

# Package ‘bayesplot’

September 12, 2017

**Type** Package

**Title** Plotting for Bayesian Models

**Version** 1.4.0

**Date** 2017-09-09

**Maintainer** Jonah Gabry <jsg2201@columbia.edu>

**Description** Plotting functions for posterior analysis, model checking, and MCMC diagnostics. The package is designed 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 R packages for Bayesian modeling, particularly (but not exclusively) packages interfacing with 'Stan'.

**License** GPL (>= 3)

**LazyData** TRUE

**URL** <http://mc-stan.org/bayesplot>

**BugReports** <https://github.com/stan-dev/bayesplot/issues/>

**SystemRequirements** pandoc

**Depends** R (>= 3.1.0)

**Imports** dplyr (>= 0.7.1), ggplot2 (>= 2.2.1), reshape2, stats, utils, rlang

**Suggests** arm, gridExtra (>= 2.2.1), knitr (>= 1.16), loo (>= 1.1.0), rmarkdown (>= 1.0.0), rstan (>= 2.14.1), rstanarm (>= 2.14.1), rstantools (>= 1.2.0), scales, shinystan (>= 2.3.0), testthat

**RoxygenNote** 6.0.1

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Jonah Gabry [aut, cre],  
Tristan Mahr [aut],  
Paul-Christian Buerkner [ctb]

**Repository** CRAN

**Date/Publication** 2017-09-12 15:08:52 UTC

## R topics documented:

bayesplot-package . . . . .	2
available_ppc . . . . .	4
bayesplot-colors . . . . .	5
bayesplot-extractors . . . . .	7
bayesplot-helpers . . . . .	9
bayesplot_grid . . . . .	14
MCMC-combos . . . . .	16
MCMC-diagnostics . . . . .	17
MCMC-distributions . . . . .	20
MCMC-intervals . . . . .	23
MCMC-nuts . . . . .	26
MCMC-overview . . . . .	29
MCMC-parcoord . . . . .	30
MCMC-recover . . . . .	33
MCMC-scatterplots . . . . .	35
MCMC-traces . . . . .	41
PPC-discrete . . . . .	45
PPC-distributions . . . . .	47
PPC-errors . . . . .	50
PPC-intervals . . . . .	53
PPC-loo . . . . .	55
PPC-overview . . . . .	58
PPC-scatterplots . . . . .	60
PPC-test-statistics . . . . .	61
pp_check . . . . .	63
theme_default . . . . .	65
<b>Index</b>	<b>67</b>

---

bayesplot-package      *Plots for Bayesian Models*

---

## Description

The **bayesplot** package provides a variety of **ggplot2**-based plotting functions for use after fitting Bayesian models (typically, though not exclusively, via Markov chain Monte Carlo). The package is designed 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) packages powered by **rstan**. Examples of packages that will soon (or already are) using **bayesplot** are **rstan** itself, as well as the **rstan**-dependent **rstanarm** and **brms** packages for applied regression modeling.

## Plotting functionality

The plotting functions in **bayesplot** are organized into several modules:

- **MCMC**: Visualizations of Markov chain Monte Carlo (MCMC) simulations generated by *any* MCMC algorithm as well as diagnostics. There are also additional functions specifically for use with models fit using the [No-U-Turn Sampler \(NUTS\)](#).
- **PPC**: Graphical posterior predictive checks (PPCs).

In future releases modules will be added specifically for forecasting/out-of-sample prediction and other inference-related tasks.

## Resources

- **Bug reports and feature requests**: If you'd like to request a new feature or if you've noticed a bug that needs to be fixed please let us know at the **bayesplot** issue tracker on GitHub: <https://github.com/stan-dev/bayesplot/issues/>.
- **General questions and help**: To ask a question about **bayesplot** on the Stan Forums forum please visit <http://discourse.mc-stan.org>.

## References

Gabry, J., Simpson, D., Vehtari, A., Betancourt, M., Gelman, A. (2017). Visualization in Bayesian workflow. arXiv preprint [arxiv:1709.01449](https://arxiv.org/abs/1709.01449).

## See Also

[theme\\_default](#) for the default ggplot theme used by **bayesplot**.  
[bayesplot-colors](#) to set or view the color scheme used for plotting.  
[ggsave](#) in **ggplot2** for saving plots.

## Examples

```
# A few quick examples (all of the functions have many examples
# on their individual help pages)

# MCMC plots
x <- example_mcmc_draws(params = 5)
mcmc_intervals(x, prob = 0.5)
mcmc_intervals(x, regex_pars = "beta")

color_scheme_set("purple")
mcmc_areas(x, regex_pars = "beta", prob = 0.8)

color_scheme_set("mix-blue-red")
mcmc_trace(x, pars = c("alpha", "sigma"),
           facet_args = list(nrow = 2))

color_scheme_set("brightblue")
```

```
mcmc_scatter(x, pars = c("beta[1]", "sigma"),
             transformations = list(sigma = "log"))

# Graphical PPCs
y <- example_y_data()
yrep <- example_yrep_draws()
ppc_dens_overlay(y, yrep[1:50, ])

color_scheme_set("pink")
ppc_stat(y, yrep, stat = "median") + grid_lines()
ppc_hist(y, yrep[1:8, ])
```

---

available\_ppc

*Get or view the names of available plotting functions*

---

## Description

Get or view the names of available plotting functions

## Usage

```
available_ppc(pattern, fixed = FALSE, invert = FALSE)
```

```
available_mcmc(pattern, fixed = FALSE, invert = FALSE)
```

## Arguments

pattern, fixed, invert  
Passed to [grep](#).

## Value

A possibly empty character vector of function names with several additional attributes (for use by a custom print method). If pattern is missing then the returned object contains the names of all available plotting functions in the [MCMC](#) or [PPC](#) module, depending on which function is called. If pattern is specified then a subset of function names is returned.

## Examples

```
available_mcmc()
available_mcmc("nuts")
available_mcmc("rhat|neff")
available_ppc("grouped")
available_ppc("grouped", invert = TRUE)
```

---

bayesplot-colors      *Set, get, or view color schemes*

---

### Description

Set, get, or view color schemes. Choose from a preset scheme or create a custom scheme.

### Usage

```
color_scheme_set(scheme = "blue")
```

```
color_scheme_get(scheme, i)
```

```
color_scheme_view(scheme)
```

### Arguments

**scheme**      For `color_scheme_set`, either a string naming one of the available color schemes or a character vector of *exactly six* colors specifying a custom scheme (see the **Custom Color Schemes** section, below, for more on specifying a custom scheme).

For `color_scheme_get`, `scheme` can be missing (to get the current color scheme) or a string naming one of the preset schemes.

For `color_scheme_view`, `scheme` can be missing (to use the current color scheme) or a character vector containing a subset of the available scheme names.

Currently, the available preset color schemes are:

- "blue", "brightblue"
- "gray", "darkgray"
- "green"
- "pink"
- "purple"
- "red"
- "teal"
- "yellow"
- "**viridis**", "viridisA", "viridisB", "viridisC"
- "mix-x-y", replacing `x` and `y` with any two of the scheme names listed above (e.g. "mix-teal-pink", "mix-blue-red", etc.). The order of `x` and `y` matters, i.e., the color schemes "mix-blue-red" and "mix-red-blue" are not identical. There is no guarantee that every possible mixed scheme will look good with every possible plot.

If you have a suggestion for a new color scheme please let us know via the **bayesplot issue tracker**.

**i**      For `color_scheme_get`, a subset of the integers from 1 (lightest) to 6 (darkest) indicating which of the colors in the scheme to return. If `i` is not specified then all six colors in the scheme are included.

**Value**

`color_scheme_set` has the side effect of setting the color scheme used for plotting. It also returns (*invisibly*) a list of the hexadecimal color values used in scheme.

`color_scheme_get` returns a list of the hexadecimal color values (without changing the current scheme). If the `scheme` argument is not specified the returned values correspond to the current color scheme. If the optional argument `i` is specified then the returned list only contains `length(i)` elements.

`color_scheme_view` returns a `ggplot` object if only a single scheme is specified and a `gtable` object if multiple schemes names are specified.

**Custom Color Schemes**

A **bayesplot** color scheme consists of six colors. To specify a custom color scheme simply pass a character vector containing either the names of six `colors` or six hexadecimal color values (or a mix of names and hex values). The colors should be in order from lightest to darkest. See the end of the **Examples** section for a demonstration.

**See Also**

`theme_default` for the default `ggplot` theme used by **bayesplot**.

**Examples**

```
color_scheme_set("blue")
color_scheme_view()

color_scheme_get()
color_scheme_get(i = c(3, 5)) # 3rd and 5th colors only

color_scheme_get("brightblue")
color_scheme_view("brightblue")

# compare multiple schemes
color_scheme_view(c("pink", "gray", "teal"))
color_scheme_view(c("viridis", "viridisA", "viridisB", "viridisC"))

color_scheme_set("pink")
x <- example_mcmc_draws()
mcmc_intervals(x)

color_scheme_set("teal")
color_scheme_view()
mcmc_intervals(x)

color_scheme_set("red")
mcmc_areas(x, regex_pars = "beta")

color_scheme_set("purple")
color_scheme_view()
y <- example_y_data()
```

```

yrep <- example_yrep_draws()
ppc_stat(y, yrep, stat = "mean") + legend_none()

color_scheme_set("mix-teal-pink")
ppc_stat(y, yrep, stat = "sd") + legend_none()
mcmc_areas(x, regex_pars = "beta")

#####
### custom color scheme ###
#####
orange_scheme <- c("#ffebcc", "#ffcc80",
                  "#ffad33", "#e68a00",
                  "#995c00", "#663d00")
color_scheme_set(orange_scheme)
mcmc_areas(x, regex_pars = "alpha")
mcmc_dens_overlay(x)
ppc_stat(y, yrep, stat = "var") + legend_none()

```

---

bayesplot-extractors *Extract quantities needed for plotting from model objects*

---

## Description

Generics and methods for extracting quantities needed for plotting from various types of model objects. Currently methods are only provided for stanfit (**rstan**) and stanreg (**rstanarm**) objects, but adding new methods should be relatively straightforward.

## Usage

```

log_posterior(object, ...)

nuts_params(object, ...)

rhat(object, ...)

neff_ratio(object, ...)

## S3 method for class 'stanfit'
log_posterior(object, inc_warmup = FALSE, ...)

## S3 method for class 'stanreg'
log_posterior(object, inc_warmup = FALSE, ...)

## S3 method for class 'stanfit'
nuts_params(object, pars = NULL, inc_warmup = FALSE, ...)

## S3 method for class 'stanreg'

```

```

nuts_params(object, pars = NULL, inc_warmup = FALSE, ...)

## S3 method for class 'list'
nuts_params(object, pars = NULL, ...)

## S3 method for class 'stanfit'
rhat(object, pars = NULL, ...)

## S3 method for class 'stanreg'
rhat(object, pars = NULL, regex_pars = NULL, ...)

## S3 method for class 'stanfit'
neff_ratio(object, pars = NULL, ...)

## S3 method for class 'stanreg'
neff_ratio(object, pars = NULL, regex_pars = NULL, ...)

```

### Arguments

object	The object to use.
...	Arguments passed to individual methods.
inc_warmup	A logical scalar (defaulting to FALSE) indicating whether to include warmup draws, if applicable.
pars	An optional character vector of parameter names. For nuts_params these will be NUTS sampler parameter names rather than model parameters. If pars is omitted all parameters are included.
regex_pars	An optional <a href="#">regular expression</a> to use for parameter selection. Can be specified instead of pars or in addition to pars.

### Value

log\_posterior log\_posterior methods return a molten data frame (see [melt](#)). The data frame should have columns "Iteration" (integer), "Chain" (integer), and "Value" (numeric). See **Examples**, below.

nuts\_params nuts\_params methods return a molten data frame (see [melt](#)). The data frame should have columns "Parameter" (factor), "Iteration" (integer), "Chain" (integer), and "Value" (numeric). See **Examples**, below.

rhat, neff\_ratio Methods return (named) vectors.

### See Also

[MCMC-nuts](#), [MCMC-diagnostics](#)

### Examples

```

## Not run:
library(rstanarm)
fit <- stan_glm(mpg ~ wt, data = mtcars)

```



```
np <- nuts_params(fit)
head(np)
tail(np)

lp <- log_posterior(fit)
head(lp)
tail(lp)

## End(Not run)
```

---

bayesplot-helpers      *Convenience functions for adding or changing plot details*

---

### Description

Convenience functions for adding to (and changing details of) ggplot objects (many of the objects returned by **bayesplot** functions). See the **Examples** section, below.

### Usage

```
vline_at(v, fun, ..., na.rm = TRUE)

hline_at(v, fun, ..., na.rm = TRUE)

vline_0(..., na.rm = TRUE)

hline_0(..., na.rm = TRUE)

abline_01(..., na.rm = TRUE)

lbub(p, med = TRUE)

legend_move(position = "right")

legend_none()

legend_text(...)

xaxis_title(on = TRUE, ...)

xaxis_text(on = TRUE, ...)

xaxis_ticks(on = TRUE, ...)

yaxis_title(on = TRUE, ...)
```

```

yaxis_text(on = TRUE, ...)
yaxis_ticks(on = TRUE, ...)
facet_text(on = TRUE, ...)
facet_bg(on = TRUE, ...)
panel_bg(on = TRUE, ...)
plot_bg(on = TRUE, ...)
grid_lines(color = "gray50", size = 0.2)
overlay_function(...)

```

### Arguments

v	Either a numeric vector specifying the value(s) at which to draw the vertical or horizontal line(s), or an object of any type to use as the first argument to fun.
fun	A function, or the name of a function, that returns a numeric vector.
...	For the various <code>vline_</code> , <code>hline_</code> , and <code>abline_</code> functions, ... is passed to <code>geom_vline</code> , <code>geom_hline</code> , and <code>geom_abline</code> , respectively, to control the appearance of the line(s). For functions ending in <code>_bg</code> , ... is passed to <code>element_rect</code> . For functions ending in <code>_text</code> or <code>_title</code> , ... is passed to <code>element_text</code> . For <code>xaxis_ticks</code> and <code>yaxis_ticks</code> , ... is passed to <code>element_line</code> . For <code>overlay_function</code> , ... is passed to <code>stat_function</code> .
na.rm	A logical scalar passed to the appropriate geom (e.g. <code>geom_vline</code> ). The default is TRUE.
p	The probability mass (in [0,1]) to include in the interval.
med	Should the median also be included in addition to the lower and upper bounds of the interval?
position	The position of the legend. Either a numeric vector (of length 2) giving the relative coordinates (between 0 and 1) for the legend, or a string among "right", "left", "top", "bottom". Using <code>position = "none"</code> is also allowed and is equivalent to using <code>legend_none()</code> .
on	For functions modifying ggplot <code>theme</code> elements, set <code>on=FALSE</code> to set the element to <code>element_blank</code> . For example, facet text can be removed by adding <code>facet_text(on=FALSE)</code> , or simply <code>facet_text(FALSE)</code> to a ggplot object. If <code>on=TRUE</code> (the default), then ... can be used to customize the appearance of the theme element.
color, size	Passed to <code>element_line</code> .

## Details

### Add vertical, horizontal, and diagonal lines to plots:

- `vline_at` and `hline_at` return an object created by either `geom_vline` or `geom_hline` that can be added to a `ggplot` object to draw a vertical or horizontal line (at one or several values). If `fun` is missing then the lines are drawn at the values in `v`. If `fun` is specified then the lines are drawn at the values returned by `fun(v)`.
- `vline_0` and `hline_0` are wrappers for `vline_at` and `hline_at` with `v = 0` and `fun` missing.
- `abline_01` is a wrapper for `geom_abline` with the intercept set to 0 and the slope set to 1.
- `lbub` returns a *function* that takes a single argument `x` and returns the lower and upper bounds (`lb`, `ub`) of the  $100 * p\%$  central interval of `x`, as well as the median (if `med` is `TRUE`).

### Control appearance of facet strips:

- `facet_text` and `facet_bg` return `ggplot2` theme objects that can be added to an existing plot (`ggplot` object) to format the text and the background for the facet strips.

### Move legend, remove legend, or style the legend text:

- `legend_move` and `legend_none` return a `ggplot2` theme object that can be added to an existing plot (`ggplot` object) in order to change the position of the legend (`legend_move`) or remove the legend (`legend_none`). `legend_text` works much like `facet_text`, except it controls the legend text.

### Control appearance of *x*-axis and *y*-axis features:

- `xaxis_title` and `yaxis_title` return a `ggplot2` theme object that can be added to an existing plot (`ggplot` object) in order to toggle or format the titles displayed on the *x* or *y* axis. (To change the titles themselves use [labs](#).)
- `xaxis_text` and `yaxis_text` return a `ggplot2` theme object that can be added to an existing plot (`ggplot` object) in order to toggle or format the text displayed on the *x* or *y* axis (e.g. tick labels).
- `xaxis_ticks` and `yaxis_ticks` return a `ggplot2` theme object that can be added to an existing plot (`ggplot` object) to change the appearance of the axis tick marks.

### Customize plot background:

- `plot_bg` returns a `ggplot2` theme object that can be added to an existing plot (`ggplot` object) to format the background of the *entire* plot.
- `panel_bg` returns a `ggplot2` theme object that can be added to an existing plot (`ggplot` object) to format the background of the just the plotting area.
- `grid_lines` returns a `ggplot2` theme object that can be added to an existing plot (`ggplot` object) to add grid lines to the plot background.

### Superimpose a function on an existing plot:

- `overlay_function` is a simple wrapper for [stat\\_function](#) but with the `inherit.aes` argument fixed to `FALSE`. Fixing `inherit.aes=FALSE` will avoid potential errors due to the [aesthetic](#) mapping used by certain **bayesplot** plotting functions).

## Value

A `ggplot2` layer or [theme](#) object that can be added to existing `ggplot` objects, like those created by many of the **bayesplot** plotting functions. See the **Details** section.

**See Also**

[theme\\_default](#) for the default ggplot theme used by **bayesplot**.

**Examples**

```

color_scheme_set("gray")
x <- example_mcmc_draws(chains = 1)
dim(x)
colnames(x)

#####
### vertical & horizontal lines ###
#####
(p <- mcmc_intervals(x, regex_pars = "beta"))

# vertical line at zero (with some optional styling)
p + vline_0()
p + vline_0(size = 0.25, color = "darkgray", linetype = 2)

# vertical line(s) at specified values
v <- c(-0.5, 0, 0.5)
p + vline_at(v, linetype = 3, size = 0.25)

my_lines <- vline_at(v, alpha = 0.25, size = 0.75 * c(1, 2, 1),
                    color = c("maroon", "skyblue", "violet"))
p + my_lines

# add vertical line(s) at computed values
# (three ways of getting lines at column means)
color_scheme_set("brightblue")
p <- mcmc_intervals(x, regex_pars = "beta")
p + vline_at(x[, 3:4], colMeans)
p + vline_at(x[, 3:4], "colMeans", color = "darkgray",
            lty = 2, size = 0.25)
p + vline_at(x[, 3:4], function(a) apply(a, 2, mean),
            color = "orange",
            size = 2, alpha = 0.1)

# using the lbub function to get interval lower and upper bounds (lb, ub)
color_scheme_set("pink")
parsed <- ggplot2::label_parsed
p2 <- mcmc_hist(x, pars = "beta[1]", binwidth = 1/20,
              facet_args = list(labeller = parsed))
(p2 <- p2 + facet_text(size = 16))

b1 <- x[, "beta[1]"]
p2 + vline_at(b1, fun = lbub(0.8), color = "gray20",
              size = 2 * c(1,.5,1), alpha = 0.75)
p2 + vline_at(b1, lbub(0.8, med = FALSE), color = "gray20",

```

```

size = 2, alpha = 0.75)

#####
### format axis titles ###
#####
color_scheme_set("green")
y <- example_y_data()
yrep <- example_yrep_draws()
(p3 <- ppc_stat(y, yrep, stat = "median", binwidth = 1/4))

# turn off the legend, turn on x-axis title
p3 +
  legend_none() +
  xaxis_title(size = 13, family = "sans") +
  ggplot2::xlab(expression(italic(T(y)) == median(italic(y))))

#####
### format axis & facet text ###
#####
color_scheme_set("gray")
p4 <- mcmc_trace(example_mcmc_draws(), pars = c("alpha", "sigma"))

myfacets <-
  facet_bg(fill = "gray30", color = NA) +
  facet_text(face = "bold", color = "skyblue", size = 14)
p4 + myfacets

#####
### control tick marks ###
#####
p4 +
  myfacets +
  yaxis_text(FALSE) +
  yaxis_ticks(FALSE) +
  xaxis_ticks(size = 1, color = "skyblue")

#####
### change plot background ###
#####
color_scheme_set("blue")

# add grid lines
ppc_stat(y, yrep) + grid_lines()

# panel_bg vs plot_bg
ppc_scatter_avg(y, yrep) + panel_bg(fill = "gray90")
ppc_scatter_avg(y, yrep) + plot_bg(fill = "gray90")

color_scheme_set("yellow")

```

```

p5 <- ppc_scatter_avg(y, yrep, alpha = 1)
p5 + panel_bg(fill = "gray20") + grid_lines(color = "white")

color_scheme_set("purple")
ppc_dens_overlay(y, yrep[1:30, ]) +
  legend_text(size = 14) +
  legend_move(c(0.75, 0.5)) +
  plot_bg(fill = "gray90") +
  panel_bg(color = "black", fill = "gray99", size = 3)

#####
### superimpose a function on existing plot ###
#####
# compare posterior of beta[1] to Gaussian with same posterior mean
# and sd as beta[1]
x <- example_mcmc_draws(chains = 4)
dim(x)
purple_gaussian <-
  overlay_function(
    fun = dnorm,
    args = list(mean(x[, "beta[1]"], sd(x[, "beta[1]"])),
               color = "purple",
               size = 2
    )

color_scheme_set("gray")
mcmc_hist(x, pars = "beta[1]") + purple_gaussian
mcmc_dens(x, pars = "beta[1]") + purple_gaussian

```

---

 bayesplot\_grid

*Arrange plots in a grid*


---

## Description

The `bayesplot_grid` function makes it simple to juxtapose plots using common  $x$  and/or  $y$  axes.

## Usage

```

bayesplot_grid(..., plots = list(), xlim = NULL, ylim = NULL,
  grid_args = list(), titles = character(), subtitles = character(),
  legends = TRUE)

```

## Arguments

<code>...</code>	One or more ggplot objects.
<code>plots</code>	A list of ggplot objects. Can be used as an alternative to specifying plot objects via <code>...</code>

xlim, ylim	Optionally, numeric vectors of length 2 specifying lower and upper limits for the axes that will be shared across all plots.
grid_args	An optional named list of arguments to pass to <a href="#">arrangeGrob</a> (nrow, ncol, widths, etc.).
titles, subtitles	Optional character vectors of plot titles and subtitles. If specified, titles and subtitles must have length equal to the number of plots specified.
legends	If any of the plots have legends should they be displayed? Defaults to TRUE.

### Value

An object of class "bayesplot\_grid" (essentially a gtable object from [arrangeGrob](#)), which has a plot method.

### Examples

```

y <- example_y_data()
yrep <- example_yrep_draws()
stats <- c("sd", "median", "max", "min")

color_scheme_set("pink")
bayesplot_grid(
  plots = lapply(stats, function(s) ppc_stat(y, yrep, stat = s)),
  titles = stats,
  legends = FALSE,
  grid_args = list(ncol = 1)
)

## Not run:
library(rstanarm)
mtcars$log_mpg <- log(mtcars$mpg)
fit1 <- stan_glm(mpg ~ wt, data = mtcars)
fit2 <- stan_glm(log_mpg ~ wt, data = mtcars)

y <- mtcars$mpg
yrep1 <- posterior_predict(fit1, draws = 50)
yrep2 <- posterior_predict(fit2, fun = exp, draws = 50)

color_scheme_set("blue")
ppc1 <- ppc_dens_overlay(y, yrep1)
ppc1
ppc1 + yaxis_text()

color_scheme_set("red")
ppc2 <- ppc_dens_overlay(y, yrep2)
bayesplot_grid(ppc1, ppc2)

# make sure the plots use the same limits for the axes
bayesplot_grid(ppc1, ppc2, xlim = c(-5, 60), ylim = c(0, 0.2))

# remove the legends and add text

```

```

bayesplot_grid(ppc1, ppc2, xlim = c(-5, 60), ylim = c(0, 0.2),
               legends = FALSE, subtitles = rep("Predicted MPG", 2))

## End(Not run)

```

MCMC-combos

*Combination plots***Description**

Combination plots

**Usage**

```

mcmc_combo(x, combo = c("dens", "trace"), widths = NULL, gg_theme = NULL,
           ...)

```

**Arguments**

x	A 3-D array, matrix, list of matrices, or data frame of MCMC draws. The <a href="#">MCMC-overview</a> page provides details on how to specify each these allowed inputs.
combo	A character vector with at least two elements. Each element of combo corresponds to a column in the resulting graphic and should be the name of one of the available <a href="#">MCMC</a> functions (omitting the <code>mcmc_</code> prefix).
widths	A numeric vector the same length as <code>combo</code> specifying relative column widths. For example, if the plot has two columns, then <code>widths = c(2, 1)</code> will allocate more space for the first column by a factor of 2 (as would <code>widths = c(.3, .15)</code> , etc.). The default, <code>NULL</code> , allocates the same horizontal space for each column.
gg_theme	Unlike most of the other <b>bayesplot</b> functions, <code>mcmc_combo</code> returns a <code>gtable</code> object rather than a <code>ggplot</code> object, and so theme objects can't be added directly to the returned plot object. The <code>gg_theme</code> argument helps get around this problem by accepting a <b>ggplot2</b> theme object that is added to each of the plots <i>before</i> combining them into the <code>gtable</code> object that is returned. This can be a theme object created by a call to <code>ggplot2::theme</code> or one of the <b>bayesplot</b> convenience functions, e.g. <a href="#">legend_none</a> (see the <b>Examples</b> section, below).
...	Arguments passed to the plotting functions named in <code>combo</code> .

**Value**

A `gtable` object (the result of calling [arrangeGrob](#)) with `length(combo)` columns and a row for each parameter.

**See Also**

Other MCMC: [MCMC-diagnostics](#), [MCMC-distributions](#), [MCMC-intervals](#), [MCMC-nuts](#), [MCMC-overview](#), [MCMC-parcoord](#), [MCMC-recover](#), [MCMC-scatterplots](#), [MCMC-traces](#)



## Examples

```
# some parameter draws to use for demonstration
x <- example_mcmc_draws()
dim(x)
dimnames(x)

mcmc_combo(x, pars = c("alpha", "sigma"))
mcmc_combo(x, pars = c("alpha", "sigma"), widths = c(1, 2))

# change second plot, show log(sigma) instead of sigma,
# and remove the legends
color_scheme_set("mix-blue-red")
mcmc_combo(
  x,
  combo = c("dens_overlay", "trace"),
  pars = c("alpha", "sigma"),
  transformations = list(sigma = "log"),
  gg_theme = legend_none()
)

# same thing but this time also change the entire ggplot theme
mcmc_combo(
  x,
  combo = c("dens_overlay", "trace"),
  pars = c("alpha", "sigma"),
  transformations = list(sigma = "log"),
  gg_theme = ggplot2::theme_gray() + legend_none()
)
```

---

MCMC-diagnostics

*General MCMC diagnostics*

---

## Description

Plots of Rhat statistics, ratios of effective sample size to total sample size, and autocorrelation of MCMC draws. See the **Plot Descriptions** section, below, for details. For models fit using the No-U-Turn-Sampler, see also [MCMC-nuts](#) for additional MCMC diagnostic plots.

## Usage

```
mcmc_rhat(rhat, ..., size = NULL)

mcmc_rhat_hist(rhat, ..., binwidth = NULL)

mcmc_rhat_data(rhat, ...)
```

```

mcmc_neff(ratio, ..., size = NULL)

mcmc_neff_hist(ratio, ..., binwidth = NULL)

mcmc_neff_data(ratio, ...)

mcmc_acf(x, pars = character(), regex_pars = character(),
  facet_args = list(), ..., lags = 20, size = NULL)

mcmc_acf_bar(x, pars = character(), regex_pars = character(),
  facet_args = list(), ..., lags = 20)

```

### Arguments

rhat	A vector of <a href="#">Rhat</a> estimates.
...	Currently ignored.
size	An optional value to override <a href="#">geom_point</a> 's default size (for <code>mcmc_rhat</code> , <code>mcmc_neff</code> ) or <a href="#">geom_line</a> 's default size (for <code>mcmc_acf</code> ).
binwidth	An optional value used as the binwidth argument to <a href="#">geom_histogram</a> to override the default binwidth.
ratio	A vector of <i>ratios</i> of effective sample size estimates to total sample size. See <a href="#">neff_ratio</a> .
x	A 3-D array, matrix, list of matrices, or data frame of MCMC draws. The <a href="#">MCMC-overview</a> page provides details on how to specify each these allowed inputs.
pars	An optional character vector of parameter names. If neither <code>pars</code> nor <code>regex_pars</code> is specified then the default is to use <i>all</i> parameters.
regex_pars	An optional <a href="#">regular expression</a> to use for parameter selection. Can be specified instead of <code>pars</code> or in addition to <code>pars</code> .
facet_args	A named list of arguments (other than facets) passed to <a href="#">facet_wrap</a> or <a href="#">facet_grid</a> to control faceting.
lags	The number of lags to show in the autocorrelation plot.

### Value

A `ggplot` object that can be further customized using the [ggplot2](#) package. The `_data` functions return the data that would have be drawn by the plotting function.

### Plot Descriptions

`mcmc_rhat`, `mcmc_rhat_hist` Rhat values as either points or a histogram. Values are colored using different shades (lighter is better). The chosen thresholds are somewhat arbitrary, but can be useful guidelines in practice.

- *light*: below 1.05 (good)
- *mid*: between 1.05 and 1.1 (ok)
- *dark*: above 1.1 (too high)

`mcmc_neff`, `mcmc_neff_hist` Ratios of effective sample size to total sample size as either points or a histogram. Values are colored using different shades (lighter is better). The chosen thresholds are somewhat arbitrary, but can be useful guidelines in practice.

- *light*: between 0.5 and 1 (high)
- *mid*: between 0.1 and 0.5 (good)
- *dark*: below 0.1 (low)

`mcmc_acf` Grid of autocorrelation plots by chain and parameter. The `lags` argument gives the maximum number of lags at which to calculate the autocorrelation function. `mcmc_acf` is a line plot whereas `mcmc_acf_bar` is a barplot.

## References

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

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

## See Also

- The *Visual MCMC Diagnostics* vignette.
- [MCMC-nuts](#) for additional MCMC diagnostic plots for models fit using the No-U-Turn-Sampler.

Other MCMC: [MCMC-combos](#), [MCMC-distributions](#), [MCMC-intervals](#), [MCMC-nuts](#), [MCMC-overview](#), [MCMC-parcoord](#), [MCMC-recover](#), [MCMC-scatterplots](#), [MCMC-traces](#)

## Examples

```
# autocorrelation
x <- example_mcmc_draws()
dim(x)
dimnames(x)

color_scheme_set("green")
mcmc_acf(x, pars = c("alpha", "beta[1]"))

color_scheme_set("pink")
(p <- mcmc_acf_bar(x, pars = c("alpha", "beta[1]")))

# add horizontal dashed line at 0.5
p + hline_at(0.5, linetype = 2, size = 0.15, color = "gray")

# fake rhat values to use for demonstration
rhat <- c(runif(100, 1, 1.15))
mcmc_rhat_hist(rhat)
mcmc_rhat(rhat)

# lollipops
color_scheme_set("purple")
```

```

mcmc_rhat(rhat[1:10], size = 5)

color_scheme_set("blue")
mcmc_rhat(runif(1000, 1, 1.07))
mcmc_rhat(runif(1000, 1, 1.3)) + legend_move("top") # add legend above plot

# fake neff ratio values to use for demonstration
ratio <- c(runif(100, 0, 1))
mcmc_neff_hist(ratio)
mcmc_neff(ratio)

## Not run:
# Example using rstanarm model (requires rstanarm package)
library(rstanarm)

# intentionally use small 'iter' so there are some
# problems with rhat and neff for demonstration
fit <- stan_glm(mpg ~ ., data = mtcars, iter = 50)
rhats <- rhat(fit)
ratios <- neff_ratio(fit)
mcmc_rhat(rhats)
mcmc_neff(ratios, size = 3)

# there's a small enough number of parameters in the
# model that we can display their names on the y-axis
mcmc_neff(ratios) + yaxis_text(hjust = 1)

# can also look at autocorrelation
draws <- as.array(fit)
mcmc_acf(draws, pars = c("wt", "cyl"), lags = 10)

# increase number of iterations and plots look much better
fit2 <- update(fit, iter = 500)
mcmc_rhat(rhat(fit2))
mcmc_neff(neff_ratio(fit2))
mcmc_acf(as.array(fit2), pars = c("wt", "cyl"), lags = 10)

## End(Not run)

```

## Description

Various types of histograms and kernel density plots of MCMC draws. See the **Plot Descriptions** section, below, for details.

**Usage**

```

mcmc_hist(x, pars = character(), regex_pars = character(),
  transformations = list(), facet_args = list(), ..., binwidth = NULL,
  freq = TRUE)

mcmc_dens(x, pars = character(), regex_pars = character(),
  transformations = list(), facet_args = list(), ..., trim = FALSE)

mcmc_hist_by_chain(x, pars = character(), regex_pars = character(),
  transformations = list(), facet_args = list(), ..., binwidth = NULL,
  freq = TRUE)

mcmc_dens_overlay(x, pars = character(), regex_pars = character(),
  transformations = list(), facet_args = list(), ..., trim = FALSE)

mcmc_violin(x, pars = character(), regex_pars = character(),
  transformations = list(), facet_args = list(), ..., probs = c(0.1, 0.5,
  0.9))

```

**Arguments**

<code>x</code>	A 3-D array, matrix, list of matrices, or data frame of MCMC draws. The <a href="#">MCMC-overview</a> page provides details on how to specify each these allowed inputs.
<code>pars</code>	An optional character vector of parameter names. If neither <code>pars</code> nor <code>regex_pars</code> is specified then the default is to use <i>all</i> parameters.
<code>regex_pars</code>	An optional <a href="#">regular expression</a> to use for parameter selection. Can be specified instead of <code>pars</code> or in addition to <code>pars</code> .
<code>transformations</code>	Optionally, transformations to apply to parameters before plotting. If <code>transformations</code> is a function or a single string naming a function then that function will be used to transform all parameters. To apply transformations to particular parameters, the <code>transformations</code> argument can be a named list with length equal to the number of parameters to be transformed. Currently only univariate transformations of scalar parameters can be specified (multivariate transformations will be implemented in a future release). If <code>transformations</code> is a list, the name of each list element should be a parameter name and the content of each list element should be a function (or any item to match as a function via <a href="#">match.fun</a> , e.g. a string naming a function). If a function is specified by its name as a string (e.g. "log"), then it can be used to construct a new parameter label for the appropriate parameter (e.g. "log(sigma)"). If a function itself is specified (e.g. log or function(x) log(x)) then "t" is used in the new parameter label to indicate that the parameter is transformed (e.g. "t(sigma)").
<code>facet_args</code>	A named list of arguments (other than facets) passed to <a href="#">facet_wrap</a> or <a href="#">facet_grid</a> to control faceting.
<code>...</code>	Currently ignored.

binwidth	An optional value used as the binwidth argument to <a href="#">geom_histogram</a> to override the default binwidth.
freq	For histograms, freq=TRUE (the default) puts count on the y-axis. Setting freq=FALSE puts density on the y-axis. (For many plots the y-axis text is off by default. To view the count or density labels on the y-axis see the <a href="#">yaxis_text</a> convenience function.)
trim	A logical scalar passed to <a href="#">geom_density</a> .
probs	A numeric vector passed to <a href="#">geom_violin</a> 's draw_quantiles argument to specify at which quantiles to draw horizontal lines. Set to NULL to remove the lines.

**Value**

A ggplot object that can be further customized using the **ggplot2** package.

**Plot Descriptions**

`mcmc_hist` Histograms of posterior draws with all chains merged.

`mcmc_dens` Kernel density plots of posterior draws with all chains merged.

`mcmc_hist_by_chain` Histograms of posterior draws with chains separated via faceting.

`mcmc_dens_overlay` Kernel density plots of posterior draws with chains separated but overlaid on a single plot.

`mcmc_violin` The density estimate of each chain is plotted as a violin with horizontal lines at notable quantiles.

**See Also**

Other MCMC: [MCMC-combos](#), [MCMC-diagnostics](#), [MCMC-intervals](#), [MCMC-nuts](#), [MCMC-overview](#), [MCMC-parcoord](#), [MCMC-recover](#), [MCMC-scatterplots](#), [MCMC-traces](#)

**Examples**

```
# some parameter draws to use for demonstration
x <- example_mcmc_draws()
dim(x)
dimnames(x)

#####
### Histograms ###
#####

# histograms of all parameters
color_scheme_set("brightblue")
mcmc_hist(x)

# histograms of some parameters
color_scheme_set("pink")
mcmc_hist(x, pars = c("alpha", "beta[2]"))

mcmc_hist(x, pars = "sigma", regex_pars = "beta")
```

```

# example of using 'transformations' argument to plot log(sigma),
# and parsing facet labels (e.g. to get greek letters for parameters)
mcmc_hist(x, transformations = list(sigma = "log"),
          facet_args = list(labeler = ggplot2::label_parsed)) +
  facet_text(size = 15)

# instead of list(sigma = "log"), you could specify the transformation as
# list(sigma = log) or list(sigma = function(x) log(x)), but then the
# label for the transformed sigma is 't(sigma)' instead of 'log(sigma)'
mcmc_hist(x, transformations = list(sigma = log))

# separate histograms by chain
color_scheme_set("pink")
mcmc_hist_by_chain(x, regex_pars = "beta")

#####
### Densities ###
#####

mcmc_dens(x, pars = c("sigma", "beta[2]"),
          facet_args = list(nrow = 2))

# separate and overlay chains
color_scheme_set("mix-teal-pink")
mcmc_dens_overlay(x, pars = c("sigma", "beta[2]"),
                  facet_args = list(nrow = 2)) +
  facet_text(size = 14)

# separate chains as violin plots
color_scheme_set("green")
mcmc_violin(x) + panel_bg(color = "gray20", size = 2, fill = "gray30")

```

---

MCMC-intervals

*Plot interval estimates from MCMC draws*


---

## Description

Plot central (quantile-based) posterior interval estimates from MCMC draws. See the **Plot Descriptions** section, below, for details.

## Usage

```

mcmc_intervals(x, pars = character(), regex_pars = character(),
               transformations = list(), ..., prob = 0.5, prob_outer = 0.9,
               point_est = c("median", "mean", "none"), rhat = numeric())

mcmc_areas(x, pars = character(), regex_pars = character(),
            transformations = list(), ..., prob = 0.5, prob_outer = 1,

```

```
point_est = c("median", "mean", "none"), rhat = numeric(), bw = NULL,
adjust = NULL, kernel = NULL)
```

### Arguments

x	A 3-D array, matrix, list of matrices, or data frame of MCMC draws. The <a href="#">MCMC-overview</a> page provides details on how to specify each these allowed inputs.
pars	An optional character vector of parameter names. If neither pars nor regex_pars is specified then the default is to use <i>all</i> parameters.
regex_pars	An optional <a href="#">regular expression</a> to use for parameter selection. Can be specified instead of pars or in addition to pars.
transformations	Optionally, transformations to apply to parameters before plotting. If transformations is a function or a single string naming a function then that function will be used to transform all parameters. To apply transformations to particular parameters, the transformations argument can be a named list with length equal to the number of parameters to be transformed. Currently only univariate transformations of scalar parameters can be specified (multivariate transformations will be implemented in a future release). If transformations is a list, the name of each list element should be a parameter name and the content of each list element should be a function (or any item to match as a function via <a href="#">match.fun</a> , e.g. a string naming a function). If a function is specified by its name as a string (e.g. "log"), then it can be used to construct a new parameter label for the appropriate parameter (e.g. "log(sigma)"). If a function itself is specified (e.g. log or function(x) log(x)) then "t" is used in the new parameter label to indicate that the parameter is transformed (e.g. "t(sigma)").
...	Currently unused.
prob	The probability mass to include in the inner interval (for <code>mcmc_intervals</code> ) or in the shaded region (for <code>mcmc_areas</code> ). The default is 0.5 (50% interval).
prob_outer	The probability mass to include in the outer interval. The default is 0.9 for <code>mcmc_intervals</code> (90% interval) and 1 for <code>mcmc_areas</code> .
point_est	The point estimate to show. Either "median" (the default), "mean", or "none".
rhat	An optional numeric vector of $\hat{R}$ estimates, with one element per parameter included in x. If rhat is provided, the intervals/areas and point estimates in the resulting plot are colored based on $\hat{R}$ value. See <a href="#">rhat</a> for methods for extracting $\hat{R}$ estimates.
bw, adjust, kernel	For <code>mcmc_areas</code> , optional arguments passed to <a href="#">density</a> to override default kernel density estimation parameters.

### Value

A `ggplot` object that can be further customized using the [ggplot2](#) package.



**Plot Descriptions**

`mcmc_intervals` Plots of uncertainty intervals computed from posterior draws with all chains merged.

`mcmc_areas` Density plots computed from posterior draws with all chains merged, with uncertainty intervals shown as shaded areas under the curves.

**See Also**

Other MCMC: [MCMC-combos](#), [MCMC-diagnostics](#), [MCMC-distributions](#), [MCMC-nuts](#), [MCMC-overview](#), [MCMC-parcoord](#), [MCMC-recover](#), [MCMC-scatterplots](#), [MCMC-traces](#)

**Examples**

```
# some parameter draws to use for demonstration
x <- example_mcmc_draws(params = 6)
dim(x)
dimnames(x)

color_scheme_set("brightblue")
mcmc_intervals(x)
mcmc_intervals(x, pars = c("beta[1]", "beta[2]"))
mcmc_areas(x, regex_pars = "beta\\[[1-3]", prob = 0.8) +
  ggplot2::labs(
    title = "Posterior distributions",
    subtitle = "with medians and 80% intervals"
  )

color_scheme_set("red")
mcmc_areas(
  x,
  pars = c("alpha", "beta[4]"),
  prob = 2/3,
  prob_outer = 0.9,
  point_est = "mean"
)

# color by rhat value
color_scheme_set("blue")
fake_rhat_values <- c(1, 1.07, 1.3, 1.01, 1.15, 1.005)
mcmc_intervals(x, rhat = fake_rhat_values)

color_scheme_set("gray")
p <- mcmc_areas(x, pars = c("alpha", "beta[4]"), rhat = c(1, 1.1))
p + legend_move("bottom")
p + legend_move("none") # or p + legend_none()

# apply transformations
mcmc_intervals(
  x,
  pars = c("beta[2]", "sigma"),
```

```

transformations = list("sigma" = "log", "beta[2]" = function(x) x + 3)
)

# apply same transformation to all selected parameters
mcmc_intervals(x, regex_pars = "beta", transformations = "exp")

## Not run:
# example using fitted model from rstanarm package
library(rstanarm)
fit <- stan_glm(
  mpg ~ 0 + wt + factor(cyl),
  data = mtcars,
  iter = 500
)
x <- as.matrix(fit)

color_scheme_set("teal")
mcmc_intervals(x, point_est = "mean", prob = 0.8, prob_outer = 0.95)
mcmc_areas(x, regex_pars = "cyl", bw = "SJ",
           rhat = rhat(fit, regex_pars = "cyl"))

## End(Not run)

```

## Description

Diagnostic plots for the No-U-Turn-Sampler (NUTS), the default MCMC algorithm used by Stan. See the **Plot Descriptions** section, below.

## Usage

```

mcmc_nuts_acceptance(x, lp, chain = NULL, ..., binwidth = NULL)

mcmc_nuts_divergence(x, lp, chain = NULL, ...)

mcmc_nuts_stepsize(x, lp, chain = NULL, ...)

mcmc_nuts_treedepth(x, lp, chain = NULL, ...)

mcmc_nuts_energy(x, ..., binwidth = NULL, alpha = 0.5,
                 merge_chains = FALSE)

```

**Arguments**

x	A molten data frame of NUTS sampler parameters, either created by <code>nuts_params</code> or in the same form as the object returned by <code>nuts_params</code> .
lp	A molten data frame of draws of the log-posterior or, more commonly, of a quantity equal to the log-posterior up to a constant. <code>lp</code> should either be created via <code>log_posterior</code> or be an object with the same form as the object returned by <code>log_posterior</code> .
chain	A positive integer for selecting a particular chain. The default (NULL) is to merge the chains before plotting. If <code>chain = k</code> then the plot for chain <code>k</code> is overlaid (in a darker shade but with transparency) on top of the plot for all chains. The <code>chain</code> argument is not used by <code>mcmc_nuts_energy</code> .
...	Currently ignored.
binwidth	An optional value used as the <code>binwidth</code> argument to <code>geom_histogram</code> to override the default binwidth.
alpha	For <code>mcmc_nuts_energy</code> only, the transparency (alpha) level in [0,1] used for the overlaid histogram.
merge_chains	For <code>mcmc_nuts_energy</code> only, should all chains be merged or displayed separately? The default is FALSE, i.e., to show the chains separately.

**Value**

A `gtable` object (the result of calling `arrangeGrob`) created from several `ggplot` objects, except for `mcmc_nuts_energy`, which returns a `ggplot` object.

**Quick Definitions**

For more details see Stan Development Team (2016) and Betancourt (2017).

- `accept_stat__`: the average acceptance probabilities of all possible samples in the proposed tree.
- `divergent__`: the number of leapfrog transitions with diverging error. Because NUTS terminates at the first divergence this will be either 0 or 1 for each iteration.
- `stepsize__`: the step size used by NUTS in its Hamiltonian simulation.
- `treedepth__`: the depth of tree used by NUTS, which is the log (base 2) of the number of leapfrog steps taken during the Hamiltonian simulation.
- `energy__`: the value of the Hamiltonian (up to an additive constant) at each iteration.

**Plot Descriptions**

`mcmc_nuts_acceptance` Three plots:

- Histogram of `accept_stat__` with vertical lines indicating the mean (solid line) and median (dashed line).
- Histogram of `lp__` with vertical lines indicating the mean (solid line) and median (dashed line).
- Scatterplot of `accept_stat__` vs `lp__`.

`mcmc_nuts_divergence` Two plots:

- Violin plots of `lp__|divergent__=1` and `lp__|divergent__=0`.
- Violin plots of `accept_stat__|divergent__=1` and `accept_stat__|divergent__=0`.

`mcmc_nuts_stepsize` Two plots:

- Violin plots of `lp__` by chain ordered by `stepsize__` value.
- Violin plots of `accept_stat__` by chain ordered by `stepsize__` value.

`mcmc_nuts_treedepth` Three plots:

- Violin plots of `lp__` by value of `treedepth__`.
- Violin plots of `accept_stat__` by value of `treedepth__`.
- Histogram of `treedepth__`.

`mcmc_nuts_energy` Overlaid histograms showing `energy__` vs the change in `energy__`. See Betancourt (2016) for details.

## References

- Betancourt, M. (2017). A conceptual introduction to Hamiltonian Monte Carlo. <https://arxiv.org/abs/1701.02434>
- Betancourt, M. and Girolami, M. (2013). Hamiltonian Monte Carlo for hierarchical models. <https://arxiv.org/abs/1312.0906>
- 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.
- Stan Development Team. (2016). *Stan Modeling Language Users Guide and Reference Manual*. <http://mc-stan.org/documentation/>

## See Also

- The *Visual MCMC Diagnostics* vignette.
- Several other plotting functions in the **bayesplot** package are not NUTS-specific but take optional extra arguments if the model was fit using NUTS:
  - `mcmc_trace`: show divergences as tick marks below the trace plot.
  - `mcmc_parcoord`: change the color/size/transparency of lines corresponding to divergences.
  - `mcmc_scatter`: change the color/size/shape of points corresponding to divergences.
  - `mcmc_pairs`: change the color/size/shape of points corresponding divergences and/or max treedepth saturation.

Other MCMC: [MCMC-combos](#), [MCMC-diagnostics](#), [MCMC-distributions](#), [MCMC-intervals](#), [MCMC-overview](#), [MCMC-parcoord](#), [MCMC-recover](#), [MCMC-scatterplots](#), [MCMC-traces](#)

## Examples

```
## Not run:
library(ggplot2)
library(rstanarm)
fit <- stan_glm(mpg ~ wt + am, data = mtcars, iter = 1000)
np <- nuts_params(fit)
```

```

lp <- log_posterior(fit)

color_scheme_set("brightblue")
mcmc_nuts_acceptance(np, lp)
mcmc_nuts_acceptance(np, lp, chain = 2)

color_scheme_set("red")
mcmc_nuts_energy(np)
mcmc_nuts_energy(np, merge_chains = TRUE, binwidth = .15)
mcmc_nuts_energy(np) +
  facet_wrap(~ Chain, nrow = 1) +
  coord_fixed(ratio = 150) +
  ggtitle("NUTS Energy Diagnostic")

## End(Not run)

```

## Description

The **bayesplot** MCMC module provides various plotting functions for creating graphical displays of Markov chain Monte Carlo (MCMC) simulations. The **MCMC plotting functions** section, below, provides links to the documentation for various categories of MCMC plots. Currently the MCMC plotting functions accept posterior draws provided in one of the following formats:

- **3-D array**: An [array](#) with dimensions [Iteration, Chain, Parameter] in that order.
- **list**: A list of matrices, where each matrix corresponds to a Markov chain. All of the matrices should have the same number of iterations (rows) and parameters (columns), and parameters should have the same names and be in the same order.
- **matrix**: A [matrix](#) with one column per parameter. If using matrix there should only be a single Markov chain or all chains should already be merged (stacked).
- **data frame**: There are two types of [data frames](#) allowed. Either a data frame with one column per parameter (if only a single chain or all chains have already been merged), or a data frame with one column per parameter plus an additional column "Chain" that contains the chain number (an integer) corresponding to each row in the data frame.

**Note:** typically the user should *not* include warmup iterations in the object passed to **bayesplot** plotting functions, although for certain plots (e.g. trace plots) it can occasionally be useful to include the warmup iterations for diagnostic purposes.

## MCMC plotting functions

**Posterior distributions** Histograms and kernel density plots of parameter draws, optionally showing each Markov chain separately.

**Uncertainty intervals** Uncertainty intervals computed from parameter draws.

**Trace plots** Times series of parameter draws.

**Scatterplots** Scatterplots, heatmaps, and pairs plots of parameter draws.

**Combinations** Combination plots (e.g. trace plot + histogram).

**General MCMC diagnostics** MCMC diagnostic plots including Rhat, effective sample size, autocorrelation.

**NUTS diagnostics** Special diagnostic plots for the No-U-Turn Sampler.

**Comparisons to "true" values** Plots comparing MCMC estimates to "true" parameter values (e.g., values used to simulate data).

## References

Gabry, J., Simpson, D., Vehtari, A., Betancourt, M., Gelman, A. (2017). Visualization in Bayesian workflow. arXiv preprint [arxiv:1709.01449](https://arxiv.org/abs/1709.01449).

## See Also

Other MCMC: [MCMC-combos](#), [MCMC-diagnostics](#), [MCMC-distributions](#), [MCMC-intervals](#), [MCMC-nuts](#), [MCMC-parcoord](#), [MCMC-recover](#), [MCMC-scatterplots](#), [MCMC-traces](#)

---

MCMC-parcoord

*Parallel coordinates plot of MCMC draws*

---

## Description

Parallel coordinates plot of MCMC draws (one dimension per parameter). See the **Plot Descriptions** section, below, for details.

## Usage

```
mcmc_parcoord(x, pars = character(), regex_pars = character(),
  transformations = list(), ..., size = 0.2, alpha = 0.3, np = NULL,
  np_style = parcoord_style_np())
```

```
mcmc_parcoord_data(x, pars = character(), regex_pars = character(),
  transformations = list(), np = NULL)
```

```
parcoord_style_np(div_color = "red", div_size = 0.2, div_alpha = 0.2)
```

## Arguments

x	A 3-D array, matrix, list of matrices, or data frame of MCMC draws. The <a href="#">MCMC-overview</a> page provides details on how to specify each these allowed inputs.
pars	An optional character vector of parameter names. If neither pars nor regex_pars is specified then the default is to use <i>all</i> parameters.

regex_pars	An optional <a href="#">regular expression</a> to use for parameter selection. Can be specified instead of pars or in addition to pars.
transformations	Optionally, transformations to apply to parameters before plotting. If transformations is a function or a single string naming a function then that function will be used to transform all parameters. To apply transformations to particular parameters, the transformations argument can be a named list with length equal to the number of parameters to be transformed. Currently only univariate transformations of scalar parameters can be specified (multivariate transformations will be implemented in a future release). If transformations is a list, the name of each list element should be a parameter name and the content of each list element should be a function (or any item to match as a function via <a href="#">match.fun</a> , e.g. a string naming a function). If a function is specified by its name as a string (e.g. "log"), then it can be used to construct a new parameter label for the appropriate parameter (e.g. "log(sigma)"). If a function itself is specified (e.g. log or function(x) log(x)) then "t" is used in the new parameter label to indicate that the parameter is transformed (e.g. "t(sigma)").
...	Currently ignored.
size, alpha	Arguments passed on to <a href="#">geom_line</a> .
np	For models fit using <a href="#">NUTS</a> (more generally, any <a href="#">symplectic integrator</a> ), an optional data frame providing NUTS diagnostic information. The data frame should be the object returned by <a href="#">nuts_params</a> or one with the same structure.
np_style	A call to the parcoord_style_np helper function to specify arguments controlling the appearance of superimposed lines representing NUTS diagnostics (in this case divergences) if the np argument is specified.
div_color, div_size, div_alpha	Optional arguments to the parcoord_style_np helper function that are eventually passed to <a href="#">geom_line</a> if the np argument is also specified. They control the color, size, and transparency specifications for showing divergences in the plot. The default values are displayed in the <b>Usage</b> section above.

### Value

A ggplot object that can be further customized using the [ggplot2](#) package. The `_data` functions return the data that would have been drawn by the plotting function.

### Plot Descriptions

`mcmc_parcoord` ([Parallel coordinates plot](#)) of MCMC draws. There is one dimension per parameter along the horizontal axis and each set of connected line segments represents a single MCMC draw (i.e., a vector of length equal to the number of parameters).

The parallel coordinates plot is most useful if the optional HMC/NUTS diagnostic information is provided via the `np` argument. In that case divergences are highlighted in the plot. The appearance of the divergences can be customized using the `np_style` argument and the `parcoord_style_np` helper function.

When the plotted model parameters are on very different scales the `transformations` argument can be useful. For example, to standardize all variables before plotting you could

use function  $(x - \text{mean}(x))/\text{sd}(x)$  when specifying the transformations argument to `mcmc_parcoord`. See the **Examples** section for how to do this.

## References

Gabry, J., Simpson, D., Vehtari, A., Betancourt, M., Gelman, A. (2017). Visualization in Bayesian workflow. arXiv preprint [arxiv:1709.01449](https://arxiv.org/abs/1709.01449).

## See Also

Ari Hartikainen's post about the parallel coordinates plot on the Stan Forums ([link](#)).

Other MCMC: [MCMC-combos](#), [MCMC-diagnostics](#), [MCMC-distributions](#), [MCMC-intervals](#), [MCMC-nuts](#), [MCMC-overview](#), [MCMC-recover](#), [MCMC-scatterplots](#), [MCMC-traces](#)

## Examples

```
color_scheme_set("pink")
x <- example_mcmc_draws(params = 5)
mcmc_parcoord(x)
mcmc_parcoord(x, regex_pars = "beta")

## Not run:
# Example using a Stan demo model
library(rstan)
fit <- stan_demo("eight_schools")
draws <- as.array(fit, pars = c("mu", "tau", "theta", "lp_"))
np <- nuts_params(fit)
str(np)
levels(np$Parameter)

color_scheme_set("brightblue")
mcmc_parcoord(draws, alpha = 0.05)
mcmc_parcoord(draws, np = np)

# customize appearance of divergences
color_scheme_set("darkgray")
div_style <- parcoord_style_np(div_color = "green", div_size = 0.05, div_alpha = 0.4)
mcmc_parcoord(draws, size = 0.25, alpha = 0.1,
              np = np, np_style = div_style)

# to use a transformation (e.g., to standarde all the variables)
# specify the 'transformations' argument (though partial argument name
# matching means we can just use 'trans' or 'transform')
mcmc_parcoord(
  draws,
  transform = function(x) {(x - mean(x)) / sd(x)},
  size = 0.25,
  alpha = 0.1,
  np = np,
  np_style = div_style
)
```



```
# mcmc_parcoord_data returns just the data in a convenient form for plotting
d <- mcmc_parcoord_data(x, np = np)
head(d)
tail(d)

## End(Not run)
```

---

MCMC-recover

*Compare MCMC estimates to "true" parameter values*


---

## Description

Plots comparing MCMC estimates to "true" parameter values. Before fitting a model to real data it is useful to simulate data according to the model using known (fixed) parameter values and to check that these "true" parameter values are (approximately) recovered by fitting the model to the simulated data. See the **Plot Descriptions** section, below, for details on the available plots.

## Usage

```
mcmc_recover_intervals(x, true, batch = rep(1, length(true)),
  facet_args = list(), ..., prob = 0.5, prob_outer = 0.9,
  point_est = c("median", "mean", "none"), size = 4, alpha = 1)

mcmc_recover_scatter(x, true, batch = rep(1, length(true)),
  facet_args = list(), ..., point_est = c("median", "mean"), size = 3,
  alpha = 1)

mcmc_recover_hist(x, true, facet_args = list(), ..., binwidth = NULL)
```

## Arguments

- |       |   |
|-------|---|
| x     | A 3-D array, matrix, list of matrices, or data frame of MCMC draws. The <a href="#">MCMC-overview</a> page provides details on how to specify each these allowed inputs.  |
| true  | A numeric vector of "true" values of the parameters in x. There should be one value in true for each parameter included in x and the order of the parameters in true should be the same as the order of the parameters in x.  |
| batch | Optionally, a vector-like object (numeric, character, integer, factor) used to split the parameters into batches. If batch is specified, it must have the same length as true and be in the same order as true. Parameters in the same batch will be grouped together in the same facet in the plot (see the <b>Examples</b> section, below). The default is to group all parameters together into a single batch. Changing the default is most useful when parameters are on very different scales, in which case batch can be used to group them into batches within which it makes sense to use the same <i>y</i> -axis. |

facet_args	A named list of arguments (other than facets) passed to <a href="#">facet_wrap</a> or <a href="#">facet_grid</a> to control faceting.
...	Currently unused.
prob	The probability mass to include in the inner interval. The default is 0.5 (50% interval).
prob_outer	The probability mass to include in the outer interval. The default is 0.9 (90% interval).
point_est	The point estimate to show. Either "median" (the default), "mean", or "none".
size, alpha	Passed to <a href="#">geom_point</a> to control the appearance of plotted points.
binwidth	An optional value used as the binwidth argument to <a href="#">geom_histogram</a> to override the default binwidth.

### Value

A ggplot object that can be further customized using the **ggplot2** package.

### Plot Descriptions

`mcmc_recover_intervals` Central intervals and point estimates computed from MCMC draws, with "true" values plotted using a different shape.

`mcmc_recover_scatter` Scatterplot of posterior means (or medians) against "true" values.

`mcmc_recover_hist` Histograms of the draws for each parameter with the "true" value overlaid as a vertical line.

### See Also

Other MCMC: [MCMC-combos](#), [MCMC-diagnostics](#), [MCMC-distributions](#), [MCMC-intervals](#), [MCMC-nuts](#), [MCMC-overview](#), [MCMC-parcoord](#), [MCMC-scatterplots](#), [MCMC-traces](#)

### Examples

```
## Not run:
library(rstanarm)
alpha <- 1; beta <- rnorm(10, 0, 3); sigma <- 2
X <- matrix(rnorm(1000), 100, 10)
y <- rnorm(100, mean = c(alpha + X %*% beta), sd = sigma)
fit <- stan_glm(y ~ ., data = data.frame(y, X))
draws <- as.matrix(fit)
print(colnames(draws))
true <- c(alpha, beta, sigma)

mcmc_recover_intervals(draws, true)

# put the coefficients on X into the same batch
mcmc_recover_intervals(draws, true, batch = c(1, rep(2, 10), 1))
# equivalent
mcmc_recover_intervals(draws, true, batch = grepl("X", colnames(draws)))
# same but facets stacked vertically
```

```

mcmc_recover_intervals(draws, true,
                       batch = grepl("X", colnames(draws)),
                       facet_args = list(ncol = 1),
                       size = 3)

# each parameter in its own facet
mcmc_recover_intervals(draws, true, batch = 1:ncol(draws))
# same but in a different order
mcmc_recover_intervals(draws, true, batch = c(1, 3, 4, 2, 5:12))
# present as bias by centering with true values
mcmc_recover_intervals(sweep(draws, 2, true), rep(0, ncol(draws))) + hline_0()

# scatterplot of posterior means vs true values
mcmc_recover_scatter(draws, true, point_est = "mean")

# histograms of parameter draws with true value added as vertical line
color_scheme_set("brightblue")
mcmc_recover_hist(draws[, 1:4], true[1:4])

## End(Not run)

```

---

MCMC-scatterplots      *Scatterplots of MCMC draws*

---

## Description

Scatterplots, hexagonal heatmaps, and pairs plots from MCMC draws. See the **Plot Descriptions** section, below, for details.

## Usage

```

mcmc_scatter(x, pars = character(), regex_pars = character(),
             transformations = list(), ..., size = 2.5, alpha = 0.8, np = NULL,
             np_style = scatter_style_np())

mcmc_hex(x, pars = character(), regex_pars = character(),
         transformations = list(), ..., binwidth = NULL)

mcmc_pairs(x, pars = character(), regex_pars = character(),
           transformations = list(), ..., diag_fun = c("hist", "dens"),
           off_diag_fun = c("scatter", "hex"), diag_args = list(),
           off_diag_args = list(), condition = pairs_condition(), lp = NULL,
           np = NULL, np_style = pairs_style_np(), max_treedepth = NULL)

scatter_style_np(div_color = "red", div_shape = 16, div_size = 2.5,
                 div_alpha = 1)

```

```
pairs_style_np(div_color = "red", div_shape = 4, div_size = 1,
              div_alpha = 1, td_color = "yellow2", td_shape = 3, td_size = 1,
              td_alpha = 1)
```

```
pairs_condition(chains = NULL, draws = NULL, nuts = NULL)
```

## Arguments

x	A 3-D array, matrix, list of matrices, or data frame of MCMC draws. The <a href="#">MCMC-overview</a> page provides details on how to specify each these allowed inputs.
pars	An optional character vector of parameter names. For <code>mcmc_scatter</code> and <code>mcmc_hex</code> only two parameters can be selected. To plot more than two parameters use <code>mcmc_pairs</code> .
regex_pars	An optional <a href="#">regular expression</a> to use for parameter selection. Can be specified instead of <code>pars</code> or in addition to <code>pars</code> .
transformations	Optionally, transformations to apply to parameters before plotting. If <code>transformations</code> is a function or a single string naming a function then that function will be used to transform all parameters. To apply transformations to particular parameters, the <code>transformations</code> argument can be a named list with length equal to the number of parameters to be transformed. Currently only univariate transformations of scalar parameters can be specified (multivariate transformations will be implemented in a future release). If <code>transformations</code> is a list, the name of each list element should be a parameter name and the content of each list element should be a function (or any item to match as a function via <code>match.fun</code> , e.g. a string naming a function). If a function is specified by its name as a string (e.g. "log"), then it can be used to construct a new parameter label for the appropriate parameter (e.g. "log(sigma)"). If a function itself is specified (e.g. <code>log</code> or <code>function(x) log(x)</code> ) then "t" is used in the new parameter label to indicate that the parameter is transformed (e.g. "t(sigma)").
...	Currently ignored.
size, alpha	For <code>mcmc_scatter</code> , passed to <a href="#">geom_point</a> to control the appearance of the points.
np	Optionally, a data frame of NUTS sampler parameters, either created by <code>nuts_params</code> or in the same form as the object returned by <code>nuts_params</code> . The colors, shapes, and sizes of the superimposed points can be customized using the <code>np_style</code> argument.
np_style	If <code>np</code> is specified, <code>np_style</code> can be a call to the <code>scatter_style_np</code> helper function (for <code>mcmc_scatter</code> ) or the <code>pairs_style_np</code> helper function (for <code>mcmc_pars</code> ) to specify arguments controlling the appearance of superimposed points representing NUTS diagnostic information. (Note: for <code>pairs_style_np</code> the size arguments are interpreted as scaling factors).
binwidth	For <code>mcmc_hex</code> , an optional numeric vector of <i>length two</i> passed to <a href="#">geom_hex</a> to override the default binwidth in both the vertical and horizontal directions.

diag_fun, off_diag_fun	For <code>mcmc_pairs</code> , the plotting function to use for the plots along the diagonal and for the off-diagonal plots, respectively. Currently <code>diag_fun</code> can be "hist" for histogram or "dens" for density, and <code>off_diag_fun</code> can be "scatter" for scatterplot or "hex" for a hexagonal heatmap.
diag_args, off_diag_args	For <code>mcmc_pairs</code> , optional named lists of arguments to pass to the functions implied by the <code>diag_fun</code> and <code>off_diag_fun</code> arguments, respectively. For example, if <code>off_diag_fun</code> is "scatter" then <code>off_diag_args</code> could include optional arguments to <code>mcmc_scatter</code> like <code>size</code> and <code>alpha</code> .
condition	For <code>mcmc_pairs</code> , a call to the <code>pairs_condition</code> helper function, which is used to specify a criterion for determining which chains (or iterations) are shown in the plots above the diagonal and which are shown in the plots below the diagonal. The histograms (or density plots) along the diagonal are always made using all chains and iterations, but the scatterplots (or hex plots) above and below the diagonal show different combinations of chains/iterations depending on <code>condition</code> . The default is a call to <code>pairs_condition</code> with none of its arguments specified. In this case half of the chains (or roughly half if there are an odd number) will be used in the plots above the diagonal and the rest in the plots below the diagonal. The <code>chains</code> , <code>draws</code> , and <code>nuts</code> arguments to <code>pairs_condition</code> , which are documented below, can be used to change this default.
lp	For <code>mcmc_pairs</code> , a molten data frame of draws of the log-posterior or, more commonly, of a quantity equal to the log-posterior up to a constant. <code>lp</code> should either be created via <code>log_posterior</code> or be an object with the same form as the object returned by <code>log_posterior</code> .
max_treedepth	For <code>mcmc_pairs</code> , an integer representing the maximum treedepth allowed when fitting the model (if fit using NUTS). This is only needed for detecting which transitions (if any) hit the maximum treedepth.
div_color, div_shape, div_size, div_alpha, td_color, td_shape, td_size, td_alpha	Optional arguments to the <code>scatter_style_np</code> or <code>pairs_style_np</code> helper functions that are eventually passed to <code>geom_point</code> . The default values are displayed in the <b>Usage</b> section above.
chains, draws, nuts	Optional arguments to the <code>pairs_condition</code> helper function, which is used to specify the <code>condition</code> argument for <code>mcmc_pairs</code> . <ul style="list-style-type: none"> <li>• The <code>chains</code> argument can be used to select some subset of the chains. If <code>chains</code> is an integer vector then the behavior is the same as the default (half the chains above the diagonal and half below) except using only the specified subset of chains. Alternatively, <code>chains</code> can be a list of two integer vectors with the first specifying the chains to be shown in the plots above the diagonal and the second for below the diagonal.</li> <li>• The <code>draws</code> argument to <code>pairs_condition</code> can be used to directly specify which realizations are plotted above and below the diagonal. <code>draws</code> can be a single proportion, which is interpreted as the proportion of realizations (among all chains) to plot in the lower panel starting with the first realization in each chain, with the complement (from the end of each chain)</li> </ul>

plotted in the upper panel. Alternatively draws can be a logical vector with length equal to the product of the number of iterations and the number of chains, in which case realizations corresponding to FALSE and TRUE will be plotted in the lower and upper panels, respectively.

- For models fit using NUTS, the `nuts` argument to `pairs_condition` can be used. It takes a (possibly abbreviated) string to select among `"accept_stat__"`, `"stepsize__"`, `"treedepth__"`, `"n_leapfrog__"`, `"divergent__"`, `"energy__"`, and `"lp__"`. These are the sampler parameters associated with NUTS (and `"lp__"` is the log-posterior up to an additive constant). In this case, plots below the diagonal will contain realizations that are below the median of the indicated variable (or are zero in the case of `"divergent__"`), and plots above the diagonal will contain realizations that are greater than or equal to the median of the indicated variable (or are one in the case of `"divergent__"`). If `"lp__"` is used then the `lp` argument to `mcmc_pairs` must also be specified. For the other NUTS parameters the `np` argument to `mcmc_pairs` must also be specified.

## Value

`mcmc_scatter` and `mcmc_hex` return a `ggplot` object that can be further customized using the **ggplot2** package.

`mcmc_pairs` returns many `ggplot` objects organized into a grid via `bayesplot_grid`.

## Plot Descriptions

`mcmc_scatter` Bivariate scatterplot of posterior draws. If using a very large number of posterior draws then `mcmc_hex` may be preferable to avoid overplotting. For models fit using NUTS the `np`, and `np_style` arguments can be used to add additional information in the plot (in this case the approximate location of divergences).

`mcmc_hex` Hexagonal heatmap of 2-D bin counts. This plot is useful in cases where the posterior sample size is large enough that `mcmc_scatter` suffers from overplotting.

`mcmc_pairs` A square plot matrix with univariate marginal distributions along the diagonal (as histograms or kernel density plots) and bivariate distributions off the diagonal (as scatterplots or hex heatmaps).

For the off-diagonal plots, the default is to split the chains so that (roughly) half are displayed above the diagonal and half are below (all chains are always merged together for the plots along the diagonal). Other possibilities are available by setting the `condition` argument.

Additionally, extra diagnostic information for models fit using NUTS can be added to the `pairs` plot using the `lp`, `np`, and `np_style` arguments. If `np` is specified (and `condition` is *not* `"divergent__"`), then points (red, by default) will be superimposed onto the off-diagonal plots indicating which (if any) iterations encountered a divergent transition. Also, if both `np` and `max_treedepth` are specified then points (yellow, by default) will be superimposed to indicate a transition that hit the maximum treedepth rather than terminated its evolution normally. The `np_style` argument can be used with the `pairs_style_np` convenience function to change the appearance of these overlaid points. See the **Examples** section.

## References

Gabry, J., Simpson, D., Vehtari, A., Betancourt, M., Gelman, A. (2017). Visualization in Bayesian workflow. arXiv preprint [arxiv:1709.01449](https://arxiv.org/abs/1709.01449).

## See Also

Other MCMC: [MCMC-combos](#), [MCMC-diagnostics](#), [MCMC-distributions](#), [MCMC-intervals](#), [MCMC-nuts](#), [MCMC-overview](#), [MCMC-parcoord](#), [MCMC-recover](#), [MCMC-traces](#)

## Examples

```
library("ggplot2")

# some parameter draws to use for demonstration
x <- example_mcmc_draws(params = 6)
dimnames(x)

# scatterplot of alpha vs log(sigma)
color_scheme_set("teal")
(p <- mcmc_scatter(x, pars = c("alpha", "sigma"),
                  transform = list(sigma = "log")))

p +
  labs(
    title = "Insert your own headline-grabbing title",
    subtitle = "with a provocative subtitle",
    caption = "and a controversial caption",
    x = expression(alpha),
    y = expression(log(sigma))
  )

# add ellipse
p + stat_ellipse(level = 0.9, color = "gray20", size = 1)

# add contour
color_scheme_set("red")
p2 <- mcmc_scatter(x, pars = c("alpha", "sigma"), size = 3.5, alpha = 0.25)
p2 + stat_density_2d(color = "black", size = .5)

# can also add lines/smooths
color_scheme_set("pink")
(p3 <- mcmc_scatter(x, pars = c("alpha", "beta[3]"), alpha = 0.25, size = 3))
p3 + geom_smooth(method = "lm", se = FALSE, color = "gray20",
                 size = .75, linetype = 2)

# hexagonal heatmap
color_scheme_set("brightblue")
(p <- mcmc_hex(x, pars = c("sigma", "alpha"), transform = list(sigma = "log")))
p + plot_bg(fill = "gray95")
p + plot_bg(fill = "gray95") + panel_bg(fill = "gray70")
```

```

color_scheme_set("purple")

# pairs plots
# default of condition=NULL implies splitting chains between upper and lower panels
mcmc_pairs(x, pars = "alpha", regex_pars = "beta\\[[1,4]\\]",
           off_diag_args = list(size = 1, alpha = 0.5))

# change to density plots instead of histograms and hex plots instead of
# scatterplots
mcmc_pairs(x, pars = "alpha", regex_pars = "beta\\[[1,4]\\]",
           diag_fun = "dens", off_diag_fun = "hex")

# plot chain 1 above diagonal and chains 2, 3, and 4 below
color_scheme_set("brightblue")
mcmc_pairs(x, pars = "alpha", regex_pars = "beta\\[[1,4]\\]",
           diag_fun = "dens", off_diag_fun = "hex",
           condition = pairs_condition(chains = list(1, 2:4)))

## Not run:
### Adding NUTS diagnostics to scatterplots and pairs plots

# examples using rstanarm package
library(rstanarm)

# for demonstration purposes, intentionally fit a model that
# will (almost certainly) have some divergences
fit <- stan_glm(
  mpg ~ ., data = mtcars,
  iter = 1000,
  # this combo of prior and adapt_delta should lead to some divergences
  prior = hs(),
  adapt_delta = 0.9
)
posterior <- as.array(fit)
np <- nuts_params(fit)

# mcmc_scatter with divergences highlighted
color_scheme_set("brightblue")
mcmc_scatter(posterior, pars = c("wt", "sigma"), np = np)

color_scheme_set("darkgray")
div_style <- scatter_style_np(div_color = "green", div_shape = 4, div_size = 4)
mcmc_scatter(posterior, pars = c("sigma", "(Intercept)"),
            np = np, np_style = div_style)

# split the draws according to above/below median accept_stat__ and
# show approximate location of divergences (red points)
color_scheme_set("brightblue")
mcmc_pairs(
  posterior,
  pars = c("wt", "cyl", "sigma"),
  off_diag_args = list(size = 1, alpha = 1/3),

```



```

    condition = pairs_condition(nuts = "accept_stat__"),
    np = np
  )

# more customizations:
# - transform sigma to log(sigma)
# - median log-posterior as 'condition'
# - hex instead of scatter for off-diagonal plots
# - show points where max treedepth hit in blue
color_scheme_set("darkgray")
mcmc_pairs(
  posterior,
  pars = c("wt", "cyl", "sigma"),
  transform = list(sigma = "log"),
  off_diag_fun = "hex",
  condition = pairs_condition(nuts = "lp__"),
  lp = log_posterior(fit),
  np = np,
  np_style = pairs_style_np(div_color = "firebrick",
                             td_color = "blue",
                             td_size = 2),
  # for demonstration purposes, set max_treedepth to a value that will
  # result in at least a few max treedepth warnings
  max_treedepth = with(np, -1 + max(Value[Parameter == "treedepth__"]))
)

## End(Not run)

```

---

MCMC-traces

*Trace plot (time series plot) of MCMC draws*


---

## Description

Trace plot (or traceplot) of MCMC draws. See the **Plot Descriptions** section, below, for details.

## Usage

```

mcmc_trace(x, pars = character(), regex_pars = character(),
  transformations = list(), facet_args = list(), ..., n_warmup = 0,
  window = NULL, size = NULL, np = NULL, np_style = trace_style_np(),
  divergences = NULL)

```

```

mcmc_trace_highlight(x, pars = character(), regex_pars = character(),
  transformations = list(), facet_args = list(), ..., n_warmup = 0,
  window = NULL, size = NULL, alpha = 0.2, highlight = 1)

```

```

trace_style_np(div_color = "red", div_size = 0.25, div_alpha = 1)

```

**Arguments**

x	A 3-D array, matrix, list of matrices, or data frame of MCMC draws. The <a href="#">MCMC-overview</a> page provides details on how to specify each these allowed inputs.
pars	An optional character vector of parameter names. If neither pars nor regex_pars is specified then the default is to use <i>all</i> parameters.
regex_pars	An optional <a href="#">regular expression</a> to use for parameter selection. Can be specified instead of pars or in addition to pars.
transformations	Optionally, transformations to apply to parameters before plotting. If transformations is a function or a single string naming a function then that function will be used to transform all parameters. To apply transformations to particular parameters, the transformations argument can be a named list with length equal to the number of parameters to be transformed. Currently only univariate transformations of scalar parameters can be specified (multivariate transformations will be implemented in a future release). If transformations is a list, the name of each list element should be a parameter name and the content of each list element should be a function (or any item to match as a function via <a href="#">match.fun</a> , e.g. a string naming a function). If a function is specified by its name as a string (e.g. "log"), then it can be used to construct a new parameter label for the appropriate parameter (e.g. "log(sigma)"). If a function itself is specified (e.g. log or function(x) log(x)) then "t" is used in the new parameter label to indicate that the parameter is transformed (e.g. "t(sigma)").
facet_args	A named list of arguments (other than facets) passed to <a href="#">facet_wrap</a> or <a href="#">facet_grid</a> to control faceting.
...	Currently ignored.
n_warmup	An integer; the number of warmup iterations included in x. The default is n_warmup = 0, i.e. to assume no warmup iterations are included. If n_warmup > 0 then the background for iterations 1:n_warmup is shaded gray.
window	An integer vector of length two specifying the limits of a range of iterations to display.
size	An optional value to override the default line size (mcmc_trace) or the default point size (mcmc_trace_highlight).
np	For models fit using <a href="#">NUTS</a> (more generally, any <a href="#">symplectic integrator</a> ), an optional data frame providing NUTS diagnostic information. The data frame should be the object returned by <a href="#">nuts_params</a> or one with the same structure. If np is specified then tick marks are added to the bottom of the trace plot indicating within which iterations there was a divergence (if there were any). See the end of the <b>Examples</b> section, below.
np_style	A call to the trace_style_np helper function to specify arguments controlling the appearance of tick marks representing divergences (if the np argument is specified).
divergences	Deprecated. Use the np argument instead.
alpha	For mcmc_trace_highlight, passed to <a href="#">geom_point</a> to control the transparency of the points for the chains not highlighted.

`highlight` For `mcmc_trace_highlight`, an integer specifying one of the chains that will be more visible than the others in the plot.

`div_color`, `div_size`, `div_alpha` Optional arguments to the `trace_style_np` helper function that are eventually passed to `geom_rug` if the `np` argument is also specified. They control the color, size, and transparency specifications for showing divergences in the plot. The default values are displayed in the **Usage** section above.

## Value

A `ggplot` object that can be further customized using the **ggplot2** package.

## Plot Descriptions

`mcmc_trace` Standard trace plots of MCMC draws. For models fit using [NUTS](#), the `np` argument can be used to also show divergences on the trace plot.

`mcmc_trace_highlight` Traces are plotted using points rather than lines and the opacity of all chains but one (specified by the `highlight` argument) is reduced.

## See Also

Other MCMC: [MCMC-combos](#), [MCMC-diagnostics](#), [MCMC-distributions](#), [MCMC-intervals](#), [MCMC-nuts](#), [MCMC-overview](#), [MCMC-parcoord](#), [MCMC-recover](#), [MCMC-scatterplots](#)

## Examples

```
# some parameter draws to use for demonstration
x <- example_mcmc_draws(chains = 4, params = 6)
dim(x)
dimnames(x)

# trace plots of the betas
color_scheme_set("viridis")
mcmc_trace(x, regex_pars = "beta")

color_scheme_set("viridisA")
mcmc_trace(x, regex_pars = "beta")

color_scheme_set("viridisC")
mcmc_trace(x, regex_pars = "beta")

# mix color schemes
color_scheme_set("mix-blue-red")
mcmc_trace(x, regex_pars = "beta")

# use traditional ggplot discrete color scale
mcmc_trace(x, pars = c("alpha", "sigma")) +
  ggplot2::scale_color_discrete()

# zoom in on a window of iterations, increase line size,
```

```

# add tick marks, move legend to the top, add gray background
color_scheme_set("viridisA")
mcmc_trace(x[, 1:4], window = c(100, 130), size = 1) +
  panel_bg(fill = "gray90", color = NA) +
  legend_move("top")

## Not run:
# parse facet label text
color_scheme_set("purple")
p <- mcmc_trace(
  x,
  regex_pars = "beta\\[[1,3]\\]",
  facet_args = list(labeler = ggplot2::label_parsed)
)
p + facet_text(size = 15)

# mark first 100 draws as warmup
mcmc_trace(x, n_warmup = 100)

# plot as points, highlighting chain 2
color_scheme_set("brightblue")
mcmc_trace_highlight(x, pars = "sigma", highlight = 2, size = 2)

# for models fit using HMC/NUTS divergences can be displayed in the trace plot
library("rstanarm")
fit <- stan_glm(mpg ~ ., data = mtcars,
  # next line to keep example fast and also ensure we get some divergences
  prior = hs(), iter = 400, adapt_delta = 0.8)

# extract draws using as.array (instead of as.matrix) to keep
# chains separate for trace plot
posterior <- as.array(fit)

# for stanfit and stanreg objects use nuts_params() to get the divergences
mcmc_trace(posterior, pars = "sigma", np = nuts_params(fit))

color_scheme_set("viridis")
mcmc_trace(
  posterior,
  pars = c("wt", "sigma"),
  size = 0.5,
  facet_args = list(nrow = 2),
  np = nuts_params(fit),
  np_style = trace_style_np(div_color = "black", div_size = 0.5)
)

color_scheme_set("viridis")
mcmc_trace(
  posterior,
  pars = c("wt", "sigma"),
  size = 0.8,
  facet_args = list(nrow = 2),
  divergences = nuts_params(fit),

```

```

    div_color = "black"
  )

  ## End(Not run)

```

PPC-discrete

*PPCs for discrete outcomes***Description**

This page documents the PPC functions that can only be used if  $y$  and  $yrep$  are discrete. Currently these include rootograms for count outcomes and bar plots for ordinal, categorical, and multinomial outcomes. See the **Plot Descriptions** section below.

**Usage**

```

ppc_bars(y, yrep, ..., prob = 0.9, width = 0.9, size = 1, fatten = 3,
         freq = TRUE)

ppc_bars_grouped(y, yrep, group, facet_args = list(), ..., prob = 0.9,
                 width = 0.9, size = 1, fatten = 3, freq = TRUE)

ppc_rootogram(y, yrep, style = c("standing", "hanging", "suspended"), ...,
              prob = 0.9, size = 1)

```

**Arguments**

$y$	A vector of observations. See <b>Details</b> .
$yrep$	An $S$ by $N$ matrix of draws from the posterior predictive distribution, where $S$ is the size of the posterior sample (or subset of the posterior sample used to generate $yrep$ ) and $N$ is the number of observations (the length of $y$ ). The columns of $yrep$ should be in the same order as the data points in $y$ for the plots to make sense. See <b>Details</b> for additional instructions.
$\dots$	Currently unused.
$prob$	A value between 0 and 1 indicating the desired probability mass to include in the $yrep$ intervals. Set $prob=0$ to remove the intervals. For <code>ppc_rootogram</code> these are intervals of the <i>square roots</i> of the expected counts.
$width$	For <code>ppc_bars</code> and <code>ppc_bars_grouped</code> , passed to <code>geom_bar</code> to control the bar width.
$size, fatten$	For <code>ppc_bars</code> and <code>ppc_bars_grouped</code> , $size$ and $fatten$ are passed to <code>geom_pointrange</code> to control the appearance of the $yrep$ points and intervals. For <code>ppc_rootogram</code> $size$ is passed to <code>geom_line</code> .
$freq$	For <code>ppc_bars</code> and <code>ppc_bars_grouped</code> , if TRUE (the default) the y-axis will display counts. Setting $freq=FALSE$ will put proportions on the y-axis.

group	A grouping variable (a vector or factor) the same length as <code>y</code> . Each value in group is interpreted as the group level pertaining to the corresponding value of <code>y</code> .
facet_args	An optional list of arguments (other than facets) passed to <code>facet_wrap</code> to control faceting.
style	For <code>ppc_rootogram</code> , a string specifying the rootogram style. The options are "standing", "hanging", and "suspended". See the <b>Plot Descriptions</b> section, below, for details on the different styles.

### Details

For all of these plots `y` and `yrep` must be non-negative integers, although they need not be integers in the strict sense of R's `integer` type.

### Value

A `ggplot` object that can be further customized using the **ggplot2** package.

### Plot Descriptions

`ppc_bars` Bar plot of `y` with `yrep` medians and uncertainty intervals superimposed on the bars.

`ppc_bars_grouped` Same as `ppc_bars` but a separate plot (facet) is generated for each level of a grouping variable.

`ppc_rootogram` Rootograms allow for diagnosing problems in count data models such as overdispersion or excess zeros. They consist of a histogram of `y` with the expected counts based on `yrep` overlaid as a line along with uncertainty intervals. The y-axis represents the square roots of the counts to approximately adjust for scale differences and thus ease comparison between observed and expected counts. Using the `style` argument, the histogram style can be adjusted to focus on different aspects of the data:

- *Standing*: basic histogram of observed counts with curve showing expected counts.
- *Hanging*: observed counts counts hanging from the curve representing expected counts.
- *Suspended*: histogram of the differences between expected and observed counts.

**All of these are plotted on the square root scale.** See Kleiber and Zeileis (2016) for advice on interpreting rootograms and selecting among the different styles.

### References

Kleiber, C. and Zeileis, A. (2016). Visualizing count data regressions using rootograms. *The American Statistician*. 70(3): 296–303. <https://arxiv.org/abs/1605.01311>.

### See Also

Other PPCs: [PPC-distributions](#), [PPC-errors](#), [PPC-intervals](#), [PPC-loo](#), [PPC-overview](#), [PPC-scatterplots](#), [PPC-test-statistics](#)

**Examples**

```

# bar plots
f <- function(N) {
  sample(1:4, size = N, replace = TRUE, prob = c(0.25, 0.4, 0.1, 0.25))
}
y <- f(100)
yrep <- t(replicate(500, f(100)))
dim(yrep)
group <- gl(2, 50, length = 100, labels = c("GroupA", "GroupB"))

color_scheme_set("mix-pink-blue")
ppc_bars(y, yrep)

# split by group, change interval width, and display proportion
# instead of count on y-axis
color_scheme_set("mix-blue-pink")
ppc_bars_grouped(y, yrep, group, prob = 0.5, freq = FALSE)

# rootograms for counts
y <- rpois(100, 20)
yrep <- matrix(rpois(10000, 20), ncol = 100)

color_scheme_set("brightblue")
ppc_rootogram(y, yrep)
ppc_rootogram(y, yrep, prob = 0)

ppc_rootogram(y, yrep, style = "hanging", prob = 0.8)
ppc_rootogram(y, yrep, style = "suspended")

```

---

PPC-distributions      *PPC distributions*

---

**Description**

Compare the empirical distribution of the data `y` to the distributions of simulated/replicated data `yrep` from the posterior predictive distribution. See the **Plot Descriptions** section, below, for details.

**Usage**

```

ppc_hist(y, yrep, ..., binwidth = NULL, freq = TRUE)

ppc_boxplot(y, yrep, ..., notch = TRUE, size = 0.5, alpha = 1)

ppc_freqpoly(y, yrep, ..., binwidth = NULL, freq = TRUE, size = 0.25,
  alpha = 1)

ppc_freqpoly_grouped(y, yrep, group, ..., binwidth = NULL, freq = TRUE,

```

```

size = 0.25, alpha = 1)

ppc_dens(y, yrep, ..., trim = FALSE, size = 0.5, alpha = 1)

ppc_dens_overlay(y, yrep, ..., trim = FALSE, size = 0.25, alpha = 0.7)

ppc_ecdf_overlay(y, yrep, ..., pad = TRUE, size = 0.25, alpha = 0.7)

ppc_violin_grouped(y, yrep, group, ..., probs = c(0.1, 0.5, 0.9), size = 1,
  alpha = 1, y_draw = c("violin", "points", "both"), y_size = 1,
  y_alpha = 1, y_jitter = 0.1)

```

### Arguments

<code>y</code>	A vector of observations. See <b>Details</b> .
<code>yrep</code>	An $S$ by $N$ matrix of draws from the posterior predictive distribution, where $S$ is the size of the posterior sample (or subset of the posterior sample used to generate <code>yrep</code> ) and $N$ is the number of observations (the length of <code>y</code> ). The columns of <code>yrep</code> should be in the same order as the data points in <code>y</code> for the plots to make sense. See <b>Details</b> for additional instructions.
<code>...</code>	Currently unused.
<code>binwidth</code>	An optional value used as the <code>binwidth</code> argument to <code>geom_histogram</code> to override the default binwidth.
<code>freq</code>	For histograms, <code>freq=TRUE</code> (the default) puts count on the y-axis. Setting <code>freq=FALSE</code> puts density on the y-axis. (For many plots the y-axis text is off by default. To view the count or density labels on the y-axis see the <code>yaxis_text</code> convenience function.)
<code>notch</code>	A logical scalar passed to <code>geom_boxplot</code> . Unlike for <code>geom_boxplot</code> , the default is <code>notch=TRUE</code> .
<code>size, alpha</code>	Passed to the appropriate geom to control the appearance of the <code>yrep</code> distributions.
<code>group</code>	A grouping variable (a vector or factor) the same length as <code>y</code> . Each value in <code>group</code> is interpreted as the group level pertaining to the corresponding value of <code>y</code> .
<code>trim</code>	A logical scalar passed to <code>geom_density</code> .
<code>pad</code>	A logical scalar passed to <code>stat_ecdf</code> .
<code>probs</code>	A numeric vector passed to <code>geom_violin</code> 's <code>draw_quantiles</code> argument to specify at which quantiles to draw horizontal lines. Set to <code>NULL</code> to remove the lines.
<code>y_draw</code>	For <code>ppc_violin_grouped</code> , a string specifying how to draw <code>y</code> : "violin" (default), "points" (jittered points), or "both".
<code>y_jitter, y_size, y_alpha</code>	For <code>ppc_violin_grouped</code> , if <code>y_draw</code> is "points" or "both" then <code>y_size</code> , <code>y_alpha</code> , and <code>y_jitter</code> are passed to to the <code>size</code> , <code>alpha</code> , and <code>width</code> arguments of <code>geom_jitter</code> to control the appearance of <code>y</code> points. The default of <code>y_jitter=NULL</code> will let <b>ggplot2</b> determine the amount of jitter.



**Details**

For Binomial data, the plots will typically be most useful if `y` and `yrep` contain the "success" proportions (not discrete "success" or "failure" counts).

**Value**

A ggplot object that can be further customized using the **ggplot2** package.

**Plot Descriptions**

`ppc_hist`, `ppc_freqpoly`, `ppc_dens`, `ppc_boxplot` A separate histogram, shaded frequency polygon, smoothed kernel density estimate, or box and whiskers plot is displayed for `y` and each dataset (row) in `yrep`. For these plots `yrep` should therefore contain only a small number of rows. See the **Examples** section.

`ppc_freqpoly_grouped` A separate frequency polygon is plotted for each level of a grouping variable for `y` and each dataset (row) in `yrep`. For this plot `yrep` should therefore contain only a small number of rows. See the **Examples** section.

`ppc_dens_overlay`, `ppc_ecdf_overlay` Kernel density or empirical CDF estimates of each dataset (row) in `yrep` are overlaid, with the distribution of `y` itself on top (and in a darker shade).

`ppc_violin_grouped` The density estimate of `yrep` within each level of a grouping variable is plotted as a violin with horizontal lines at notable quantiles. `y` is overlaid on the plot either as a violin, points, or both, depending on the `y_draw` argument.

**References**

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. (Ch. 6)

**See Also**

Other PPCs: [PPC-discrete](#), [PPC-errors](#), [PPC-intervals](#), [PPC-loo](#), [PPC-overview](#), [PPC-scatterplots](#), [PPC-test-statistics](#)

**Examples**

```
color_scheme_set("brightblue")
y <- example_y_data()
yrep <- example_yrep_draws()
dim(yrep)
ppc_dens_overlay(y, yrep[1:25, ])

ppc_ecdf_overlay(y, yrep[sample(nrow(yrep), 25), ])

# for ppc_hist,dens,freqpoly,boxplot definitely use a subset yrep rows so
# only a few (instead of nrow(yrep)) histograms are plotted
ppc_hist(y, yrep[1:8, ])
```

```

color_scheme_set("red")
ppc_boxplot(y, yrep[1:8, ])

# wizard hat plot
color_scheme_set("blue")
ppc_dens(y, yrep[200:202, ])

ppc_freqpoly(y, yrep[1:3,], alpha = 0.1, size = 1, binwidth = 5)

# if groups are different sizes then the 'freq' argument can be useful
group <- example_group_data()
ppc_freqpoly_grouped(y, yrep[1:3,], group) + yaxis_text()

ppc_freqpoly_grouped(y, yrep[1:3,], group, freq = FALSE) + yaxis_text()

# don't need to only use small number of rows for ppc_violin_grouped
# (as it pools yrep draws within groups)
color_scheme_set("gray")
ppc_violin_grouped(y, yrep, group, size = 1.5)

ppc_violin_grouped(y, yrep, group, alpha = 0)

# change how y is drawn
ppc_violin_grouped(y, yrep, group, alpha = 0, y_draw = "points", y_size = 1.5)
ppc_violin_grouped(y, yrep, group, alpha = 0, y_draw = "both",
                    y_size = 1.5, y_alpha = 0.5, y_jitter = 0.33)

```

---

PPC-errors

*PPC errors*


---

## Description

Various plots of predictive errors  $y - \hat{y}$ . See the **Details** and **Plot Descriptions** sections, below.

## Usage

```

ppc_error_hist(y, yrep, ..., binwidth = NULL, freq = TRUE)

ppc_error_hist_grouped(y, yrep, group, ..., binwidth = NULL, freq = TRUE)

ppc_error_scatter(y, yrep, ..., size = 2.5, alpha = 0.8)

ppc_error_scatter_avg(y, yrep, ..., size = 2.5, alpha = 0.8)

ppc_error_scatter_avg_vs_x(y, yrep, x, ..., size = 2.5, alpha = 0.8)

ppc_error_binned(y, yrep, ..., size = 1, alpha = 0.25)

```

**Arguments**

<code>y</code>	A vector of observations. See <b>Details</b> .
<code>yrep</code>	An $S$ by $N$ matrix of draws from the posterior predictive distribution, where $S$ is the size of the posterior sample (or subset of the posterior sample used to generate <code>yrep</code> ) and $N$ is the number of observations (the length of <code>y</code> ). The columns of <code>yrep</code> should be in the same order as the data points in <code>y</code> for the plots to make sense. See <b>Details</b> for additional instructions.
<code>...</code>	Currently unused.
<code>binwidth</code>	An optional value used as the <code>binwidth</code> argument to <code>geom_histogram</code> to override the default binwidth.
<code>freq</code>	For histograms, <code>freq=TRUE</code> (the default) puts count on the y-axis. Setting <code>freq=FALSE</code> puts density on the y-axis. (For many plots the y-axis text is off by default. To view the count or density labels on the y-axis see the <code>yaxis_text</code> convenience function.)
<code>group</code>	A grouping variable (a vector or factor) the same length as <code>y</code> . Each value in <code>group</code> is interpreted as the group level pertaining to the corresponding value of <code>y</code> .
<code>size, alpha</code>	For scatterplots, arguments passed to <code>geom_point</code> to control the appearance of the points. For the binned error plot, arguments controlling the size of the outline and opacity of the shaded region indicating the 2-SE bounds.
<code>x</code>	A numeric vector the same length as <code>y</code> to use as the x-axis variable.

**Details**

All of these functions (aside from the `*_scatter_avg` functions) compute and plot predictive errors for each row of the matrix `yrep`, so it is usually a good idea for `yrep` to contain only a small number of draws (rows). See **Examples**, below.

For binomial and Bernoulli data the `ppc_error_binned` function can be used to generate binned error plots. Bernoulli data can be input as a vector of 0s and 1s, whereas for binomial data `y` and `yrep` should contain "success" proportions (not counts). See the **Examples** section, below.

**Value**

A `ggplot` object that can be further customized using the **ggplot2** package.

**Plot descriptions**

`ppc_error_hist` A separate histogram is plotted for the predictive errors computed from `y` and each dataset (row) in `yrep`. For this plot `yrep` should have only a small number of rows.

`ppc_error_hist_grouped` Like `ppc_error_hist`, except errors are computed within levels of a grouping variable. The number of histograms is therefore equal to the product of the number of rows in `yrep` and the number of groups (unique values of `group`).

`ppc_error_scatter` A separate scatterplot is displayed for `y` vs. the predictive errors computed from `y` and each dataset (row) in `yrep`. For this plot `yrep` should have only a small number of rows.

`ppc_error_scatter_avg` A single scatterplot of  $y$  vs. the average of the errors computed from  $y$  and each dataset (row) in `yrep`. For each individual data point  $y[n]$  the average error is the average of the errors for  $y[n]$  computed over the the draws from the posterior predictive distribution.

`ppc_error_scatter_avg_vs_x` Same as `ppc_error_scatter_avg`, except the average is plotted on the  $y$ -axis and a predictor variable  $x$  is plotted on the  $x$ -axis.

`ppc_error_binned` Intended for use with binomial data. A separate binned error plot (similar to [binnedplot](#)) is generated for each dataset (row) in `yrep`. For this plot  $y$  and `yrep` should contain proportions rather than counts, and `yrep` should have only a small number of rows.

## References

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. (Ch. 6)

## See Also

Other PPCs: [PPC-discrete](#), [PPC-distributions](#), [PPC-intervals](#), [PPC-loo](#), [PPC-overview](#), [PPC-scatterplots](#), [PPC-test-statistics](#)

## Examples

```
y <- example_y_data()
yrep <- example_yrep_draws()
ppc_error_hist(y, yrep[1:3, ])

# errors within groups
group <- example_group_data()
(p1 <- ppc_error_hist_grouped(y, yrep[1:3, ], group))
p1 + yaxis_text() # defaults to showing counts on y-axis

table(group) # more obs in GroupB, can set freq=FALSE to show density on y-axis
(p2 <- ppc_error_hist_grouped(y, yrep[1:3, ], group, freq = FALSE))
p2 + yaxis_text()

# scatterplots
ppc_error_scatter(y, yrep[10:14, ])
ppc_error_scatter_avg(y, yrep)

x <- example_x_data()
ppc_error_scatter_avg_vs_x(y, yrep, x)

# ppc_error_binned with binomial model from rstanarm
## Not run:
library(rstanarm)
example("example_model", package = "rstanarm")
formula(example_model)

# get observed proportion of "successes"
y <- example_model$y # matrix of "success" and "failure" counts
```

```

trials <- rowSums(y)
y_prop <- y[, 1] / trials # proportions

# get predicted success proportions
yrep <- posterior_predict(example_model)
yrep_prop <- sweep(yrep, 2, trials, "/")

ppc_error_binned(y_prop, yrep_prop[1:6, ])

## End(Not run)

```

---

PPC-intervals

*PPC intervals*


---

### Description

Medians and central interval estimates of `yrep` with `y` overlaid. See the **Plot Descriptions** section, below.

### Usage

```

ppc_intervals(y, yrep, x = NULL, ..., prob = 0.9, size = 1, fatten = 3)

ppc_intervals_grouped(y, yrep, x = NULL, group, facet_args = list(), ...,
  prob = 0.9, size = 1, fatten = 3)

ppc_ribbon(y, yrep, x = NULL, ..., prob = 0.9, alpha = 0.33,
  size = 0.25)

ppc_ribbon_grouped(y, yrep, x = NULL, group, facet_args = list(), ...,
  prob = 0.9, alpha = 0.33, size = 0.25)

ppc_intervals_data(y, yrep, x = NULL, group = NULL, prob = 0.9, ...)

ppc_ribbon_data(y, yrep, x = NULL, group = NULL, prob = 0.9, ...)

```

### Arguments

<code>y</code>	A vector of observations. See <b>Details</b> .
<code>yrep</code>	An $S$ by $N$ matrix of draws from the posterior predictive distribution, where $S$ is the size of the posterior sample (or subset of the posterior sample used to generate <code>yrep</code> ) and $N$ is the number of observations (the length of <code>y</code> ). The columns of <code>yrep</code> should be in the same order as the data points in <code>y</code> for the plots to make sense. See <b>Details</b> for additional instructions.
<code>x</code>	A numeric vector the same length as <code>y</code> to use as the x-axis variable. For example, <code>x</code> could be a predictor variable from a regression model, a time variable for time-series models, etc. If <code>x</code> is missing or <code>NULL</code> , then <code>1:length(y)</code> is used for the x-axis.

...	Currently unused.
prob	A value between 0 and 1 indicating the desired probability mass to include in the yrep intervals. The default is 0.9.
group	A grouping variable (a vector or factor) the same length as y. Each value in group is interpreted as the group level pertaining to the corresponding value of y.
facet_args	An optional list of arguments (other than facets) passed to <a href="#">facet_wrap</a> to control faceting.
alpha, size, fatten	Arguments passed to geoms. For ribbon plots alpha and size are passed to <a href="#">geom_ribbon</a> . For interval plots size and fatten are passed to <a href="#">geom_pointrange</a> .

### Value

A ggplot object that can be further customized using the **ggplot2** package. The `_data` functions return the data that would have been drawn by the plotting function.

### Plot Descriptions

`ppc_intervals`, `ppc_ribbon`  $100 \times \text{prob} \%$  central intervals for yrep at each x value. `ppc_intervals` plots intervals as vertical bars with points indicating yrep medians and darker points indicating observed y values. `ppc_ribbon` plots a ribbon of connected intervals with a line through the median of yrep and a darker line connecting observed y values. In both cases an optional x variable can also be specified for the x-axis variable.

Depending on the number of observations and the variability in the predictions at different values of x, one or the other of these plots may be easier to read than the other.

`ppc_intervals_grouped`, `ppc_ribbon_grouped` Same as `ppc_intervals` and `ppc_ribbon`, respectively, but a separate plot (facet) is generated for each level of a grouping variable.

### References

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. (Ch. 6)

### See Also

Other PPCs: [PPC-discrete](#), [PPC-distributions](#), [PPC-errors](#), [PPC-loo](#), [PPC-overview](#), [PPC-scatterplots](#), [PPC-test-statistics](#)

### Examples

```
y <- rnorm(50)
yrep <- matrix(rnorm(5000, 0, 2), ncol = 50)

color_scheme_set("brightblue")
ppc_ribbon(y, yrep)
ppc_intervals(y, yrep)

color_scheme_set("teal")
```

```

year <- 1950:1999
ppc_ribbon(y, yrep, x = year, alpha = 0, size = 0.75) + ggplot2::xlab("Year")

color_scheme_set("pink")
year <- rep(2000:2009, each = 5)
group <- gl(5, 1, length = 50, labels = LETTERS[1:5])
ppc_ribbon_grouped(y, yrep, x = year, group) +
  ggplot2::scale_x_continuous(breaks = pretty)

ppc_ribbon_grouped(
  y, yrep, x = year, group,
  facet_args = list(scales = "fixed"),
  alpha = 1,
  size = 2
) +
  xaxis_text(FALSE) +
  xaxis_ticks(FALSE) +
  panel_bg(fill = "gray20")

ppc_data <- ppc_intervals_data(y, yrep, x = year, prob = 0.5)
ppc_group_data <- ppc_intervals_data(y, yrep, x = year, group, prob = 0.5)

## Not run:
library("rstanarm")
fit <- stan_glm(mpg ~ wt + (1|cyl), data = mtcars)
yrep <- posterior_predict(fit)

color_scheme_set("purple")
with(mtcars, ppc_intervals(mpg, yrep, x = wt, prob = 0.5)) +
  panel_bg(fill="gray90", color = NA) +
  grid_lines(color = "white")

ppc_intervals_grouped(y = mtcars$mpg, yrep, prob = 0.8,
  x = mtcars$wt, group = mtcars$cyl)

color_scheme_set("gray")
ppc_intervals(mtcars$mpg, yrep, prob = 0.5) +
  ggplot2::scale_x_continuous(
    labels = rownames(mtcars),
    breaks = 1:nrow(mtcars)
  ) +
  xaxis_text(angle = -70, vjust = 1, hjust = 0)

## End(Not run)

```

**Description**

Leave-One-Out (LOO) predictive checks. See the **Plot Descriptions** section below for details.

**Usage**

```
ppc_loo_pit(y, yrep, lw, pit, compare = c("uniform", "normal"), ...,
  size = 2, alpha = 1)
```

```
ppc_loo_intervals(y, yrep, lw, intervals = NULL, ..., prob = 0.9,
  size = 1, fatten = 3, order = c("index", "median"))
```

```
ppc_loo_ribbon(y, yrep, lw, intervals = NULL, ..., prob = 0.9,
  alpha = 0.33, size = 0.25)
```

**Arguments**

<code>y</code>	A vector of observations. See <b>Details</b> .
<code>yrep</code>	An $S$ by $N$ matrix of draws from the posterior predictive distribution, where $S$ is the size of the posterior sample (or subset of the posterior sample used to generate <code>yrep</code> ) and $N$ is the number of observations (the length of <code>y</code> ). The columns of <code>yrep</code> should be in the same order as the data points in <code>y</code> for the plots to make sense. See <b>Details</b> for additional instructions.
<code>lw</code>	A matrix of (smoothed) log weights with the same dimensions as <code>yrep</code> . See the <a href="#"><code>psislw</code></a> function in the <b>loo</b> package, which returns smoothed weights that can be used to specify <code>lw</code> .
<code>pit</code>	For <code>ppc_loo_pit</code> , optionally a vector of precomputed PIT values that can be specified instead of <code>y</code> , <code>yrep</code> , and <code>lw</code> (these are all ignored if <code>pit</code> is specified). If not specified the PIT values are computed internally before plotting.
<code>compare</code>	For <code>ppc_loo_pit</code> , a string that can be either "uniform" or "normal". If "uniform" (the default) the Q-Q plot compares computed PIT values to the standard uniform distribution. If <code>compare="normal"</code> , the Q-Q plot compares standardized PIT values to the standard normal distribution.
<code>...</code>	Currently unused.
<code>alpha</code> , <code>size</code> , <code>fatten</code>	Arguments passed to code geoms to control plot aesthetics. For <code>ppc_loo_pit</code> , <code>size</code> and <code>alpha</code> are passed to <a href="#"><code>geom_point</code></a> . For <code>ppc_loo_intervals</code> , <code>size</code> and <code>fatten</code> are passed to <a href="#"><code>geom_pointrange</code></a> . For <code>ppc_loo_ribbon</code> , <code>alpha</code> and <code>size</code> are passed to <a href="#"><code>geom_ribbon</code></a> .
<code>intervals</code>	For <code>ppc_loo_intervals</code> and <code>ppc_loo_ribbon</code> , optionally a matrix of precomputed LOO predictive intervals intervals with that can be specified instead of <code>yrep</code> and <code>lw</code> (these are both ignored if <code>intervals</code> is specified). If not specified the intervals are computed internally before plotting. If specified, <code>intervals</code> must be a matrix with number of rows equal to the number of data points and three columns in the following order: the first for the lower bound of the interval, the second for median (50%), and the third for the interval upper bound (column names are ignored).



prob	A value between 0 and 1 indicating the desired probability mass to include in the intervals. The default is 0.9.
order	For <code>ppc_loo_intervals</code> A string indicating how to arrange the plotted intervals. The default ("index") is to plot them in the order of the observations. The alternative ("median") arranges them by median value from smallest (left) to largest (right).

### Value

A `ggplot` object that can be further customized using the **ggplot2** package.

### Plot Descriptions

`ppc_loo_pit` The calibration of marginal predictions can be assessed using probability integral transformation (PIT) checks. LOO improves the check by avoiding the double use of data. See the section on marginal predictive checks in Gelman et al. (2013, p. 152–153). The default LOO PIT predictive check is a quantile-quantile (Q-Q) plot comparing the LOO PITs to the standard uniform distribution. Comparing to a uniform distribution is not good for extreme probabilities close to 0 and 1, so it can be useful to set the `compare` argument to "normal", which will produce a Q-Q plot comparing standardized PIT values to the standard normal distribution. This can be helpful to see the calibration better for the extreme values.

`ppc_loo_intervals`, `ppc_loo_ribbon` Similar to `ppc_intervals` and `ppc_ribbon` but the intervals are for the LOO predictive distribution.

### References

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. (p. 152–153)

Vehtari, A., Gelman, A., and Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*. 27(5), 1413–1432. doi:10.1007/s11222-016-9696-4. arXiv preprint: <http://arxiv.org/abs/1507.04544/>

### See Also

Other PPCs: [PPC-discrete](#), [PPC-distributions](#), [PPC-errors](#), [PPC-intervals](#), [PPC-overview](#), [PPC-scatterplots](#), [PPC-test-statistics](#)

### Examples

```
## Not run:
library(rstanarm)
library(loo)

head(radon)
fit <- stan_lmer(log_radon ~ floor + log_uranium + floor:log_uranium
                + (1 + floor | county), data = radon, cores = 2)
y <- radon$log_radon
yrep <- posterior_predict(fit)
psis <- psislw(-log_lik(fit), cores = 2)
```

```

# marginal predictive check using LOO probability integral transform
color_scheme_set("orange")
ppc_loo_pit(y, yrep, lw = psis$lw_smooth)
ppc_loo_pit(y, yrep, lw = psis$lw_smooth, compare = "normal")

# loo predictive intervals vs observations
sel <- 800:900
ppc_loo_intervals(y[sel], yrep[, sel], psis$lw_smooth[, sel],
                  prob = 0.9, size = 0.5)

color_scheme_set("gray")
ppc_loo_intervals(y[sel], yrep[, sel], psis$lw_smooth[, sel],
                  order = "median", prob = 0.8, size = 0.5)

## End(Not run)

```

## Description

The **bayesplot** PPC module provides various plotting functions for creating graphical displays comparing observed data to simulated data from the posterior predictive distribution. See below for a brief discussion of the ideas behind posterior predictive checking, a description of the structure of this package, and tips on providing an interface to **bayesplot** from another package.

## Details

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.

**Posterior predictive distribution:** To generate the data used for posterior predictive checks 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 outcome values), and typically predictors  $X$ , to update our beliefs about the unknown parameters  $\theta$  in the model. For each draw of the parameters  $\theta$  from the posterior distribution  $p(\theta | y, X)$  we generate an entire vector of outcomes. The result is an  $S \times N$  matrix of simulations, where  $S$  is the size of the posterior sample (number of draws from the posterior distribution) and  $N$  is the number of data points in  $y$ . That is, each row of the matrix is an individual "replicated" dataset of  $N$  observations.

**Notation:** 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. This corresponds to the notation from Gelman et. al. (2013) and is the notation used throughout the documentation for this package.

**Graphical posterior predictive checking:** Using the datasets  $y^{rep}$  drawn from the posterior predictive distribution, the functions in the **bayesplot** package produce various graphical displays comparing the observed data  $y$  to the replications. For a more thorough discussion of posterior predictive checking see Chapter 6 of Gelman et. al. (2013).

### PPC plotting functions

The plotting functions for posterior predictive checking in this package are organized into several categories, each with its own documentation:

**Distributions** Histograms, kernel density estimates, boxplots, and other plots comparing the empirical distribution of the observed data  $y$  to the distributions of individual replicated datasets (rows) in  $yrep$ .

**Test statistics** The distribution of a test statistic, or a pair of test statistics, over the replicated datasets (rows) in  $yrep$  compared to value of the statistic(s) computed from  $y$ .

**Intervals** Interval estimates of  $yrep$  with  $y$  overlaid. The x-axis variable can be optionally specified by the user (e.g. to plot against against a predictor variable or over time).

**Predictive errors** Plots of predictive errors ( $y - yrep$ ) computed from  $y$  and replicated datasets (rows) in  $yrep$ . For binomial models binned error plots are also available.

**Scatterplots** Scatterplots (and similar visualizations) of the observed data  $y$  vs. individual replicated datasets (rows) in  $yrep$ , or vs. the average value of the distributions of each data point (columns) in  $yrep$ .

**Plots for discrete outcomes** PPC functions that can only be used if  $y$  and  $yrep$  are discrete. For example, rootograms for count outcomes and bar plots for ordinal, categorical, and multinomial outcomes.

**LOO predictive checks** PPC functions for predictive checks based on (approximate) leave-one-out (LOO) cross-validation.

### Providing an interface for posterior predictive checking from another package

In addition to the various plotting functions, the **bayesplot** package provides the S3 generic `pp_check`. Authors of R packages for Bayesian inference are encouraged to define `pp_check` methods for the fitted model objects created by their packages. See the package vignettes for more details and a simple example, and see the **rstanarm** and **brms** packages for full examples of `pp_check` methods.

### References

- Gabry, J., Simpson, D., Vehtari, A., Betancourt, M., Gelman, A. (2017). Visualization in Bayesian workflow. arXiv preprint [arxiv:1709.01449](https://arxiv.org/abs/1709.01449).
- 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. (Ch. 6)

### See Also

Other PPCs: [PPC-discrete](#), [PPC-distributions](#), [PPC-errors](#), [PPC-intervals](#), [PPC-loo](#), [PPC-scatterplots](#), [PPC-test-statistics](#)

---

 PPC-scatterplots      *PPC scatterplots*


---

### Description

Scatterplots of the observed data  $y$  vs. simulated/replicated data  $y_{rep}$  from the posterior predictive distribution. See the **Plot Descriptions** and **Details** sections, below.

### Usage

```
ppc_scatter(y, yrep, ..., size = 2.5, alpha = 0.8)

ppc_scatter_avg(y, yrep, ..., size = 2.5, alpha = 0.8)

ppc_scatter_avg_grouped(y, yrep, group, ..., size = 2.5, alpha = 0.8)
```

### Arguments

<code>y</code>	A vector of observations. See <b>Details</b> .
<code>yrep</code>	An $S$ by $N$ matrix of draws from the posterior predictive distribution, where $S$ is the size of the posterior sample (or subset of the posterior sample used to generate $y_{rep}$ ) and $N$ is the number of observations (the length of $y$ ). The columns of $y_{rep}$ should be in the same order as the data points in $y$ for the plots to make sense. See <b>Details</b> for additional instructions.
<code>...</code>	Currently unused.
<code>size, alpha</code>	Arguments passed to <code>geom_point</code> to control the appearance of the points.
<code>group</code>	A grouping variable (a vector or factor) the same length as $y$ . Each value in <code>group</code> is interpreted as the group level pertaining to the corresponding value of $y$ .

### Details

For Binomial data, the plots will typically be most useful if  $y$  and  $y_{rep}$  contain the "success" proportions (not discrete "success" or "failure" counts).

### Value

A ggplot object that can be further customized using the **ggplot2** package.

### Plot Descriptions

`ppc_scatter` For each dataset (row) in  $y_{rep}$  a scatterplot is generated showing  $y$  against that row of  $y_{rep}$ . For this plot  $y_{rep}$  should only contain a small number of rows.

`ppc_scatter_avg` A scatterplot of  $y$  against the average values of  $y_{rep}$ , i.e., the points  $(\text{mean}(y_{rep}[, n]), y[n])$ , where each  $y_{rep}[, n]$  is a vector of length equal to the number of posterior draws.

`ppc_scatter_avg_grouped` The same as `ppc_scatter_avg`, but a separate plot is generated for each level of a grouping variable.

## References

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. (Ch. 6)

## See Also

Other PPCs: [PPC-discrete](#), [PPC-distributions](#), [PPC-errors](#), [PPC-intervals](#), [PPC-loo](#), [PPC-overview](#), [PPC-test-statistics](#)

## Examples

```
y <- example_y_data()
yrep <- example_yrep_draws()
p1 <- ppc_scatter_avg(y, yrep)
p1
p2 <- ppc_scatter(y, yrep[20:23, ], alpha = 0.5, size = 1.5)
p2

# give x and y axes the same limits
lims <- ggplot2::lims(x = c(0, 160), y = c(0, 160))
p1 + lims
p2 + lims

group <- example_group_data()
ppc_scatter_avg_grouped(y, yrep, group, alpha = 0.7) + lims
```

---

PPC-test-statistics    *PPC test statistics*

---

## Description

The distribution of a test statistic  $T(\text{yrep})$ , or a pair of test statistics, over the simulated datasets in `yrep`, compared to the observed value  $T(y)$  computed from the data `y`. See the **Plot Descriptions** and **Details** sections, below.

## Usage

```
ppc_stat(y, yrep, stat = "mean", ..., binwidth = NULL, freq = TRUE)

ppc_stat_grouped(y, yrep, group, stat = "mean", ..., facet_args = list(),
  binwidth = NULL, freq = TRUE)

ppc_stat_freqpoly_grouped(y, yrep, group, stat = "mean", ...,
  facet_args = list(), binwidth = NULL, freq = TRUE)

ppc_stat_2d(y, yrep, stat = c("mean", "sd"), ..., size = 2.5, alpha = 0.7)
```

**Arguments**

<code>y</code>	A vector of observations. See <b>Details</b> .
<code>yrep</code>	An $S$ by $N$ matrix of draws from the posterior predictive distribution, where $S$ is the size of the posterior sample (or subset of the posterior sample used to generate <code>yrep</code> ) and $N$ is the number of observations (the length of <code>y</code> ). The columns of <code>yrep</code> should be in the same order as the data points in <code>y</code> for the plots to make sense. See <b>Details</b> for additional instructions.
<code>stat</code>	A single function or a string naming a function, except for <code>ppc_stat_2d</code> which requires a vector of exactly two functions or function names. In all cases the function(s) should take a vector input and return a scalar test statistic. If specified as a string (or strings) then the legend will display function names. If specified as a function (or functions) then generic naming is used in the legend.
<code>...</code>	Currently unused.
<code>binwidth</code>	An optional value used as the <code>binwidth</code> argument to <code>geom_histogram</code> to override the default binwidth.
<code>freq</code>	For histograms, <code>freq=TRUE</code> (the default) puts count on the y-axis. Setting <code>freq=FALSE</code> puts density on the y-axis. (For many plots the y-axis text is off by default. To view the count or density labels on the y-axis see the <code>yaxis_text</code> convenience function.)
<code>group</code>	A grouping variable (a vector or factor) the same length as <code>y</code> . Each value in <code>group</code> is interpreted as the group level pertaining to the corresponding value of <code>y</code> .
<code>facet_args</code>	A named list of arguments (other than <code>facets</code> ) passed to <code>facet_wrap</code> or <code>facet_grid</code> to control faceting.
<code>size, alpha</code>	Arguments passed to <code>geom_point</code> to control the appearance of scatterplot points.

**Details**

For Binomial data, the plots will typically be most useful if `y` and `yrep` contain the "success" proportions (not discrete "success" or "failure" counts).

**Value**

A `ggplot` object that can be further customized using the **ggplot2** package.

**Plot Descriptions**

- `ppc_stat` A histogram of the distribution of a test statistic computed by applying `stat` to each dataset (row) in `yrep`. The value of the statistic in the observed data, `stat(y)`, is overlaid as a vertical line.
- `ppc_stat_grouped`, `ppc_stat_freqpoly_grouped` The same as `ppc_stat`, but a separate plot is generated for each level of a grouping variable. In the case of `ppc_stat_freqpoly_grouped` the plots are frequency polygons rather than histograms.
- `ppc_stat_2d` A scatterplot showing the joint distribution of two test statistics computed over the datasets (rows) in `yrep`. The value of the statistics in the observed data is overlaid as large point.

## References

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. (Ch. 6)

## See Also

Other PPCs: [PPC-discrete](#), [PPC-distributions](#), [PPC-errors](#), [PPC-intervals](#), [PPC-loo](#), [PPC-overview](#), [PPC-scatterplots](#)

## Examples

```
y <- example_y_data()
yrep <- example_yrep_draws()
ppc_stat(y, yrep)
ppc_stat(y, yrep, stat = "sd") + legend_none()
ppc_stat_2d(y, yrep)
ppc_stat_2d(y, yrep, stat = c("median", "mean")) + legend_move("bottom")

color_scheme_set("teal")
group <- example_group_data()
ppc_stat_grouped(y, yrep, group)

color_scheme_set("mix-red-blue")
ppc_stat_freqpoly_grouped(y, yrep, group, facet_args = list(nrow = 2))

# use your own function to compute test statistics
color_scheme_set("brightblue")
q25 <- function(y) quantile(y, 0.25)
ppc_stat(y, yrep, stat = "q25") # legend includes function name

# can define the function in the 'stat' argument but then
# the legend doesn't include a function name
ppc_stat(y, yrep, stat = function(y) quantile(y, 0.25))
```

---

pp\_check

*Posterior predictive checks (S3 generic and default method)*


---

## Description

S3 generic with simple default method. The intent is to provide a generic so authors of other R packages who wish to provide interfaces to the functions in **bayesplot** will be encouraged to include `pp_check` methods in their package, preserving the same naming conventions for posterior predictive checking across many R packages for Bayesian inference. This is for the convenience of both users and developers. See the **Details** and **Examples** sections, below, and the package vignettes for examples of defining `pp_check` methods.

**Usage**

```
pp_check(object, ...)

## Default S3 method:
pp_check(object, yrep, fun, ...)
```

**Arguments**

object	Typically a fitted model object. The default method, however, takes object to be a y (outcome) vector.
...	For the generic, arguments passed to individual methods. For the default method, these are additional arguments to pass to fun.
yrep	For the default method, a yrep matrix passed to fun.
fun	For the default method, the plotting function to call. Can be any of the <a href="#">PPC functions</a> . The "ppc_" prefix can optionally be dropped if fun is specified as a string.

**Details**

A package that creates fitted model objects of class "foo" can include a method `pp_check.foo` that prepares the appropriate inputs (y, yrep, etc.) for the **bayesplot** functions. The `pp_check.foo` method may, for example, let the user choose between various plots, calling the functions from **bayesplot** internally as needed. See **Examples**, below, and the package vignettes.

**Value**

The exact form of the value returned by `pp_check` may vary by the class of object, but for consistency we encourage authors of methods to return the `ggplot` object created by one of **bayesplot**'s plotting functions. The default method returns the object returned by fun.

**Examples**

```
# default method
y <- example_y_data()
yrep <- example_yrep_draws()
pp_check(y, yrep[1:50,], ppc_dens_overlay)

g <- example_group_data()
pp_check(y, yrep, fun = "stat_grouped", group = g, stat = "median")

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



```

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

```

---

theme_default	<i>Default bayesplot plotting theme</i>
---------------	---

---

## Description

The `theme_default` function returns the default `ggplot` [theme](#) used by the **bayesplot** plotting functions. After loading the **bayesplot** package, this theme will be the default for *all* graphs made with **ggplot2**. See the **Details** section, below.

## Usage

```

theme_default(base_size = getOption("bayesplot.base_size", 12),
             base_family = getOption("bayesplot.base_family", "serif"))

```

## Arguments

`base_size`, `base_family`  
 Base font size and family (passed to [theme\\_bw](#)). It is possible to set `"bayesplot.base_size"` and `"bayesplot.base_family"` via [options](#) to change the defaults, which are 12 and `"serif"`, respectively.

## Details

After loading **bayesplot**, if you subsequently change the default **ggplot2** theme (i.e., by calling [theme\\_set](#) or loading a different package that changes the theme) then **bayesplot** will use that theme instead. To change back to the default **bayesplot** theme use `bayesplot::theme_default()` as the argument to [theme\\_set](#).

## Value

A `ggplot` [theme](#) object.

## See Also

[bayesplot-helpers](#) for a variety of convenience functions, many of which provide shortcuts for tweaking theme elements after creating a plot.

[bayesplot-colors](#) to set or view the color scheme used for plotting.

**Examples**

```
class(theme_default())

# plot using the default theme
ggplot2::theme_set(theme_default())
x <- example_mcmc_draws()
mcmc_hist(x)

# change the default font size and family
ggplot2::theme_set(theme_default(base_size = 8, base_family = "sans"))
mcmc_hist(x)
mcmc_areas(x, regex_pars = "beta")

# change back
ggplot2::theme_set(theme_default())
mcmc_areas(x, regex_pars = "beta")

# use one of the themes included in ggplot2
ggplot2::theme_set(ggplot2::theme_gray())
mcmc_dens_overlay(x)

# change back to bayesplot default theme
ggplot2::theme_set(bayesplot::theme_default())
mcmc_dens_overlay(x)
```

# Index

`abline_01` (bayesplot-helpers), 9  
`aes`, 11  
`arrangeGrob`, 15, 16, 27  
`array`, 29  
`available_mcmc` (available\_ppc), 4  
`available_ppc`, 4

`bayesplot` (bayesplot-package), 2  
`bayesplot-colors`, 5  
`bayesplot-extractors`, 7  
`bayesplot-helpers`, 9, 65  
`bayesplot-package`, 2  
`bayesplot_grid`, 14, 38  
`binnedplot`, 52

`color_scheme_get` (bayesplot-colors), 5  
`color_scheme_set` (bayesplot-colors), 5  
`color_scheme_view` (bayesplot-colors), 5  
`colors`, 6  
`Combinations`, 30  
`Comparisons to true values`, 30

`data frames`, 29  
`density`, 24  
`Distributions`, 59

`element_blank`, 10  
`element_line`, 10  
`element_rect`, 10  
`element_text`, 10

`facet_bg` (bayesplot-helpers), 9  
`facet_grid`, 18, 21, 34, 42, 62  
`facet_text` (bayesplot-helpers), 9  
`facet_wrap`, 18, 21, 34, 42, 46, 54, 62

`General MCMC diagnostics`, 30  
`geom_abline`, 10  
`geom_bar`, 45  
`geom_boxplot`, 48  
`geom_density`, 22, 48  
`geom_hex`, 36  
`geom_histogram`, 18, 22, 27, 34, 48, 51, 62  
`geom_hline`, 10  
`geom_jitter`, 48  
`geom_line`, 18, 31, 45  
`geom_point`, 18, 34, 36, 37, 42, 51, 56, 60, 62  
`geom_pointrange`, 45, 54, 56  
`geom_ribbon`, 54, 56  
`geom_rug`, 43  
`geom_violin`, 22, 48  
`geom_vline`, 10  
`ggsave`, 3  
`grep`, 4  
`grid_lines` (bayesplot-helpers), 9

`hline_0` (bayesplot-helpers), 9  
`hline_at` (bayesplot-helpers), 9

`integer`, 46  
`Intervals`, 59  
`invisibly`, 6

`labs`, 11  
`lhub` (bayesplot-helpers), 9  
`legend_move` (bayesplot-helpers), 9  
`legend_none`, 16  
`legend_none` (bayesplot-helpers), 9  
`legend_text` (bayesplot-helpers), 9  
`log_posterior`, 27, 37  
`log_posterior` (bayesplot-extractors), 7  
`LOO predictive checks`, 59

`match.fun`, 21, 24, 31, 36, 42  
`matrix`, 29  
`MCMC`, 3, 4, 16  
`MCMC (MCMC-overview)`, 29  
`MCMC-combos`, 16  
`MCMC-diagnostics`, 17  
`MCMC-distributions`, 20  
`MCMC-intervals`, 23

- MCMC-nuts, [17](#), [19](#), [26](#)
- MCMC-overview, [16](#), [18](#), [21](#), [24](#), [29](#), [30](#), [33](#), [36](#), [42](#)
- MCMC-parcoord, [30](#)
- MCMC-recover, [33](#)
- MCMC-scatterplots, [35](#)
- MCMC-traces, [41](#)
- mcmc\_acf (MCMC-diagnostics), [17](#)
- mcmc\_acf\_bar (MCMC-diagnostics), [17](#)
- mcmc\_areas (MCMC-intervals), [23](#)
- mcmc\_combo (MCMC-combos), [16](#)
- mcmc\_dens (MCMC-distributions), [20](#)
- mcmc\_dens\_overlay (MCMC-distributions), [20](#)
- mcmc\_hex (MCMC-scatterplots), [35](#)
- mcmc\_hist (MCMC-distributions), [20](#)
- mcmc\_hist\_by\_chain (MCMC-distributions), [20](#)
- mcmc\_intervals (MCMC-intervals), [23](#)
- mcmc\_neff (MCMC-diagnostics), [17](#)
- mcmc\_neff\_data (MCMC-diagnostics), [17](#)
- mcmc\_neff\_hist (MCMC-diagnostics), [17](#)
- mcmc\_nuts\_acceptance (MCMC-nuts), [26](#)
- mcmc\_nuts\_divergence (MCMC-nuts), [26](#)
- mcmc\_nuts\_energy (MCMC-nuts), [26](#)
- mcmc\_nuts\_stepsize (MCMC-nuts), [26](#)
- mcmc\_nuts\_tredepth (MCMC-nuts), [26](#)
- mcmc\_pairs, [28](#)
- mcmc\_pairs (MCMC-scatterplots), [35](#)
- mcmc\_parcoord, [28](#)
- mcmc\_parcoord (MCMC-parcoord), [30](#)
- mcmc\_parcoord\_data (MCMC-parcoord), [30](#)
- mcmc\_recover\_hist (MCMC-recover), [33](#)
- mcmc\_recover\_intervals (MCMC-recover), [33](#)
- mcmc\_recover\_scatter (MCMC-recover), [33](#)
- mcmc\_rhat (MCMC-diagnostics), [17](#)
- mcmc\_rhat\_data (MCMC-diagnostics), [17](#)
- mcmc\_rhat\_hist (MCMC-diagnostics), [17](#)
- mcmc\_scatter, [28](#)
- mcmc\_scatter (MCMC-scatterplots), [35](#)
- mcmc\_trace, [28](#)
- mcmc\_trace (MCMC-traces), [41](#)
- mcmc\_trace\_highlight (MCMC-traces), [41](#)
- mcmc\_violin (MCMC-distributions), [20](#)
- melt, [8](#)
- neff\_ratio, [18](#)
- neff\_ratio (bayesplot-extractors), [7](#)
- No-U-Turn Sampler (NUTS), [3](#)
- NUTS, [31](#), [38](#), [42](#), [43](#)
- NUTS (MCMC-nuts), [26](#)
- NUTS diagnostics, [30](#)
- nuts\_params, [27](#), [31](#), [36](#), [42](#)
- nuts\_params (bayesplot-extractors), [7](#)
- options, [65](#)
- overlay\_function (bayesplot-helpers), [9](#)
- pairs\_condition (MCMC-scatterplots), [35](#)
- pairs\_style\_np (MCMC-scatterplots), [35](#)
- panel\_bg (bayesplot-helpers), [9](#)
- parcoord\_style\_np (MCMC-parcoord), [30](#)
- plot\_bg (bayesplot-helpers), [9](#)
- Plots for discrete outcomes, [59](#)
- Posterior distributions, [29](#)
- pp\_check, [59](#), [63](#)
- PPC, [3](#), [4](#)
- PPC (PPC-overview), [58](#)
- PPC functions, [64](#)
- PPC-discrete, [45](#)
- PPC-distributions, [47](#)
- PPC-errors, [50](#)
- PPC-intervals, [53](#)
- PPC-loo, [55](#)
- PPC-overview, [58](#)
- PPC-scatterplots, [60](#)
- PPC-test-statistics, [61](#)
- ppc\_bars (PPC-discrete), [45](#)
- ppc\_bars\_grouped (PPC-discrete), [45](#)
- ppc\_boxplot (PPC-distributions), [47](#)
- ppc\_dens (PPC-distributions), [47](#)
- ppc\_dens\_overlay (PPC-distributions), [47](#)
- ppc\_ecdf\_overlay (PPC-distributions), [47](#)
- ppc\_error\_binned (PPC-errors), [50](#)
- ppc\_error\_hist (PPC-errors), [50](#)
- ppc\_error\_hist\_grouped (PPC-errors), [50](#)
- ppc\_error\_scatter (PPC-errors), [50](#)
- ppc\_error\_scatter\_avg (PPC-errors), [50](#)
- ppc\_error\_scatter\_avg\_vs\_x (PPC-errors), [50](#)
- ppc\_freqpoly (PPC-distributions), [47](#)
- ppc\_freqpoly\_grouped (PPC-distributions), [47](#)
- ppc\_hist (PPC-distributions), [47](#)
- ppc\_intervals, [57](#)
- ppc\_intervals (PPC-intervals), [53](#)
- ppc\_intervals\_data (PPC-intervals), [53](#)

- ppc\_intervals\_grouped (PPC-intervals),  
53
- ppc\_loo\_intervals (PPC-loo), 55
- ppc\_loo\_pit (PPC-loo), 55
- ppc\_loo\_ribbon (PPC-loo), 55
- ppc\_ribbon, 57
- ppc\_ribbon (PPC-intervals), 53
- ppc\_ribbon\_data (PPC-intervals), 53
- ppc\_ribbon\_grouped (PPC-intervals), 53
- ppc\_rootogram (PPC-discrete), 45
- ppc\_scatter (PPC-scatterplots), 60
- ppc\_scatter\_avg (PPC-scatterplots), 60
- ppc\_scatter\_avg\_grouped  
(PPC-scatterplots), 60
- ppc\_stat (PPC-test-statistics), 61
- ppc\_stat\_2d (PPC-test-statistics), 61
- ppc\_stat\_freqpoly\_grouped  
(PPC-test-statistics), 61
- ppc\_stat\_grouped (PPC-test-statistics),  
61
- ppc\_violin\_grouped (PPC-distributions),  
47
- Predictive errors, 59
- psislw, 56
  
- regular expression, 8, 18, 21, 24, 31, 36, 42
- Rhat, 18
- rhat, 24
- rhat (bayesplot-extractors), 7
- rstan, 2
  
- scatter\_style\_np (MCMC-scatterplots), 35
- Scatterplots, 30, 59
- stat\_ecdf, 48
- stat\_function, 10, 11
  
- Test statistics, 59
- theme, 10, 11, 16, 65
- theme\_bw, 65
- theme\_default, 3, 6, 12, 65, 65
- theme\_set, 65
- Trace plots, 30
- trace\_style\_np (MCMC-traces), 41
  
- Uncertainty intervals, 29
  
- vline\_0 (bayesplot-helpers), 9
- vline\_at (bayesplot-helpers), 9
  
- xaxis\_text (bayesplot-helpers), 9
- xaxis\_ticks (bayesplot-helpers), 9
- xaxis\_title (bayesplot-helpers), 9
  
- yaxis\_text, 22, 48, 51, 62
- yaxis\_text (bayesplot-helpers), 9
- yaxis\_ticks (bayesplot-helpers), 9
- yaxis\_title (bayesplot-helpers), 9