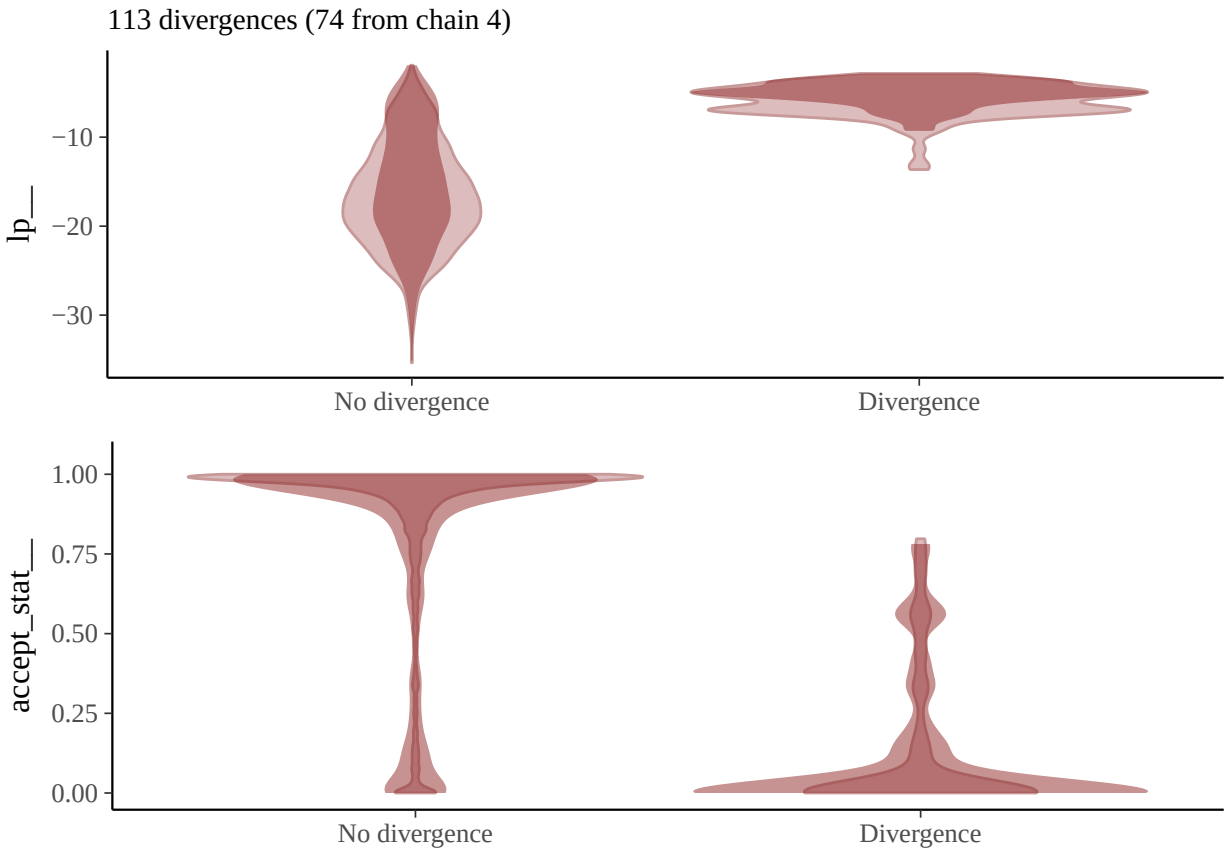


In the top panel we see the distribution of the log-posterior when there was no divergence vs the distribution when there was a divergence. Divergences often indicate that some part of the posterior isn't being explored and `lp|Divergence` will have lighter tails than `lp|No divergence`.

The bottom panel shows the same thing but instead of the log-posterior the NUTS acceptance statistic is shown.

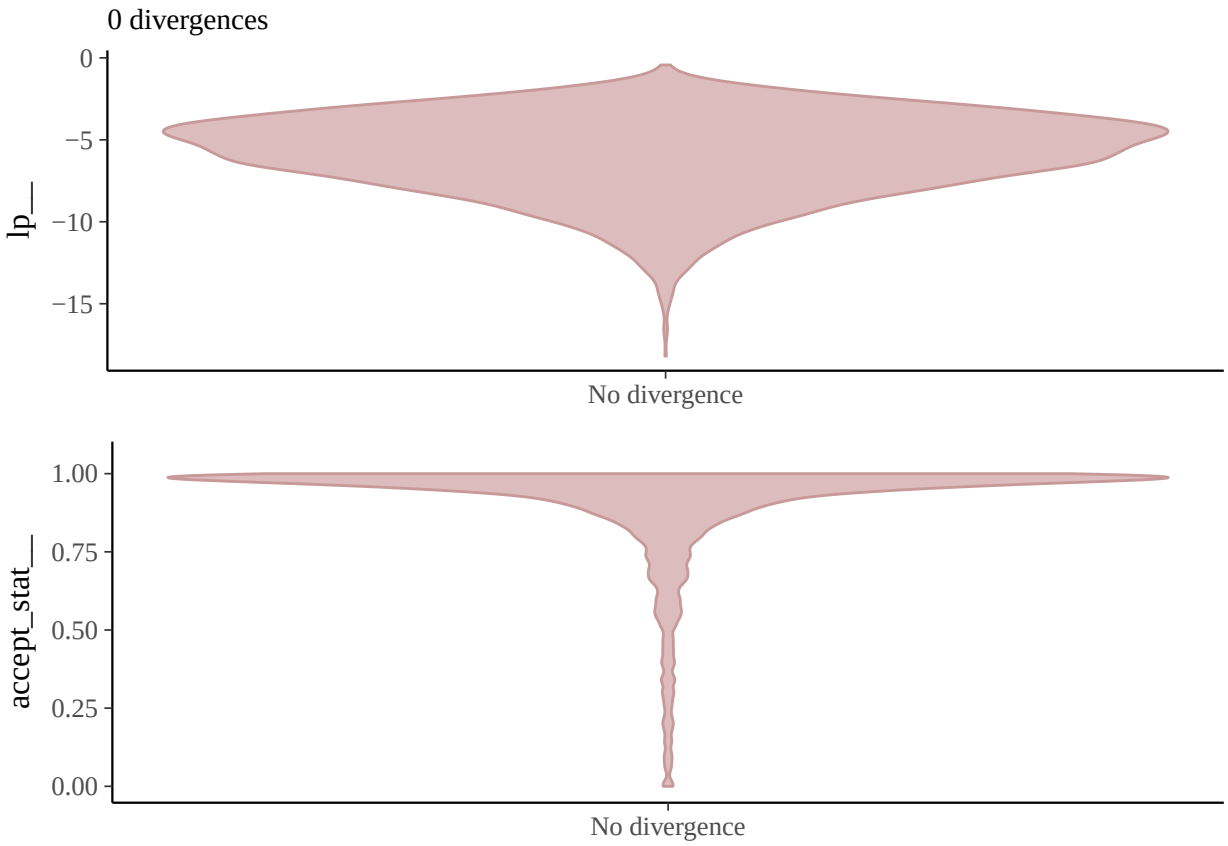
Specifying the optional `chain` argument will overlay the plot just for a particular Markov chain on the plot for all chains combined:

```
mcmc_nuts_divergence(np1, lp1, chain = 4)
```



For the non-centered parameterization we also had warnings about divergences

```
mcmc_nuts_divergence(np2, lp2)
```



but we have far fewer of them to worry about. If there are only a few divergences we can often get rid of them by increasing the target acceptance rate (`adapt_delta`), which has the effect of lowering the stepsize used by the sampler and allowing the Markov chains to explore more complicated curvature in the target distribution.

```
fit1b <- sampling(schools_mod1, data = schools_dat,
                  control = list(adapt_delta = 0.99))
```

```
## Warning: There were 9 divergent transitions after warmup. Increasing adapt_delta above 0.99 may help
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
```

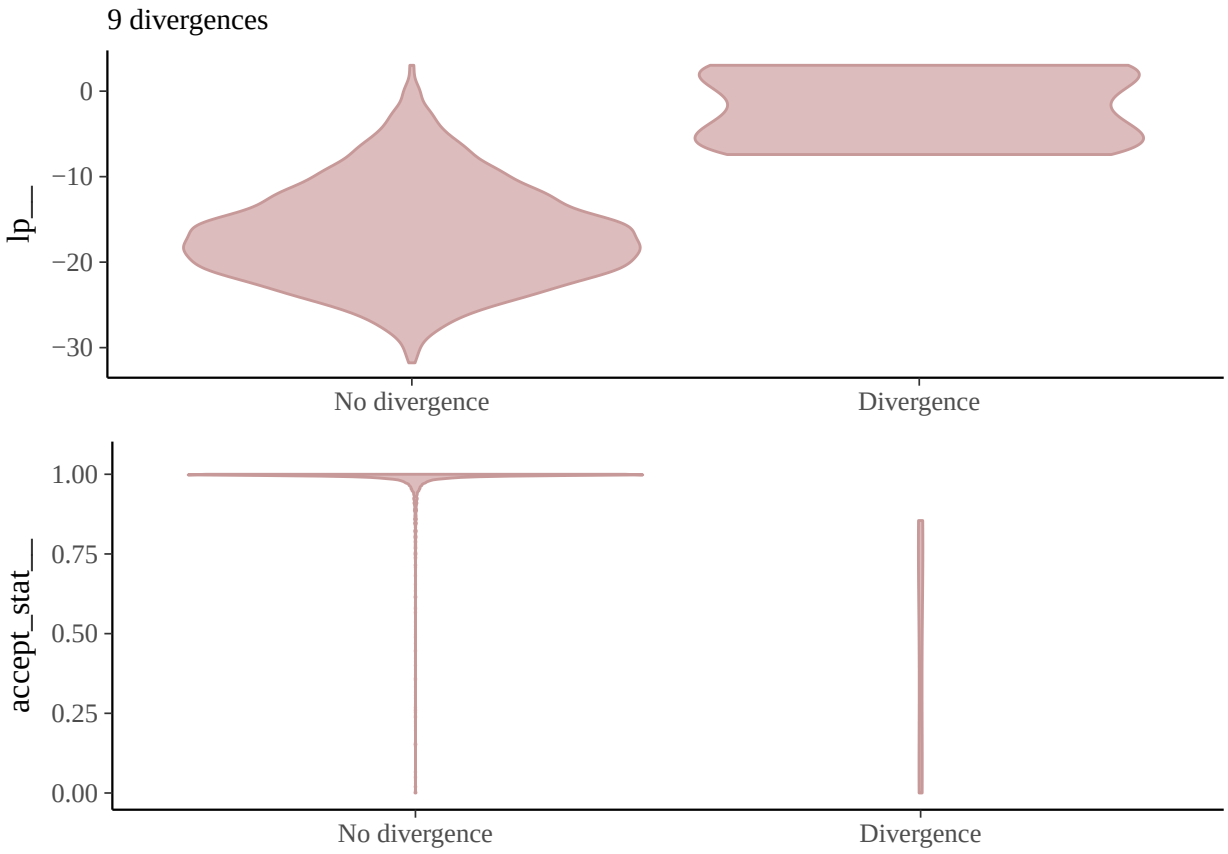
```
## Warning: There were 4 chains where the estimated Bayesian Fraction of Missing Information was low. S
## http://mc-stan.org/misc/warnings.html#bfmi-low
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

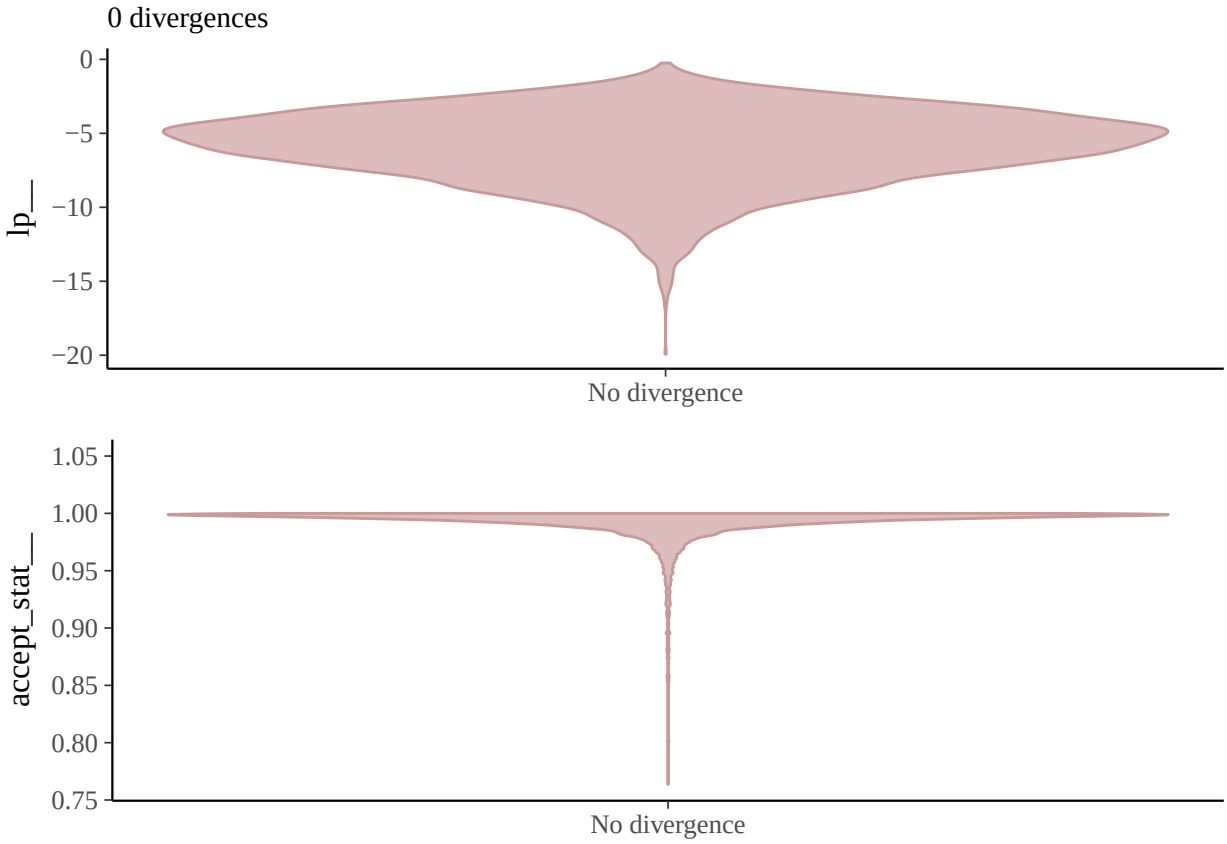
```
fit2b <- sampling(schools_mod2, data = schools_dat,
                  control = list(adapt_delta = 0.99))
```

For the first model and this particular data, increasing `adapt_delta` will not solve the problem and a reparameterization is required.

```
mcmc_nuts_divergence(nuts_params(fit1b), log_posterior(fit1b))
```




```
mcmc_nuts_divergence(nuts_params(fit2b), log_posterior(fit2b))
```

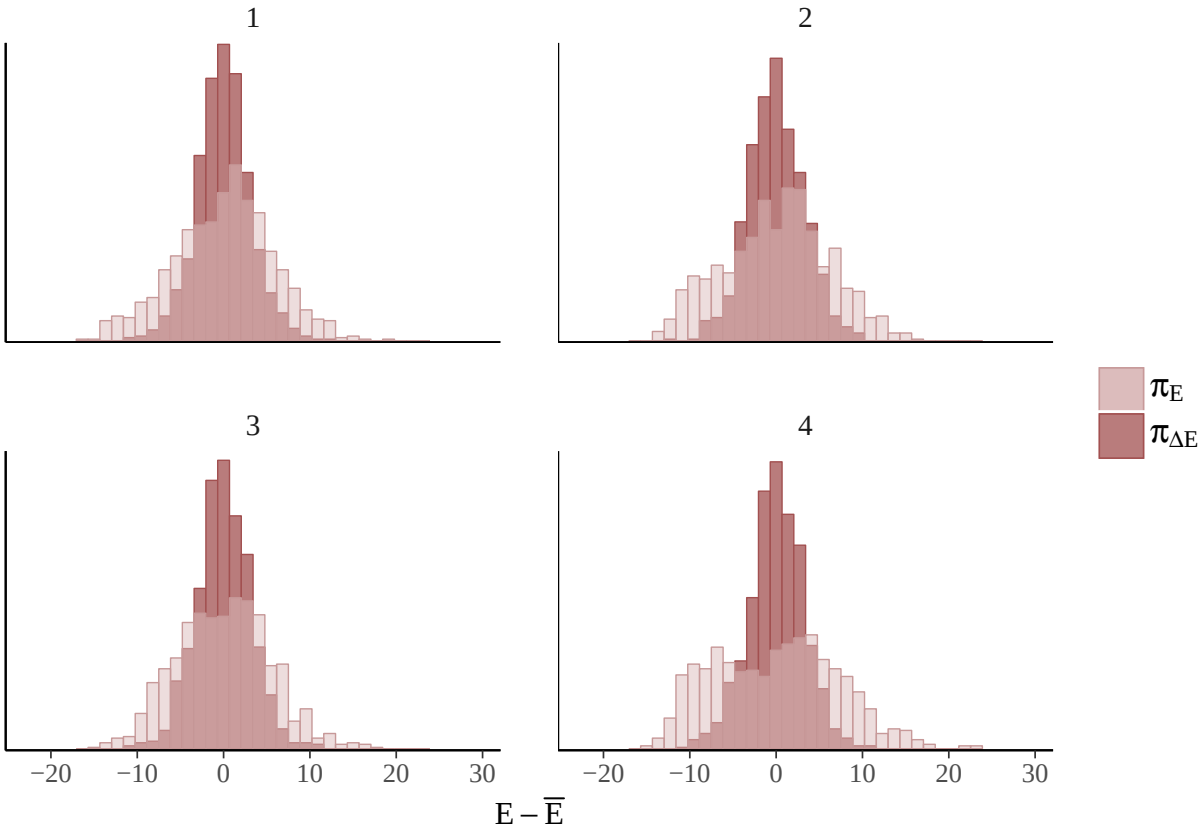


3.2.2 Energy and Bayesian fraction of missing information

The `mcmc_nuts_energy` function creates plots similar to those presented in Betancourt (2017). While `mcmcm_nuts_divergence` can identify light tails and incomplete exploration of the target distribution, the `mcmc_nuts_energy` function can identify overly heavy tails that are also challenging for sampling. Informally, the energy diagnostic for HMC (and the related energy-based Bayesian fraction of missing information) quantifies the heaviness of the tails of the posterior distribution.

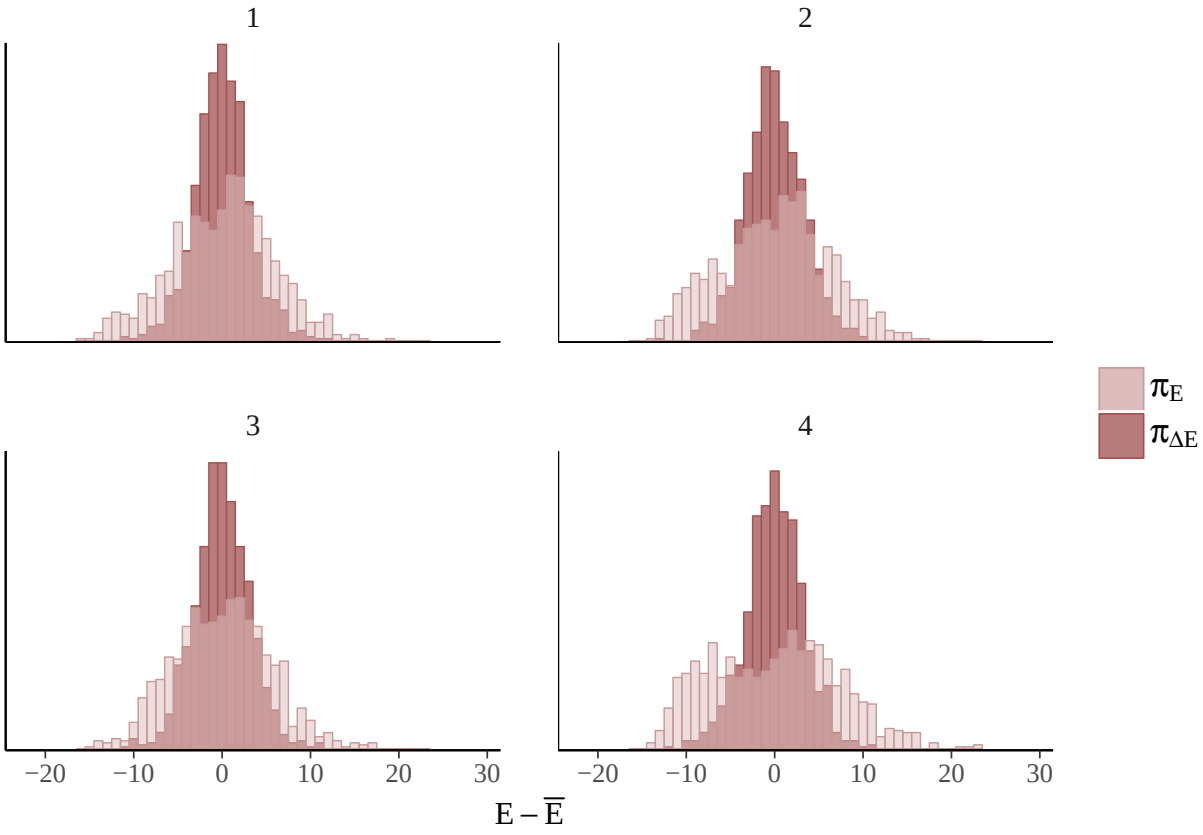
The plot created by `mcmc_nuts_energy` shows overlaid histograms of the (centered) marginal energy distribution π_E and the first-differenced distribution $\pi_{\Delta E}$,

```
mcmc_nuts_energy(np1)
```



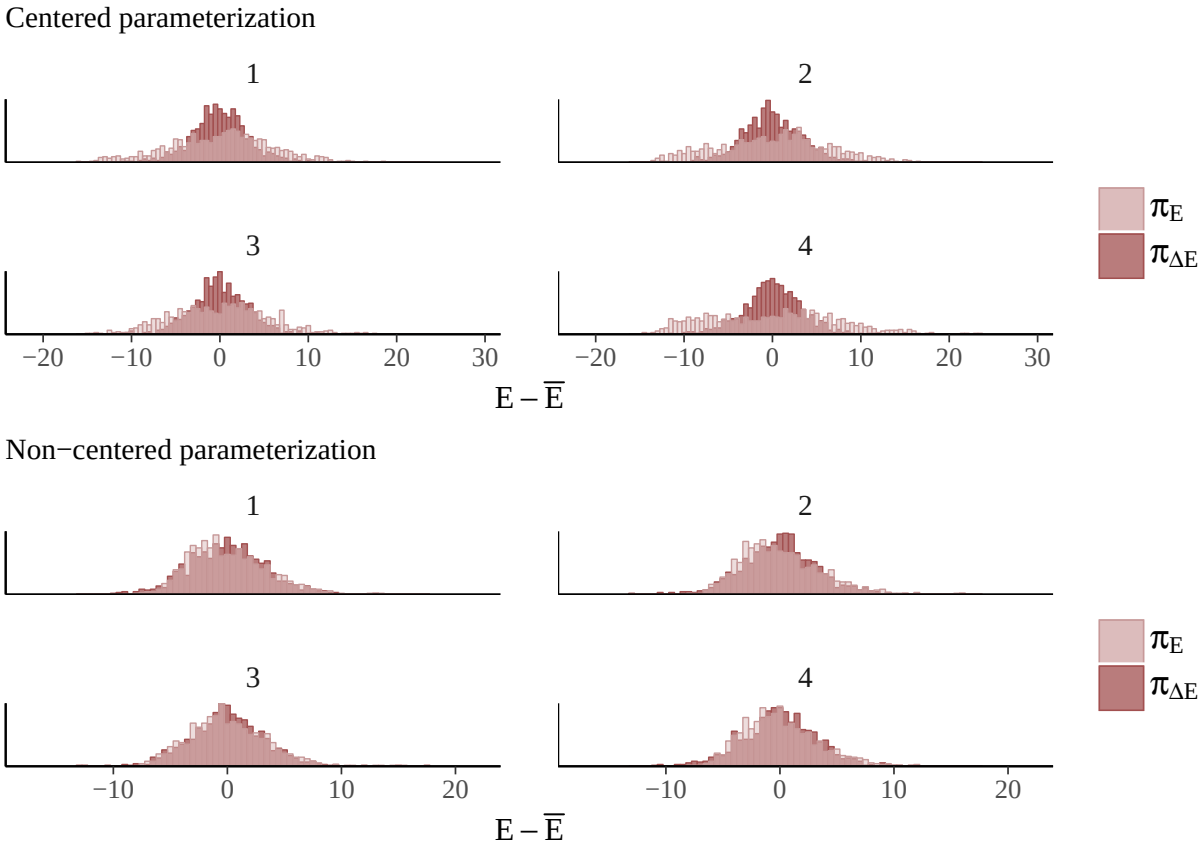
To keep the chains separate add `merge_chains=FALSE`:

```
mcmc_nuts_energy(np1, merge_chains = FALSE, binwidth = 1)
```



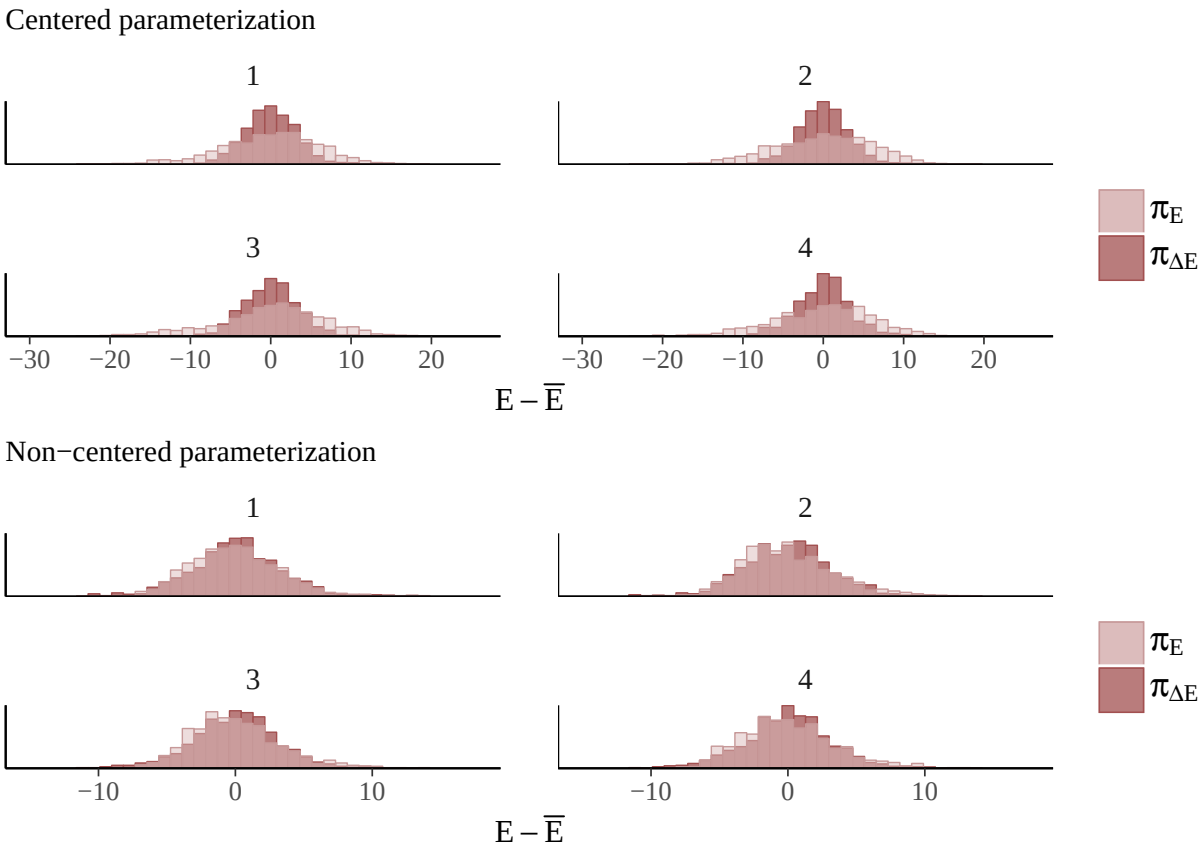
The two histograms ideally look the same (Betancourt, 2017), which is only the case for the non-centered parameterization (bottom panel):

```
compare_cp_ncp(
  mcmc_nuts_energy(np1, binwidth = 1/2),
  mcmc_nuts_energy(np2, binwidth = 1/2)
)
```



The difference between the parameterizations is even more apparent if we force the stepsize to a smaller value and help the chains explore more of the posterior:

```
np1b <- nuts_params(fit1b)
np2b <- nuts_params(fit2b)
compare_cp_ncp(
  mcmc_nuts_energy(np1b),
  mcmc_nuts_energy(np2b)
)
```



See Betancourt (2017) for more on this particular example as well as the general theory behind the energy plots.

Chapter 4

References

Betancourt, M. (2017). A conceptual introduction to Hamiltonian Monte Carlo. https://arxiv.org/abs/1701.02434

Betancourt, M. (2016). Diagnosing suboptimal cotangent disintegrations in Hamiltonian Monte Carlo. https://arxiv.org/abs/1604.00695

Betancourt, M. and Girolami, M. (2013). Hamiltonian Monte Carlo for hierarchical models. https://arxiv.org/abs/1312.0906

Buerkner, P. (2018). brms: Bayesian Regression Models using Stan. R package version 1.4.0. https://CRAN.R-project.org/package=brms

Gabry, J., and Goodrich, B. (2017). rstanarm: Bayesian Applied Regression Modeling via Stan. R package version 2.14.1. http://mc-stan.org/interfaces/rstanarm.html, https://CRAN.R-project.org/package=rstanarm

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2013). *Bayesian Data Analysis*. Chapman & Hall/CRC Press, London, third edition.

Gelman, A. and Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences. *Statistical Science*. 7(4): 457–472.

Hoffman, M. D. and Gelman, A. (2014). The No-U-Turn Sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*. 15:1593–1623.

Rubin, D. B. (1981). Estimation in Parallel Randomized Experiments. *Journal of Educational and Behavioral Statistics*. 6:377–401.

Stan Development Team. (2016). *Stan Modeling Language Users Guide and Reference Manual*. http://mc-stan.org/documentation/