Package: bayesplot (via r-universe)

November 20, 2026

Type Package

Title Plotting for Bayesian Models

Version 1.17.0

Date 2026-11-19

Maintainer Jonah Gabry <jgabry@gmail.com>

Description Plotting functions for posterior analysis, MCMC diagnostics, prior and posterior predictive checks, and other visualizations to support the applied Bayesian workflow advocated in Gabry, Simpson, Vehtari, Betancourt, and Gelman (2019) <doi:10.1111/rssa.12378>. 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)

URL https://mc-stan.org/bayesplot/

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

SystemRequirements pandoc (>= 1.12.3)

Depends R (>= 4.1.0)

Imports dplyr (>= 1.0.0), ggdist, ggplot2 (>= 3.5.0), ggridges (>= 0.5.5), glue, lifecycle, posterior (>= 1.8.0), reshape2, rlang (>= 1.0.0), stats, tibble (>= 2.0.0), tidyr, tidyselect, utils

Suggests cmdstanr, ggfortify, gridExtra (>= 2.2.1), hexbin, knitr (>= 1.16), loo (>= 2.0.0), monotone, patchwork, RColorBrewer, rmarkdown (>= 1.0.0), rstan (>= 2.17.1), rstanarm (>= 2.17.4), rstantools (>= 2.0.0), scales, shinystan (>= 2.3.0), survival, testthat (>= 3.0.0), vdiffr (>= 1.0.2)

VignetteBuilder knitr

Encoding UTF-8

Config/testthat/edition 3

Additional_repositories https://stan-dev.r-universe.dev/

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Jonah Gabry [aut, cre], Tristan Mahr [aut] (ORCID: <https://orcid.org/0000-0002-8890-5116>), Paul-Christian Bürkner [ctb], Martin Modrák [ctb], Malcolm Barrett [ctb], Frank Weber [ctb], Eduardo Coronado Sroka [ctb], Teemu Sailynoja [ctb], Aki Vehtari [ctb], Behram Ulukır [ctb], Visruth Srimath Kandali [ctb], Mattan S. Ben-Shachar [ctb], Ishaan Arora [ctb], Utkarsh Pawade [ctb], Florence Bockting [ctb], Saku Suorsa [ctb] (ORCID: <https://orcid.org/0009-0009-6476-0045>)

Repository https://cran.r-universe.dev

Date/Publication 2026-11-20 01:10:02 UTC

RemoteUrl https://github.com/cran/bayesplot

RemoteRef HEAD

RemoteSha 483089361ffc93d87d55cd50e85878d3154b56b0

Contents

bayesplot-package . . . 3
available_ppc . . . 5
bayesplot-colors . . . 6
bayesplot-extractors . . . 9
bayesplot-helpers . . . 11
bayesplot_grid . . . 17
bayesplot_theme_get . . . 19
MCMC-combos . . . 21
MCMC-diagnostics . . . 22
MCMC-distributions . . . 26
MCMC-intervals . . . 32
MCMC-nuts . . . 39
MCMC-overview . . . 42
MCMC-parcoord . . . 44
MCMC-recover . . . 47
MCMC-scatterplots . . . 50
MCMC-traces . . . 57
pp_check . . . 64
PPC-calibration . . . 65
PPC-censoring . . . 70
PPC-discrete . . . 73
PPC-distributions . . . 78
PPC-errors . . . 86
PPC-intervals . . . 90
PPC-loo . . . 95

PPC-overview 102
PPC-scatterplots 104
PPC-test-statistics 107
PPD-distributions 111
PPD-intervals 115
PPD-overview 118
PPD-test-statistics 119
theme_default 122
tidy-params 123

Index **127**

`bayesplot-package` **bayesplot***: Plotting for Bayesian Models*

Description

Stan Development Team

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 (the R interface to Stan). 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 or prior) predictive checks (PPCs).
- PPD: Plots of (posterior or prior) predictive distributions without comparisons to observed data.

Resources

- **Online documentation and vignettes**: Visit the **bayesplot** website at `https://mc-stan.org/bayesplot/`
- **Bug reports and feature requests**: If you would like to request a new feature or if you have noticed a bug that needs to be fixed please let us know at the **bayesplot** issue tracker at `https://github.com/stan-dev/bayesplot/issues/`
- **General questions and help**: To ask a question about **bayesplot** on the Stan Forums forum please visit `https://discourse.mc-stan.org`.

Author(s)

Maintainer: Jonah Gabry <jgabry@gmail.com>

Authors:

- Jonah Gabry <jgabry@gmail.com>
- Tristan Mahr (ORCID)

Other contributors:

- Paul-Christian Bürkner [contributor]
- Martin Modrák [contributor]
- Malcolm Barrett [contributor]
- Frank Weber [contributor]
- Eduardo Coronado Sroka [contributor]
- Teemu Sailynoja [contributor]
- Aki Vehtari [contributor]
- Behram Ulukır [contributor]
- Visruth Srimath Kandali [contributor]
- Mattan S. Ben-Shachar [contributor]
- Ishaan Arora [contributor]
- Utkarsh Pawade [contributor]
- Florence Bockting [contributor]

References

Gabry, J. , Simpson, D. , Vehtari, A. , Betancourt, M. and Gelman, A. (2019), Visualization in Bayesian workflow. *J. R. Stat. Soc. A*, 182: 389-402. doi:10.1111/rssa.12378. (journal version, arXiv preprint, code on GitHub)

See Also

`theme_default()` for the default ggplot theme used by **bayesplot** and `bayesplot_theme_set()` to change it.

bayesplot-colors to set or view the color scheme used for plotting.

`ggplot2::ggsave()` 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, ])

# Same plots but without y (using ppd_ instead of ppc_)
bayesplot_theme_set(ggplot2::theme_gray())
ypred <- yrep
ppd_dens_overlay(ypred[1:50, ])
ppd_stat(ypred, stat = "median") + grid_lines()
ppd_hist(ypred[1:8, ])
```

`available_ppc` *Get or view the names of available plotting or data functions*

Description

Get or view the names of available plotting or data functions

Usage

```
available_ppc(pattern = NULL, fixed = FALSE, invert = FALSE, plots_only = TRUE)

available_ppd(pattern = NULL, fixed = FALSE, invert = FALSE, plots_only = TRUE)

available_mcmc(
  pattern = NULL,
  fixed = FALSE,
  invert = FALSE,
  plots_only = TRUE
)
```

Arguments

`pattern`, `fixed`, `invert`
: Passed to `base::grep()`.

`plots_only`
: If `TRUE` (the default) only plotting functions are searched for. If `FALSE` then functions that return data for plotting (functions ending in `_data()`) are also included.

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, PPC, or PPD 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()
available_ppc("grouped")
available_ppc("grouped", invert = TRUE)

available_ppd()
available_ppd("grouped")

# can also see which functions that return data are available
available_ppc(plots_only = FALSE)

# only show the _data functions
available_ppc("_data", plots_only = FALSE)
available_ppd("_data", plots_only = FALSE)
available_mcmc("_data", plots_only = FALSE)
```

---

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

Description

Set, get, or view color schemes. Choose from a preset scheme or create a custom scheme. See the **Available color schemes** section below for a list of available scheme names. The **Custom color schemes** section describes how to specify a custom scheme.

Usage

```
color_scheme_set(scheme = "blue")

color_scheme_get(scheme = NULL, i = NULL)

color_scheme_view(scheme = NULL)
```

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. 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. See the **Available color schemes** section below for a list of available scheme names. The **Custom color schemes** section describes how to specify a custom scheme.
i	For `color_scheme_get()`, an optional 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.

Available color schemes

Currently, the available preset color schemes are:

- `"blue"`, `"brightblue"`
- `"gray"`, `"darkgray"`
- `"green"`
- `"pink"`
- `"purple"`
- `"red"`
- `"teal"`
- `"yellow"`

- "viridis", "viridisA", "viridisB", "viridisC", "viridisD", "viridisE"
- "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.
- "brewer-x", replacing x with the name of a palette available from RColorBrewer::brewer.pal() (e.g., brewer-PuBuGn).

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

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** and bayesplot_theme_set() to change it.

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()

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

########################
### ColorBrewer scheme ###
########################
color_scheme_set("brewer-Spectral")
color_scheme_view()
mcmc_trace(x, pars = "sigma")

########################
### Custom color scheme ###
########################
orange_scheme <- c("#ffebcc", "#ffcc80",
                   "#ffad33", "#e68a00",
                   "#995c00", "#663d00")
color_scheme_set(orange_scheme)
color_scheme_view()
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 provided for stanfit (**rstan**), CmdStanMCMC (**cmdstanr**), 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 'CmdStanMCMC'
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 'CmdStanMCMC'
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 'CmdStanMCMC'
rhat(object, 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, ...)

## S3 method for class 'CmdStanMCMC'
neff_ratio(object, 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 regular expression to use for parameter selection. Can be specified instead of `pars` or in addition to `pars`. When using `pars` for tidy parameter selection, the `regex_pars` argument is ignored since select helpers perform a similar function.

Value

`log_posterior()` `log_posterior()` methods return a molten data frame (see `reshape2::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 `reshape2::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, refresh = 0)

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", linewidth = 0.2, size = deprecated())

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 `vline_`, `hline_`, and `abline_` functions, `...` is passed to `ggplot2::geom_vline()`, `ggplot2::geom_hline()`, and `ggplot2::geom_abline()`, respectively, to control the appearance of the line(s). For functions ending in `_bg`, `...` is passed to `ggplot2::element_rect()`. For functions ending in `_text` or `_title`, `...` is passed to `ggplot2::element_text()`. For `xaxis_ticks` and `yaxis_ticks`, `...` is passed to `ggplot2::element_line()`. For `overlay_function`, `...` is passed to `ggplot2::stat_function()`.
`na.rm`	A logical scalar passed to the appropriate geom (e.g. `ggplot2::geom_vline()`). 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 `position = "none"` is also allowed and is equivalent to using `legend_none()`.
`on`	For functions modifying ggplot theme elements, set `on=FALSE` to set the element to `ggplot2::element_blank()`. For example, facet text can be removed by adding `facet_text(on=FALSE)`, or simply `facet_text(FALSE)` to a ggplot object. If `on=TRUE` (the default), then `...` can be used to customize the appearance of the theme element.
`color, linewidth`	Passed to `ggplot2::element_line()`.
`size`	**[Deprecated]** Use `linewidth` instead.

Details

Add vertical, horizontal, and diagonal lines to plots:

- `vline_at()` and `hline_at()` return an object created by either `ggplot2::geom_vline()` or `ggplot2::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 `ggplot2::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\` of `x`, as well as the median (if `med=TRUE`).

Control appearance of facet strips:

- `facet_text()` returns ggplot2 theme objects that can be added to an existing plot (ggplot object) to format the text in facet strips.
- `facet_bg()` can be added to a plot to change the background of 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 or remove it.
- `legend_text()` works much like `facet_text()` but for the legend.

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 `ggplot2::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 `ggplot2::stat_function()` but with the `inherit.aes` argument fixed to `FALSE`. Fixing `inherit.aes=FALSE` will avoid potential errors due to the `ggplot2::aes()`thetic mapping used by certain **bayesplot** plotting functions.

Value

A **ggplot2** layer or `ggplot2::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(linewidth = 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, linewidth = 0.25)

my_lines <- vline_at(v, alpha = 0.25, linewidth = 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, linewidth = 0.25)
p + vline_at(x[, 3:4], function(a) apply(a, 2, mean),
             color = "orange",
             linewidth = 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",
              linewidth = 2 * c(1,.5,1), alpha = 0.75)
p2 + vline_at(b1, lbub(0.8, med = FALSE), color = "gray20",
              linewidth = 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(linewidth = 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", linewidth = 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",
    linewidth = 2
  )

color_scheme_set("gray")
mcmc_hist(x, pars = "beta[1]", freq = FALSE) + 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,
  save_gg_objects = TRUE
)
```

Arguments

...	One or more ggplot objects.
plots	A list of ggplot objects. Can be used as an alternative to specifying plot objects via ....
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 gridExtra::arrangeGrob() (nrow, ncol, widths, etc.).

titles, subtitles
: Optional character vectors of plot titles and subtitles. If specified, `titles` and `subtitles` must 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`.

save_gg_objects
: If `TRUE`, the default, then the ggplot objects specified in `...` or via the `plots` argument are saved in a list in the `"bayesplots"` component of the returned object. Setting this to `FALSE` will make the returned object smaller but these individual plot objects will not be available.

Value

An object of class `"bayesplot_grid"` (essentially a gtable object from `gridExtra::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, refresh = 0)
fit2 <- stan_glm(log_mpg ~ wt, data = mtcars, refresh = 0)

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)
```

---

bayesplot_theme_get *Get, set, and modify the active* **bayesplot** *theme*

Description

These functions are the **bayesplot** equivalent to **ggplot2**'s ggplot2::theme_set() and friends. They set, get, and update the active theme but only apply them to bayesplots. The current/active theme is automatically applied to every bayesplot you draw.

Use bayesplot_theme_get() to get the current **bayesplot** theme and bayesplot_theme_set() to set a new theme. bayesplot_theme_update() and bayesplot_theme_replace() are shorthands for changing individual elements.

Usage

```
bayesplot_theme_get()

bayesplot_theme_set(new = theme_default())

bayesplot_theme_update(...)

bayesplot_theme_replace(...)
```

Arguments

new	The new theme (list of theme elements) to use. This is analogous to the new argument to ggplot2::theme_set().
...	A named list of theme settings.

Details

bayesplot_theme_set() and friends only apply to bayesplots. However, ggplot2::theme_set() can also be used to change the **bayesplot** theme. Currently, setting a theme with ggplot2::theme_set() (other than the **ggplot2** default ggplot2::theme_grey()) will override the **bayesplot** theme.

Value

bayesplot_theme_get() returns the current theme. The other three functions (set, update, replace) invisibly return the *previous* theme so it can be saved and easily restored later. This is the same behavior as the **ggplot2** versions of these functions.

See Also

theme_default() for the default **bayesplot** theme.

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

```
library(ggplot2)

# plot using the current value of bayesplot_theme_get()
# (the default is bayesplot::theme_default())
x <- example_mcmc_draws()
mcmc_hist(x)

# change the bayesplot theme to theme_minimal and save the old theme
old <- bayesplot_theme_set(theme_minimal())
mcmc_hist(x)

# change back to the previous theme
bayesplot_theme_set(old)
mcmc_hist(x)

# change the default font size and family for bayesplots
bayesplot_theme_update(text = element_text(size = 16, family = "sans"))
mcmc_hist(x)

# change back to the default
bayesplot_theme_set() # same as bayesplot_theme_set(theme_default())
mcmc_hist(x)

# updating theme elements
color_scheme_set("brightblue")
bayesplot_theme_set(theme_dark())
mcmc_hist(x)

bayesplot_theme_update(panel.background = element_rect(fill = "black"))
mcmc_hist(x)

# to get the same plot without updating the theme we could also have
# used the bayeplot convenience function panel_bg()
bayesplot_theme_set(theme_dark())
mcmc_hist(x) + panel_bg(fill = "black")

# reset
bayesplot_theme_set()
```

MCMC-combos *Combination plots*

Description

Combination plots

Usage

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

Arguments

x	An object containing MCMC draws: • A 3-D array, matrix, list of matrices, or data frame. The MCMC-overview page provides details on how to specify each these. • A `draws` object from the **posterior** package (e.g., `draws_array`, `draws_rvars`, etc.). • An object with an `as.array()` method that returns the same kind of 3-D array described on the MCMC-overview page.
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 MCMC functions (omitting the `mcmc_` prefix).
...	Arguments passed to the plotting functions named in `combo`.
widths	A numeric vector the same length as `combo` specifying relative column widths. For example, if the plot has two columns, then `widths = c(2, 1)` will allocate more space for the first column by a factor of 2 (as would `widths = c(.3, .15)`, etc.). The default, `NULL`, allocates the same horizontal space for each column.
gg_theme	Unlike most of the other **bayesplot** functions, `mcmc_combo` returns a gtable object rather than a ggplot object, and so theme objects can't be added directly to the returned plot object. The `gg_theme` argument helps get around this problem by accepting a **ggplot2** theme object that is added to each of the plots *before* combining them into the gtable object that is returned. This can be a theme object created by a call to `ggplot2::theme()` or one of the **bayesplot** convenience functions, e.g. `legend_none()` (see the **Examples** section, below).

Value

A gtable object (the result of calling `gridExtra::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, bins = NULL, breaks = NULL)

mcmc_rhat_data(rhat, ...)
```

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

mcmc_neff_hist(ratio, ..., binwidth = NULL, bins = NULL, breaks = 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 R-hat estimates.
...	Currently ignored.
size	Optional values to override ggplot2::geom_point()'s default size (for mcmc_rhat(), mcmc_neff()) or ggplot2::geom_line()'s default line width (for mcmc_acf()).
binwidth	Passed to ggplot2::geom_histogram(), ggplot2::geom_area(), and ggdist::stat_dots() to override the default binwidth.
bins	Passed to ggplot2::geom_histogram() and ggplot2::geom_area() to override the default binning.
breaks	Passed to ggplot2::geom_histogram() as an alternative to binwidth.
ratio	A vector of *ratios* of effective sample size estimates to total sample size. See neff_ratio().
x	An object containing MCMC draws: • A 3-D array, matrix, list of matrices, or data frame. The MCMC-overview page provides details on how to specify each these. • A draws object from the **posterior** package (e.g., draws_array, draws_rvars, etc.). • An object with an as.array() method that returns the same kind of 3-D array described on the MCMC-overview page.

`pars` An optional character vector of parameter names. If neither `pars` nor `regex_pars` is specified then the default is to use *all* parameters. As of version `1.7.0`, **bayesplot** also supports 'tidy' parameter selection by specifying `pars = vars(...)`, where `...` is specified the same way as in dplyr::select(...) and similar functions. Examples of using `pars` in this way can be found on the Tidy parameter selection page.

`regex_pars` An optional regular expression to use for parameter selection. Can be specified instead of `pars` or in addition to `pars`. When using `pars` for tidy parameter selection, the `regex_pars` argument is ignored since select helpers perform a similar function.

`facet_args` A named list of arguments (other than `facets`) passed to `ggplot2::facet_wrap()` or `ggplot2::facet_grid()` to control faceting. Note: if `scales` is not included in `facet_args` then **bayesplot** may use `scales="free"` as the default (depending on the plot) instead of the **ggplot2** default of `scales="fixed"`.

`lags` The number of lags to show in the autocorrelation plot.

Value

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

Plot Descriptions

`mcmc_rhat_data()`, `mcmc_neff_data()` Data-preparation back ends for the R-hat and effective sample size plots. Users can call these functions directly to obtain the data frame of diagnostic values with rating labels and create custom diagnostic visualizations with **ggplot2**.

`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()`, `mcmc_acf_bar()` 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. *Stan Modeling Language Users Guide and Reference Manual.* https://mc-stan.org/users/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 horiztonal dashed line at 0.5
p + hline_at(0.5, linetype = 2, linewidth = 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, refresh = 0)
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)
```

`MCMC-distributions` *Histograms and kernel density plots of MCMC draws*

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,
  bins = NULL,
  breaks = NULL,
  freq = TRUE,
```

```
  alpha = 1
)

mcmc_dens(
  x,
  pars = character(),
  regex_pars = character(),
  transformations = list(),
  ...,
  facet_args = list(),
  trim = FALSE,
  bw = NULL,
  adjust = NULL,
  kernel = NULL,
  n_dens = NULL,
  bounds = NULL,
  alpha = 1
)

mcmc_hist_by_chain(
  x,
  pars = character(),
  regex_pars = character(),
  transformations = list(),
  ...,
  facet_args = list(),
  binwidth = NULL,
  bins = NULL,
  freq = TRUE,
  alpha = 1
)

mcmc_dens_overlay(
  x,
  pars = character(),
  regex_pars = character(),
  transformations = list(),
  ...,
  facet_args = list(),
  color_chains = TRUE,
  trim = FALSE,
  bw = NULL,
  adjust = NULL,
  kernel = NULL,
  n_dens = NULL,
  bounds = NULL
)
```

```
mcmc_dens_chains(
  x,
  pars = character(),
  regex_pars = character(),
  transformations = list(),
  ...,
  color_chains = TRUE,
  bw = NULL,
  adjust = NULL,
  kernel = NULL,
  n_dens = NULL,
  bounds = NULL
)

mcmc_dens_chains_data(
  x,
  pars = character(),
  regex_pars = character(),
  transformations = list(),
  ...,
  bw = NULL,
  adjust = NULL,
  kernel = NULL,
  n_dens = NULL,
  bounds = NULL
)

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

mcmc_dots(
  x,
  pars = character(),
  regex_pars = character(),
  transformations = list(),
  ...,
  facet_args = list(),
  binwidth = NA,
  alpha = 1,
  quantiles = 100
)
```

```
mcmc_dots_by_chain(
  x,
  pars = character(),
  regex_pars = character(),
  transformations = list(),
  ...,
  facet_args = list(),
  binwidth = NA,
  alpha = 1,
  quantiles = 100
)
```

Arguments

x	An object containing MCMC draws: • A 3-D array, matrix, list of matrices, or data frame. The MCMC-overview page provides details on how to specify each these. • A draws object from the **posterior** package (e.g., draws_array, draws_rvars, etc.). • An object with an as.array() method that returns the same kind of 3-D array described on the MCMC-overview page.
pars	An optional character vector of parameter names. If neither pars nor regex_pars is specified then the default is to use *all* parameters. As of version 1.7.0, **bayesplot** also supports 'tidy' parameter selection by specifying pars = vars(...), where ... is specified the same way as in dplyr::select(...) and similar functions. Examples of using pars in this way can be found on the Tidy parameter selection page.
regex_pars	An optional regular expression to use for parameter selection. Can be specified instead of pars or in addition to pars. When using pars for tidy parameter selection, the regex_pars argument is ignored since select helpers perform a similar function.
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 match.fun(), 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)").

Note: due to partial argument matching `transformations` can be abbreviated for convenience in interactive use (e.g., `transform`).

`...` For dot plots, optional additional arguments to pass to `ggdist::stat_dots()`.

`facet_args` A named list of arguments (other than `facets`) passed to `ggplot2::facet_wrap()` or `ggplot2::facet_grid()` to control faceting. Note: if `scales` is not included in `facet_args` then **bayesplot** may use `scales="free"` as the default (depending on the plot) instead of the **ggplot2** default of `scales="fixed"`.

`binwidth` Passed to `ggplot2::geom_histogram()`, `ggplot2::geom_area()`, and `ggdist::stat_dots()` to override the default binwidth.

`bins` Passed to `ggplot2::geom_histogram()` and `ggplot2::geom_area()` to override the default binning.

`breaks` Passed to `ggplot2::geom_histogram()` as an alternative to `binwidth`.

`freq` For histograms and frequency polygons, `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 `yaxis_text()` convenience function.)

`alpha` Passed to the geom to control the transparency.

`trim` A logical scalar passed to `ggplot2::geom_density()`.

`bw`, `adjust`, `kernel`, `n_dens`, `bounds`
Optional arguments passed to `stats::density()` (and `bounds` to `ggplot2::stat_density()`) to override default kernel density estimation parameters or truncate the density support. If `NULL` (default), `bw` is set to `"nrd0"`, `adjust` to `1`, `kernel` to `"gaussian"`, and `n_dens` to `1024`.

`color_chains` Option for whether to separately color chains.

`probs` A numeric vector of probabilities controlling where quantile lines are drawn. Set to `NULL` to remove the lines.

`quantiles` For dot plots, an optional integer passed to `ggdist::stat_dots()` specifying the number of quantiles to use for a quantile dot plot. If `quantiles` is `NA` then all data points are plotted. The default is `quantiles=100` so that each dot represent one percent of posterior mass.

Value

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

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_dots()` Dot 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_dots_by_chain()` Dot plots of posterior draws with chains separated via faceting.

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

`mcmc_dens_chains()` Ridgeline kernel density plots of posterior draws with chains separated but overlaid on a single plot. In `mcmc_dens_overlay()` parameters appear in separate facets; in `mcmc_dens_chains()` they appear in the same panel and can overlap vertically.

`mcmc_dens_chains_data()` Data-preparation back end for `mcmc_dens_chains()`. Users can call this function directly to obtain the prepared long-format data frame of MCMC draws (with chain information retained) and create custom visualizations with **ggplot2**.

See Also

Other MCMC: MCMC-combos, MCMC-diagnostics, MCMC-intervals, MCMC-nuts, MCMC-overview, MCMC-parcoord, MCMC-recover, MCMC-scatterplots, MCMC-traces

Examples

```
set.seed(9262017)
# 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(labeller = 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("orange")
mcmc_hist_by_chain(x, regex_pars = "beta")
```

```
#################
### Densities ###
#################
color_scheme_set("purple")
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)
x2 <- example_mcmc_draws(params = 6)
mcmc_dens_chains(x2, pars = c("beta[1]", "beta[2]", "beta[3]"))

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


#################
### Dot Plots ###
#################

# dot plots of some parameters
color_scheme_set("pink")
mcmc_dots(x, pars = c("alpha", "beta[2]"))


color_scheme_set("teal")
# separate dot plots by chain
mcmc_dots_by_chain(x, regex_pars = "beta")

# custom facet labels (will change row labels to e.g. "Chain: 1" instead of just "1")
chain_labeller <- ggplot2::labeller(.rows = ggplot2::label_both)
mcmc_dots_by_chain(x, regex_pars = "beta", facet_args = list(labeller = chain_labeller))
```

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"),
  outer_size = 0.5,
  inner_size = 2,
  point_size = 4,
  rhat = numeric()
)

mcmc_areas(
  x,
  pars = character(),
  regex_pars = character(),
  transformations = list(),
  ...,
  area_method = c("equal area", "equal height", "scaled height"),
  prob = 0.5,
  prob_outer = 1,
  point_est = c("median", "mean", "none"),
  rhat = numeric(),
  border_size = NULL,
  bw = NULL,
  adjust = NULL,
  kernel = NULL,
  n_dens = NULL,
  bounds = NULL
)

mcmc_areas_ridges(
  x,
  pars = character(),
  regex_pars = character(),
  transformations = list(),
  ...,
  prob_outer = 1,
  prob = 1,
  border_size = NULL,
  bw = NULL,
  adjust = NULL,
  kernel = NULL,
  n_dens = NULL,
```

```
  bounds = NULL
)

mcmc_intervals_data(
  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_data(
  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,
  n_dens = NULL,
  bounds = NULL
)

mcmc_areas_ridges_data(
  x,
  pars = character(),
  regex_pars = character(),
  transformations = list(),
  ...,
  prob_outer = 1,
  prob = 1,
  bw = NULL,
  adjust = NULL,
  kernel = NULL,
  n_dens = NULL,
  bounds = NULL
)
```

Arguments

x	An object containing MCMC draws: • A 3-D array, matrix, list of matrices, or data frame. The MCMC-overview page provides details on how to specify each these. • A draws object from the **posterior** package (e.g., draws_array, draws_rvars, etc.). • An object with an as.array() method that returns the same kind of 3-D array described on the MCMC-overview page.
pars	An optional character vector of parameter names. If neither pars nor regex_pars is specified then the default is to use *all* parameters. As of version 1.7.0, **bayesplot** also supports 'tidy' parameter selection by specifying pars = vars(...), where ... is specified the same way as in dplyr::select(...) and similar functions. Examples of using pars in this way can be found on the Tidy parameter selection page.
regex_pars	An optional regular expression to use for parameter selection. Can be specified instead of pars or in addition to pars. When using pars for tidy parameter selection, the regex_pars argument is ignored since select helpers perform a similar function.
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 match.fun(), 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)"). Note: due to partial argument matching transformations can be abbreviated for convenience in interactive use (e.g., transform).
...	Currently unused.
prob	The probability mass to include in the inner interval (for mcmc_intervals()) or in the shaded region (for mcmc_areas()). The default is 0.5 (50% interval) and 1 for mcmc_areas_ridges().
prob_outer	The probability mass to include in the outer interval. The default is 0.9 for mcmc_intervals() (90% interval) and 1 for mcmc_areas() and for mcmc_areas_ridges().
point_est	The point estimate to show. Either "median" (the default), "mean", or "none".
inner_size, outer_size	For mcmc_intervals(), the size of the inner and interval segments, respectively.

point_size For mcmc_intervals(), the size of point estimate.

rhat An optional numeric vector of R-hat 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 R-hat value. See rhat() for methods for extracting R-hat estimates.

area_method How to constrain the areas in mcmc_areas(). The default is "equal area", setting the density curves to have the same area. With "equal height", the curves are scaled so that the highest points across the curves are the same height. The method "scaled height" tries a compromise between to the two: the heights from "equal height" are scaled using height*sqrt(height)

border_size For mcmc_areas() and mcmc_areas_ridges(), the size of the ridgelines.

bw, adjust, kernel, n_dens, bounds
Optional arguments passed to stats::density() (and bounds to ggplot2::stat_density()) to override default kernel density estimation parameters or truncate the density support. If NULL (default), bw is set to "nrd0", adjust to 1, kernel to "gaussian", and n_dens to 1024.

Value

The plotting functions return a ggplot object that can be further customized using the **ggplot2** package. The functions with suffix _data() return the data that would have been drawn by the plotting function.

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.

mcmc_areas_ridges() Density plot, as in mcmc_areas(), but drawn with overlapping ridgelines. This plot provides a compact display of (hierarchically) related distributions.

mcmc_intervals_data(), mcmc_areas_data(), mcmc_areas_ridges_data() Data-preparation back ends for mcmc_intervals(), mcmc_areas(), and mcmc_areas_ridges(), respectively. Users can call these functions directly to obtain the prepared data frames of posterior interval summaries and create custom interval or density-area visualizations with **ggplot2**.

See Also

Other MCMC: MCMC-combos, MCMC-diagnostics, MCMC-distributions, MCMC-nuts, MCMC-overview, MCMC-parcoord, MCMC-recover, MCMC-scatterplots, MCMC-traces

Examples

```
set.seed(9262017)

# load ggplot2 to use its functions to modify our plots
library(ggplot2)
```

```
# 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) +
 labs(
   title = "Posterior distributions",
   subtitle = "with medians and 80% intervals"
 )

color_scheme_set("red")
p <- mcmc_areas(
   x,
   pars = c("alpha", "beta[4]"),
   prob = 2/3,
   prob_outer = 0.9,
   point_est = "mean",
   border_size = 1.5 # make the ridgelines fatter
)
plot(p)


# control spacing at top and bottom of plot
# see ?ggplot2::expansion
p + scale_y_discrete(
  limits = c("beta[4]", "alpha"),
  expand = expansion(add = c(1, 2))
)
p + scale_y_discrete(
  limits = c("beta[4]", "alpha"),
  expand = expansion(add = c(.1, .3))
)

# relabel parameters
p + scale_y_discrete(
  labels = c("alpha" = "param label 1",
             "beta[4]" = "param label 2")
)

# relabel parameters and define the order
p + scale_y_discrete(
  labels = c("alpha" = "param label 1",
             "beta[4]" = "param label 2"),
  limits = c("beta[4]", "alpha")
)

# 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)

# get the dataframe that is used in the plotting functions
mcmc_intervals_data(x)
mcmc_intervals_data(x, rhat = fake_rhat_values)
mcmc_areas_data(x, pars = "alpha")

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()



# Different area calculations
b3 <- c("beta[1]", "beta[2]", "beta[3]")

mcmc_areas(x, pars = b3, area_method = "equal area") +
  labs(
    title = "Curves have same area",
    subtitle = "A wide, uncertain interval is spread thin when areas are equal"
   )

mcmc_areas(x, pars = b3, area_method = "equal height") +
  labs(
    title = "Curves have same maximum height",
    subtitle = "Local curvature is clearer but more uncertain curves use more area"
  )

mcmc_areas(x, pars = b3, area_method = "scaled height") +
  labs(
    title = "Same maximum heights but heights scaled by square-root",
    subtitle = "Compromise: Local curvature is accentuated and less area is used"
   )


# 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,
 refresh = 0
)
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)

## Not run:
# Example of hierarchically related parameters
# plotted with ridgelines
m <- shinystan::eight_schools@posterior_sample
mcmc_areas_ridges(m, pars = "mu", regex_pars = "theta", border_size = 0.75) +
  ggtitle("Treatment effect on eight schools (Rubin, 1981)")

## End(Not run)
```

MCMC-nuts *Diagnostic plots for the No-U-Turn-Sampler (NUTS)*

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,
  bins = NULL,
  breaks = 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,
  bins = NULL,
  breaks = NULL,
  alpha = 0.5,
  merge_chains = FALSE
)
```

Arguments

x	A molten data frame of NUTS sampler parameters, either created by nuts_params() or in the same form as the object returned by nuts_params().
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. lp should either be created via log_posterior() or be an object with the same form as the object returned by log_posterior().
chain	A positive integer for selecting a particular chain. The default (NULL) is to merge the chains before plotting. If chain = k then the plot for chain k is overlaid (in a darker shade but with transparency) on top of the plot for all chains. For mcmc_nuts_stepsize(), chains are always plotted separately, and chain simply highlights the selected chain. The chain argument is not used by mcmc_nuts_energy().
...	Currently ignored.
binwidth	Passed to ggplot2::geom_histogram(), ggplot2::geom_area(), and ggdist::stat_dots() to override the default binwidth.
bins	Passed to ggplot2::geom_histogram() and ggplot2::geom_area() to override the default binning.
breaks	Passed to ggplot2::geom_histogram() as an alternative to binwidth.
alpha	For mcmc_nuts_energy() only, the transparency (alpha) level in [0,1] used for the overlaid histogram.
merge_chains	For mcmc_nuts_energy() 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 gridExtra::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. *Stan Modeling Language Users Guide and Reference Manual.* https://mc-stan.org/users/documentation/

See Also

- The Visual MCMC Diagnostics vignette.
- Several other plotting functions 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, refresh = 0)
np <- nuts_params(fit)
lp <- log_posterior(fit)

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

mcmc_nuts_divergence(np, lp)
mcmc_nuts_stepsize(np, lp)
mcmc_nuts_treedepth(np, lp)

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

## End(Not run)
```

MCMC-overview *Plots for Markov chain Monte Carlo simulations*

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 (2-D array)**: 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. When a `"Chain"` column is supplied, each chain must have the same number of iterations. Chain labels are used to identify groups and are renumbered internally to `1:N`.
- **draws**: Any of the `draws` formats supported by the **posterior** package.

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, optionally including HMC/NUTS diagnostic information.
- Scatterplots: Scatterplots, heatmaps, and pairs plots of parameter draws, optionally including HMC/NUTS diagnostic information.
- Parallel coordinates plots: Parallel coordinates plot of MCMC draws (one dimension per parameter), optionally including HMC/NUTS diagnostic information.
- Combos: Combination plots (e.g. trace plot + histogram).
- General MCMC diagnostics: MCMC diagnostic plots including R-hat, 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. and Gelman, A. (2019), Visualization in Bayesian workflow. *J. R. Stat. Soc. A*, 182: 389-402. doi:10.1111/rssa.12378. (journal version, arXiv preprint, code on GitHub)

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, and see Gabry et al. (2019) for more background and a real example.

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	An object containing MCMC draws: • A 3-D array, matrix, list of matrices, or data frame. The MCMC-overview page provides details on how to specify each these. • A draws object from the **posterior** package (e.g., draws_array, draws_rvars, etc.). • An object with an as.array() method that returns the same kind of 3-D array described on the MCMC-overview page.
pars	An optional character vector of parameter names. If neither pars nor regex_pars is specified then the default is to use *all* parameters. As of version 1.7.0, **bayesplot** also supports 'tidy' parameter selection by specifying pars = vars(...), where ... is specified the same way as in dplyr::select(...) and similar functions. Examples of using pars in this way can be found on the Tidy parameter selection page.

`regex_pars` An optional regular expression to use for parameter selection. Can be specified instead of `pars` or in addition to `pars`. When using `pars` for tidy parameter selection, the `regex_pars` argument is ignored since select helpers perform a similar function.

`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 `match.fun()`, 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)"`).

Note: due to partial argument matching `transformations` can be abbreviated for convenience in interactive use (e.g., `transform`).

`...` Currently ignored.

`size, alpha` Arguments passed on to `ggplot2::geom_line()`.

`np` For models fit using NUTS (more generally, any symplectic integrator), an optional data frame providing NUTS diagnostic information. The data frame should be the object returned by `nuts_params()` 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 `ggplot2::geom_line()` 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

The plotting functions return a ggplot object that can be further customized using the **ggplot2** package. The functions with suffix `_data()` 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. This version of the plot is the same as the parallel coordinates plot described in Gabry et al. (2019).

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 - mean(x))/sd(x) when specifying the transformations argument to mcmc_parcoord. See the **Examples** section for how to do this.

mcmc_parcoord_data() Data-preparation back end for mcmc_parcoord(). Users can call mcmc_parcoord_data() directly to obtain the prepared long-format data frame of MCMC draws (with optional NUTS diagnostic information) and create custom visualizations with **ggplot2**.

References

Gabry, J. , Simpson, D. , Vehtari, A. , Betancourt, M. and Gelman, A. (2019), Visualization in Bayesian workflow. *J. R. Stat. Soc. A*, 182: 389-402. doi:10.1111/rssa.12378. (journal version, arXiv preprint, code on GitHub)

Hartikainen, A. (2017, Aug 23). Concentration of divergences (Msg 21). Message posted to The Stan Forums: https://discourse.mc-stan.org/t/concentration-of-divergences/1590/21.

See Also

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., standardizing all the variables can be helpful)
# 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 conventient 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,
  bins = NULL,
  breaks = NULL
)
```

Arguments

x	An object containing MCMC draws: • A 3-D array, matrix, list of matrices, or data frame. The MCMC-overview page provides details on how to specify each these. • A draws object from the **posterior** package (e.g., draws_array, draws_rvars, etc.). • An object with an as.array() method that returns the same kind of 3-D array described on the MCMC-overview page.
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 **Examples** 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 y-axis.
...	Currently unused.
facet_args	A named list of arguments (other than facets) passed to ggplot2::facet_wrap() or ggplot2::facet_grid() to control faceting. Note: if scales is not included in facet_args then **bayesplot** may use scales="free" as the default (depending on the plot) instead of the **ggplot2** default of scales="fixed".
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 ggplot2::geom_point() to control the appearance of plotted points.
binwidth	Passed to ggplot2::geom_histogram(), ggplot2::geom_area(), and ggdist::stat_dots() to override the default binwidth.
bins	Passed to ggplot2::geom_histogram() and ggplot2::geom_area() to override the default binning.
breaks	Passed to ggplot2::geom_histogram() as an alternative to 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), refresh = 0)
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(),
  ...,
  shape = 21,
  size = 2.5,
  alpha = 0.8,
  np = NULL,
  np_style = scatter_style_np()
)

mcmc_hex(
  x,
  pars = character(),
  regex_pars = character(),
  transformations = list(),
```

```
  ...,
  bins = 30,
  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,
  grid_args = list(),
  save_gg_objects = TRUE
)

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	An object containing MCMC draws: • A 3-D array, matrix, list of matrices, or data frame. The MCMC-overview

page provides details on how to specify each these.

- A `draws` object from the **posterior** package (e.g., `draws_array`, `draws_rvars`, etc.).
- An object with an `as.array()` method that returns the same kind of 3-D array described on the MCMC-overview page.

`pars` An optional character vector of parameter names. If neither `pars` nor `regex_pars` is specified then the default is to use *all* parameters. As of version `1.7.0`, **bayesplot** also supports 'tidy' parameter selection by specifying `pars = vars(...)`, where `...` is specified the same way as in dplyr::select(...) and similar functions. Examples of using `pars` in this way can be found on the Tidy parameter selection page.

`regex_pars` An optional regular expression to use for parameter selection. Can be specified instead of `pars` or in addition to `pars`. When using `pars` for tidy parameter selection, the `regex_pars` argument is ignored since select helpers perform a similar function.

`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 `match.fun()`, 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)"`).

Note: due to partial argument matching `transformations` can be abbreviated for convenience in interactive use (e.g., `transform`).

`...` Currently ignored.

`shape`, `size`, `alpha`

For `mcmc_scatter()`, passed to `ggplot2::geom_point()` to control the appearance of the points.

`np` Optionally, a data frame of NUTS sampler parameters, either created by `nuts_params()` or in the same form as the object returned by `nuts_params()`. The colors, shapes, and sizes of the superimposed points can be customized using the `np_style` argument.

`np_style` If `np` is specified, `np_style` can be a call to the `scatter_style_np()` helper function (for `mcmc_scatter()`) or the `pairs_style_np()` helper function (for `mcmc_pairs()`) to specify arguments controlling the appearance of superimposed points representing NUTS diagnostic information. (Note: for `pairs_style_np()` the `size` arguments are interpreted as scaling factors).

`bins`, `binwidth` For `mcmc_hex()`, an optional numeric vector of *length two* passed to `ggplot2::geom_hex()` to override the default binwidth in both the vertical and horizontal directions.

diag_fun, off_diag_fun
: For mcmc_pairs(), the plotting function to use for the plots along the diagonal and for the off-diagonal plots, respectively. Currently diag_fun can be "hist" for histogram or "dens" for density, and off_diag_fun can be "scatter" for scatterplot or "hex" for a hexagonal heatmap.

diag_args, off_diag_args
: For mcmc_pairs(), optional named lists of arguments to pass to the functions implied by the diag_fun and off_diag_fun arguments, respectively. For example, if off_diag_fun is "scatter" then off_diag_args could include optional arguments to mcmc_scatter() like size and alpha.

condition
: For mcmc_pairs(), a call to the pairs_condition() 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 condition. The default is a call to pairs_condition() 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 chains, draws, and nuts arguments to pairs_condition(), which are documented below, can be used to change this default.

lp
: For mcmc_pairs(), 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. lp should either be created via log_posterior() or be an object with the same form as the object returned by log_posterior().

max_treedepth
: For mcmc_pairs(), 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.

grid_args, save_gg_objects
: For mcmc_pairs(), arguments to pass to bayesplot_grid(). For example, since mcmc_pairs() returns more than a single ggplot object, using ggtitle() afterwards will not work. But you you can still add a title to the plot using grid_args = list(top="My title").

div_color, div_shape, div_size, div_alpha, td_color, td_shape, td_size, td_alpha
: Optional arguments to the scatter_style_np() or pairs_style_np() helper functions that are eventually passed to ggplot2::geom_point().The default values are displayed in the **Usage** section above.

chains, draws, nuts
: Optional arguments to the pairs_condition() helper function, which is used to specify the condition argument for mcmc_pairs().

  - The chains argument can be used to select some subset of the chains. If chains 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, chains 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.

- The `draws` argument to `pairs_condition()` can be used to directly specify which realizations are plotted above and below the diagonal. `draws` 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) 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). For more on why the scatter plot with divergences is a useful diagnostic tool see Gabry et al. (2019).

`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. and Gelman, A. (2019), Visualization in Bayesian workflow. *J. R. Stat. Soc. A*, 182: 389-402. doi:10.1111/rssa.12378. (journal version, arXiv preprint, code on GitHub)

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", linewidth = 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", linewidth = .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",
                 linewidth = .75, linetype = 2)
```

```
if (requireNamespace("hexbin", quietly = TRUE)) {
 # 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, refresh = 0,
  # 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 and rank plots of MCMC draws*

Description

Trace and rank plots 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,
  iter1 = 0,
  window = NULL,
  size = NULL,
  alpha = 0.4,
  highlight = 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)

mcmc_rank_overlay(
  x,
  pars = character(),
  regex_pars = character(),
  transformations = list(),
  facet_args = list(),
  ...,
  n_bins = 20,
  ref_line = FALSE
)

mcmc_rank_hist(
  x,
  pars = character(),
  regex_pars = character(),
```

```
  transformations = list(),
  ...,
  facet_args = list(),
  n_bins = 20,
  ref_line = FALSE
)

mcmc_rank_ecdf(
  x,
  pars = character(),
  regex_pars = character(),
  transformations = list(),
  ...,
  K = NULL,
  facet_args = list(),
  prob = 0.99,
  plot_diff = FALSE,
  interpolate_adj = NULL
)

mcmc_trace_data(
  x,
  pars = character(),
  regex_pars = character(),
  transformations = list(),
  ...,
  highlight = NULL,
  n_warmup = 0,
  iter1 = 0
)
```

Arguments

x	An object containing MCMC draws: • A 3-D array, matrix, list of matrices, or data frame. The MCMC-overview page provides details on how to specify each these. • A draws object from the **posterior** package (e.g., draws_array, draws_rvars, etc.). • An object with an as.array() method that returns the same kind of 3-D array described on the MCMC-overview page.
pars	An optional character vector of parameter names. If neither pars nor regex_pars is specified then the default is to use *all* parameters. As of version 1.7.0, **bayesplot** also supports 'tidy' parameter selection by specifying pars = vars(...), where ... is specified the same way as in dplyr::select(...) and similar functions. Examples of using pars in this way can be found on the Tidy parameter selection page.
regex_pars	An optional regular expression to use for parameter selection. Can be specified instead of pars or in addition to pars. When using pars for tidy parameter

selection, the regex_pars argument is ignored since select helpers perform a similar function.

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 match.fun(), 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)").

Note: due to partial argument matching transformations can be abbreviated for convenience in interactive use (e.g., transform).

...

Currently ignored.

facet_args

A named list of arguments (other than facets) passed to ggplot2::facet_wrap() or ggplot2::facet_grid() to control faceting. Note: if scales is not included in facet_args then **bayesplot** may use scales="free" as the default (depending on the plot) instead of the **ggplot2** default of scales="fixed".

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.

iter1

An integer; the iteration number of the first included draw (default is 0). This can be used to make it more obvious that the warmup iterations have been discarded from the traceplot. It cannot be specified if n_warmup is also set to a positive value.

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 for mcmc_trace() or the default point size for mcmc_trace_highlight().

alpha

For mcmc_trace() and mcmc_trace_highlight(), controls the transparency of the lines or points for the chains not highlighted.

highlight

For mcmc_trace(), NULL (the default) or an integer specifying one chain to emphasize. For mcmc_trace_highlight(), an integer specifying one chain to emphasize.

np

For models fit using NUTS (more generally, any symplectic integrator), an optional data frame providing NUTS diagnostic information. The data frame should be the object returned by nuts_params() 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 **Examples** 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.

`div_color`, `div_size`, `div_alpha`
Optional arguments to the `trace_style_np()` helper function that are eventually passed to `ggplot2::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.

`n_bins` For the rank plots, the number of bins to use for the histogram of rank-normalized MCMC samples. Defaults to `20`.

`ref_line` For the rank plots, whether to draw a horizontal line at the average number of ranks per bin. Defaults to `FALSE`.

`K` An optional integer defining the number of equally spaced evaluation points for the PIT-ECDF. Reducing K when using `interpolate_adj = FALSE` makes computing the confidence bands faster. For `ppc_pit_ecdf()` and `ppc_pit_ecdf_grouped()` when `method = 'independent'`. If `pit` is supplied, defaults to `length(pit)`, otherwise `yrep` determines the maximum accuracy of the estimated PIT values and `K` is set to `min(nrow(yrep) + 1, 1000)`. For `mcmc_rank_ecdf()`, defaults to the number of iterations per chain in `x`.

`prob` For `mcmc_rank_ecdf()`, a value between 0 and 1 specifying the desired simultaneous confidence of the confidence bands to be drawn for the rank ECDF plots.

`plot_diff` For `mcmc_rank_ecdf()`, a boolean specifying if the difference between the observed rank ECDFs and the theoretical expectation should be drawn instead of the unmodified rank ECDF plots.

`interpolate_adj`
A boolean defining if the simultaneous confidence bands should be interpolated based on precomputed values rather than computed exactly. Computing the bands may be computationally intensive and the approximation gives a fast method for assessing the ECDF trajectory. For `ppc_pit_ecdf()` and `ppc_pit_ecdf_grouped()` when `method = 'independent'` and for `mcmc_rank_ecdf()`. The default is to use interpolation if `K` is greater than 200.

Value

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

`mcmc_trace_data()` returns the data for the trace *and* rank plots in the same data frame.

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. One chain can be emphasized using the `highlight` argument.

`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.

`mcmc_rank_hist()` Whereas traditional trace plots visualize how the chains mix over the course of sampling, rank histograms visualize how the values from the chains mix together in terms of ranking. An ideal plot would show the rankings mixing or overlapping in a uniform distribution. See Vehtari et al. (2019) for details.

`mcmc_rank_overlay()` Ranks from `mcmc_rank_hist()` are plotted using overlaid lines in a single panel.

`mcmc_rank_ecdf()` The ECDFs of the ranks from `mcmc_rank_hist()` are plotted with the simultaneous confidence bands with a coverage determined by `prob`, that is, bands that completely cover all of the rank ECDFs with the probability `prob`. If `plot_diff = TRUE`, the difference between the observed rank ECDFs and the theoretical expectation for samples originating from the same distribution is drawn. See Säilynoja et al. (2022) for details.

`mcmc_trace_data()` Data-preparation back end for `mcmc_trace()`, `mcmc_trace_highlight()`, `mcmc_rank_hist()`, `mcmc_rank_overlay()`, and `mcmc_rank_ecdf()`. The returned data frame contains columns for both the original draw values and their within-parameter ranks, so it can be used to build both trace and rank-based visualizations with **ggplot2**.

References

Vehtari, A., Gelman, A., Simpson, D., Carpenter, B., Bürkner, P. (2019). Rank-normalization, folding, and localization: An improved *R*-hat for assessing convergence of MCMC. arXiv preprint.

Säilynoja, T., Bürkner, P.-C., and Vehtari, A. (2022). Graphical test for discrete uniformity and its applications in goodness-of-fit evaluation and multiple sample comparison. *Statistics and Computing*. 32(2), 32. doi:10.1007/s11222-022-10090-6. arXiv preprint: https://arxiv.org/abs/2103.10522

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")
mcmc_trace(x, regex_pars = "beta", highlight = 2)

# 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")

# Rank-normalized histogram plots. Instead of showing how chains mix over
# time, look at how the ranking of MCMC samples mixed between chains.
color_scheme_set("viridisE")
mcmc_rank_hist(x, "alpha")
mcmc_rank_hist(x, pars = c("alpha", "sigma"), ref_line = TRUE)
mcmc_rank_overlay(x, "alpha")

# ECDF and ECDF difference plots of the ranking of MCMC samples between chains.
# Provide 99% simultaneous confidence intervals for the chains sampling from
# the same distribution.
mcmc_rank_ecdf(x, prob = 0.99)
mcmc_rank_ecdf(x, prob = 0.99, plot_diff = TRUE)

## Not run:
# parse facet label text
color_scheme_set("purple")
p <- mcmc_trace(
  x,
  regex_pars = "beta\\[[1,3]\\]",
  facet_args = list(labeller = 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, refresh = 0,
  # 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)
)

## End(Not run)
```

---

pp_check *Posterior (or prior) 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 (and prior) 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 PPC functions. 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")
```

PPC-calibration *PPC calibration*

Description

Assess the calibration of the predictions, or predictive probabilities in relation to binary observations. See the **Plot Descriptions** and **Details** sections below and also the PPC Calibration vignette for more details and examples.

Usage

```
ppc_calibration_overlay(y, prep, ..., linewidth = 0.25, alpha = 0.2)

ppc_calibration_overlay_grouped(
  y,
  prep,
  group,
  ...,
  linewidth = 0.25,
  alpha = 0.2
)

ppc_calibration(
  y,
  prep = NULL,
  yrep = NULL,
  prob = 0.95,
  interval = c("confidence", "consistency"),
  help_text = TRUE,
  B = 200,
  show_mean = TRUE,
  show_qdots = TRUE,
  qdots_quantiles = 100,
  ...,
  linewidth = 1,
  alpha = 0.1
)

ppc_calibration_grouped(
  y,
  yrep = NULL,
  prep = NULL,
  group,
  prob = 0.95,
  interval = c("confidence", "consistency"),
  help_text = TRUE,
  B = 200,
  show_mean = TRUE,
  show_qdots = TRUE,
  qdots_quantiles = 100,
  ...,
  linewidth = 1,
  alpha = 0.1
)

ppc_loo_calibration(
  y,
  yrep,
```

```
  lw = NULL,
  psis_object = NULL,
  prob = 0.95,
  interval = c("confidence", "consistency"),
  help_text = TRUE,
  B = 200,
  show_mean = TRUE,
  show_qdots = TRUE,
  qdots_quantiles = 100,
  ...,
  linewidth = 1,
  alpha = 0.1
)

ppc_loo_calibration_grouped(
  y,
  yrep,
  lw = NULL,
  psis_object = NULL,
  group,
  prob = 0.95,
  interval = c("confidence", "consistency"),
  help_text = TRUE,
  B = 200,
  show_mean = TRUE,
  show_qdots = TRUE,
  qdots_quantiles = 100,
  ...,
  linewidth = 1,
  alpha = 0.1
)

ppc_calibration_data(
  y,
  prep = NULL,
  yrep = NULL,
  group = NULL,
  type = c("overlay", "interval"),
  prob = 0.95,
  interval = c("confidence", "consistency"),
  B = 200
)
```

Arguments

y	A vector of observations. See **Details**.
prep	For ppc_calibration(), ppc_calibration_grouped(), ppc_calibration_overlay(), and ppc_calibration_overlay_grouped(), an S by N matrix of predicted

probabilities in `[0, 1]`, where `S` is the number of draws and `N` the number of observations (`N = length(y)`).

`...` Currently unused.

`linewidth, alpha`
Arguments passed to geoms controlling line width and opacity.

`group` A grouping variable of the same length as `y`. Will be coerced to factor if not already a factor. Each value in `group` is interpreted as the group level pertaining to the corresponding observation.

`yrep` An `S` by `N` matrix of draws from the posterior (or prior) predictive distribution, or a `posterior::draws` object. The number of rows, `S`, is the size of the posterior (or prior) sample used to generate `yrep`. The number of columns, `N` is the number of predicted observations (`length(y)`). The columns of `yrep` should be in the same order as the data points in `y` for the plots to make sense. See the **Details** and **Plot Descriptions** sections for additional advice specific to particular plots.

`prob` For `ppc_calibration()`, `ppc_calibration_grouped()`, `ppc_loo_calibration()`, and `ppc_loo_calibration_grouped()`. Probability used to compute the uncertainty intervals. Defaults to `0.95`.

`interval` For `ppc_calibration()`, `ppc_calibration_grouped()`, `ppc_loo_calibration()`, and `ppc_loo_calibration_grouped()`, pointwise uncertainty interval around the calibration curve. Choose `"confidence"` (default) to answer the question: "Where does the calibration curve of the model lie?" or `"consistency"` to answer the question: "If the model is correctly specified, where would we expect the calibration curve to fall?".

`help_text` For `ppc_calibration()`, `ppc_calibration_grouped()`, `ppc_loo_calibration()`, and `ppc_loo_calibration_grouped()`, if `TRUE` (default) display a label in the plot indicating the interval type as `CI` (confidence) or `CsI` (consistency) with the selected `prob`.

`B` For `ppc_calibration()`, `ppc_calibration_grouped()`, `ppc_loo_calibration()`, and `ppc_loo_calibration_grouped()` that use `yrep` with `interval = "confidence"`, the number of bootstrap samples. Default is `200`. Ignored if `prep` is used or `interval = "consistency"`.

`show_mean` For `ppc_calibration()`, `ppc_calibration_grouped()`, `ppc_loo_calibration()`, and `ppc_loo_calibration_grouped()`, if `TRUE` (default), draw the estimated calibration curve.

`show_qdots` For `ppc_calibration()`, `ppc_calibration_grouped()`, `ppc_loo_calibration()`, and `ppc_loo_calibration_grouped()`, if `TRUE` (default) add a quantile dot plot at the bottom of the panel to show the marginal distribution of predicted probabilities.

`qdots_quantiles`
For `ppc_calibration()`, `ppc_calibration_grouped()`, `ppc_loo_calibration()`, and `ppc_loo_calibration_grouped()`, positive integer indicating the number of dots in the quantile dot plot. Default is `100`.

`lw` For `ppc_loo_calibration()` and `ppc_loo_calibration_grouped()`, a matrix of log weights with the same dimensions as `yrep`. Either `psis_object` or `lw` has to be specified.

`psis_object` For `ppc_loo_calibration()` and `ppc_loo_calibration_grouped()`, an object of class `"psis"` that is created when the `loo()` function calls `psis()` internally to do the PSIS procedure. Either `psis_object` or `lw` has to be specified.

`type` For `ppc_calibration_data()`, the data structure to compute: `"overlay"` for `ppc_calibration_overlay()` or `"interval"` for `ppc_calibration()` and their corresponding _grouped and _loo variants.

Details

The PPC calibration functions are designed to assess the calibration of a model with binary outcomes. In this context, calibration refers to the agreement between predicted probabilities and conditional event probabilities (CEPs) see Dimitriadis et al. (2021) and Säilynoja et al. (2026) for details.

The required inputs are `y`, representing binary observations (0 or 1), and either `yrep` or `prep`. Specifically, `ppc_calibration_overlay()` and `ppc_calibration_overlay_grouped()` require `prep`, while `ppc_calibration()`, `ppc_calibration_grouped()`, `ppc_loo_calibration()`, and `ppc_loo_calibration_grouped()` accept either `prep` or `yrep`.

`prep` or `yrep`.

A document with detailed explanations and examples is available in the vignettes.

Value

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

Plot Descriptions

`ppc_calibration()`,`ppc_calibration_grouped()` PAV-adjusted calibration plots showing the relationship between the predicted event probabilities and the conditional event probabilities. The `interval` parameter controls whether confidence intervals, or consistency intervals are computed around the calibration curve.

`ppc_calibration_overlay()`,`ppc_calibration_overlay_grouped()` Overlay plots showing posterior samples of PAV-adjusted calibration curves for each posterior draw, which can be used to visually assess the uncertainty in the calibration curve.

`ppc_loo_calibration()`,`ppc_loo_calibration_grouped()` PAV-adjusted calibration plots to assess the calibration of the leave-one-out (LOO) predictive probabilities, computed by resampling each observation's posterior predictive draws using LOO importance weights.

`ppc_calibration_data()` Data frame containing the data underlying the calibration plots, which can be used to build custom calibration plots. The `type` argument controls whether the data frame for `ppc_calibration_overlay()` and its `_grouped`` variant is computed (type = "overlay"), or the data frame for ppc_calibration()and its_groupedor_loo variant is computed (type = "interval"‘).

References

Dimitriadis, T., Gneiting, T., & Jordan, A. I. (2021). Stable reliability diagrams for probabilistic classifiers. Proceedings of the National Academy of Sciences, 118(8). https://doi.org/10.1073/pnas.2016191118

Säilynoja, T., Johnson, A., Martin, O., and Vehtari, A. (2026). Recommendations for visual predictive checks in Bayesian workflow. *Journal of Visualization and Interaction*. 1(1). doi:10.54337/jovi.v1i1.11478.

See Also

Other PPCs: PPC-censoring, PPC-discrete, PPC-distributions, PPC-errors, PPC-intervals, PPC-loo, PPC-overview, PPC-scatterplots, PPC-test-statistics

Examples

```
color_scheme_set("brightblue")

# Make an example dataset of binary observations
ymin <- range(example_y_data(), example_yrep_draws())[1]
ymax <- range(example_y_data(), example_yrep_draws())[2]
y <- rbinom(length(example_y_data()), 1, (example_y_data() - ymin) / (ymax - ymin))
prep <- (example_yrep_draws() - ymin) / (ymax - ymin)

ppc_calibration_overlay(y, prep[1:50, ])

# Compare confidence vs consistency intervals
ppc_calibration(y, prep, interval = "confidence")
ppc_calibration(y, prep, interval = "consistency")
```

PPC-censoring *PPC censoring*

Description

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

Although some of the other **bayesplot** plots can be used with censored data, `ppc_km_overlay()` is currently the only plotting function designed *specifically* for censored data. We encourage you to suggest or contribute additional plots at github.com/stan-dev/bayesplot.

Usage

```
ppc_km_overlay(
  y,
  yrep,
  ...,
  status_y,
  left_truncation_y = NULL,
```

```
  extrapolation_factor = 1.2,
  y_draw = c("lines", "points"),
  size = 0.25,
  alpha = 0.7
)

ppc_km_overlay_grouped(
  y,
  yrep,
  group,
  ...,
  status_y,
  left_truncation_y = NULL,
  extrapolation_factor = 1.2,
  y_draw = c("lines", "points"),
  size = 0.25,
  alpha = 0.7
)
```

Arguments

y	A vector of observations. See **Details**.
yrep	An S by N matrix of draws from the posterior (or prior) predictive distribution, or a posterior::draws object. The number of rows, S, is the size of the posterior (or prior) sample used to generate yrep. The number of columns, N is the number of predicted observations (length(y)). The columns of yrep should be in the same order as the data points in y for the plots to make sense. See the **Details** and **Plot Descriptions** sections for additional advice specific to particular plots.
...	Currently only used internally.
status_y	The status indicator for the observations from y. This must be a numeric vector of the same length as y with values in {0, 1} (0 = right censored, 1 = event).
left_truncation_y	Optional parameter that specifies left-truncation (delayed entry) times for the observations from y. This must be a numeric vector of the same length as y. If NULL (default), no left-truncation is assumed.
extrapolation_factor	A numeric value (>=1) that controls how far the plot is extended beyond the largest observed value in y. The default value is 1.2, which corresponds to 20 % extrapolation. Note that all posterior predictive draws may not be shown by default because of the controlled extrapolation. To display all posterior predictive draws, set extrapolation_factor = Inf.
y_draw	How should the observed data be plotted? Possible values are "lines" and "points". If "lines" (default), event times and censoring times are connected with lines. If "points", event times are marked as points and censoring times are marked as plus signs.
size, alpha	Passed to the appropriate geom to control the appearance of the yrep distributions.

group — A grouping variable of the same length as y. Will be coerced to factor if not already a factor. Each value in group is interpreted as the group level pertaining to the corresponding observation.

Value

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

Plot Descriptions

ppc_km_overlay() Empirical CCDF estimates of each dataset (row) in yrep are overlaid, with the Kaplan-Meier estimate (Kaplan and Meier, 1958) for y itself on top (and in a darker shade). This is a PPC suitable for right-censored y. Note that the replicated data from yrep is assumed to be uncensored. Left truncation (delayed entry) times for y can be specified using left_truncation_y.

ppc_km_overlay_grouped() The same as ppc_km_overlay(), but with separate facets by group.

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)

Kaplan, E. L. and Meier, P. (1958). Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*. 53(282), 457–481. doi:10.1080/01621459.1958.10501452.

See Also

Other PPCs: PPC-calibration, PPC-discrete, PPC-distributions, PPC-errors, PPC-intervals, PPC-loo, PPC-overview, PPC-scatterplots, PPC-test-statistics

Examples

```
color_scheme_set("brightblue")

# For illustrative purposes, (right-)censor values y > 110:
y <- example_y_data()
status_y <- as.numeric(y <= 110)
y <- pmin(y, 110)

# In reality, the replicated data (yrep) would be obtained from a
# model which takes the censoring of y properly into account. Here,
# for illustrative purposes, we simply use example_yrep_draws():
yrep <- example_yrep_draws()
dim(yrep)

# Overlay 25 curves
ppc_km_overlay(y, yrep[1:25, ], status_y = status_y)

# With extrapolation_factor = 1 (no extrapolation)
ppc_km_overlay(y, yrep[1:25, ], status_y = status_y, extrapolation_factor = 1)

# With extrapolation_factor = Inf (show all posterior predictive draws)
```

```
ppc_km_overlay(y, yrep[1:25, ], status_y = status_y, extrapolation_factor = Inf)

# With separate facets by group:
group <- example_group_data()
ppc_km_overlay_grouped(y, yrep[1:25, ], group = group, status_y = status_y)

# With left-truncation (delayed entry) times:
min_vals <- pmin(y, apply(yrep, 2, min))
left_truncation_y <- rep(0, length(y))
condition <- y > mean(y) / 2
left_truncation_y[condition] <- pmin(
  runif(sum(condition), min = 0.6, max = 0.99) * y[condition],
  min_vals[condition] - 0.001
)
ppc_km_overlay(y, yrep[1:25, ], status_y = status_y,
               left_truncation_y = left_truncation_y)

# With y_draw = "points"
ppc_km_overlay(y, yrep[1:25, ], status_y = status_y, y_draw = "points")
```

PPC-discrete *PPCs for discrete outcomes*

Description

Many of the PPC functions in **bayesplot** can be used with discrete data. The small subset of these functions that can *only* be used if `y` and `yrep` are discrete are documented on this page. 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 = 2.5,
  linewidth = 1,
  freq = TRUE
)

ppc_bars_grouped(
  y,
  yrep,
  group,
```

```
  ...,
  facet_args = list(),
  prob = 0.9,
  width = 0.9,
  size = 1,
  fatten = 2.5,
  linewidth = 1,
  freq = TRUE
)

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

ppc_rootogram_grouped(
  y,
  yrep,
  group,
  style = c("standing", "hanging", "suspended", "discrete"),
  ...,
  facet_args = list(),
  prob = 0.9,
  size = 1,
  bound_distinct = TRUE
)

ppc_bars_data(y, yrep, group = NULL, prob = 0.9, freq = TRUE)
```

Arguments

y	A vector of observations. See **Details**.
yrep	An S by N matrix of draws from the posterior (or prior) predictive distribution, or a posterior::draws object. The number of rows, S, is the size of the posterior (or prior) sample used to generate yrep. The number of columns, N is the number of predicted observations (length(y)). The columns of yrep should be in the same order as the data points in y for the plots to make sense. See the **Details** and **Plot Descriptions** sections for additional advice specific to particular plots.
...	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. (Note: for rootograms these are intervals of the *square roots* of the expected counts.)
width	For bar plots only, passed to ggplot2::geom_bar() to control the bar width.

`size`, `fatten`, `linewidth`
: For bar plots, `size`, `fatten`, and `linewidth` are passed to `ggplot2::geom_pointrange()` to control the appearance of the `yrep` points and intervals. For rootograms `size` is passed to `ggplot2::geom_line()` and `ggplot2::geom_pointrange()`.

`freq`
: For bar plots only, if `TRUE` (the default) the y-axis will display counts. Setting `freq=FALSE` will put proportions on the y-axis.

`group`
: A grouping variable of the same length as `y`. Will be coerced to factor if not already a factor. Each value in `group` is interpreted as the group level pertaining to the corresponding observation.

`facet_args`
: An optional list of arguments (other than `facets`) passed to `ggplot2::facet_wrap()` to control faceting.

`style`
: For `ppc_rootogram`, a string specifying the rootogram style. The options are `"discrete"`, `"standing"`, `"hanging"`, and `"suspended"`. See the **Plot Descriptions** section, below, for details on the different styles.

`bound_distinct`
: For `ppc_rootogram(style = "discrete)` and `ppc_rootogram_grouped(style = "discrete)`, if `TRUE` then the observed counts will be plotted with different shapes depending on whether they are within the bounds of the `y` quantiles.

Details

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

Value

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

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_bars_data()` Data-preparation back end for `ppc_bars()` and `ppc_bars_grouped()`. Users can call `ppc_bars_data()` directly to obtain the prepared data frame and create custom visualizations with **ggplot2**.

`ppc_rootogram()` Rootograms allow for diagnosing problems in count data models such as overdispersion or excess zeros. In `standing`, `hanging`, and `suspended` styles, they consist of a histogram of `y` with the expected counts based on `yrep` overlaid as a line along with uncertainty intervals.

Meanwhile, in `discrete` style, median counts based on `yrep` are laid as a point range with uncertainty intervals along with dots representing the `y`.

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 rootogram 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 hanging from the curve representing expected counts.
- *Suspended*: histogram of the differences between expected and observed counts.
- *Discrete*: a dot-and-whisker plot of the median counts and dots representing observed counts.

As it emphasizes the discrete nature of the count data, using `discrete` style is suggested.

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

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

Related functions

In addition to the functions on this page that are restricted to discrete outcomes, some general PPC/PPD functions also support discrete data when requested:

- `ppc_stat()` and `ppc_stat_grouped()` can visualize discrete test statistics with predictive checks when `discrete = TRUE`.
- `ppd_stat()` and `ppd_stat_grouped()` can visualize discrete test statistics from predictive draws when `discrete = TRUE`.
- ppc_ecdf_overlay can visualize empirical CDFs for discrete statistics with `discrete = TRUE`.
- `ppc_pit_ecdf()` and `ppc_pit_ecdf_grouped()` can also handle discrete variables to plot PIT-ECDF of the empirical PIT values.

These functions are not limited to discrete outcomes, but offer discrete-friendly displays for integer-valued statistics.

References

Säilynoja, T., Johnson, A., Martin, O., and Vehtari, A. (2026). Recommendations for visual predictive checks in Bayesian workflow. *Journal of Visualization and Interaction*. 1(1). doi:10.54337/jovi.v1i1.11478.

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-calibration`, `PPC-censoring`, `PPC-distributions`, `PPC-errors`, `PPC-intervals`, `PPC-loo`, `PPC-overview`, `PPC-scatterplots`, `PPC-test-statistics`

Examples

```
set.seed(9222017)

# 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)

## Not run:
# example for ordinal regression using rstanarm
library(rstanarm)
fit <- stan_polr(
  tobgp ~ agegp,
  data = esoph,
  method = "probit",
  prior = R2(0.2, "mean"),
  init_r = 0.1,
  seed = 12345,
  # cores = 4,
  refresh = 0
 )

# coded as character, so convert to integer
yrep_char <- posterior_predict(fit)
print(yrep_char[1, 1:4])

yrep_int <- sapply(data.frame(yrep_char, stringsAsFactors = TRUE), as.integer)
y_int <- as.integer(esoph$tobgp)

ppc_bars(y_int, yrep_int)

ppc_bars_grouped(
  y = y_int,
  yrep = yrep_int,
  group = esoph$agegp,
  freq=FALSE,
  prob = 0.5,
  fatten = 1,
  size = 1.5
)

## End(Not run)

# 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_rootogram(y, yrep, style = "discrete")

# rootograms for counts with groups
group <- gl(2, 50, length = 100, labels = c("GroupA", "GroupB"))
ppc_rootogram_grouped(y, yrep, group)
ppc_rootogram_grouped(y, yrep, group, style = "hanging", facet_args = list(nrow = 2))
ppc_rootogram_grouped(y, yrep, group, style = "discrete", prob = 0.5)
```

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_data(y, yrep, group = NULL)

ppc_dens_overlay(
  y,
  yrep,
  ...,
  size = 0.25,
  alpha = 0.7,
  trim = FALSE,
  bw = NULL,
  adjust = NULL,
  kernel = NULL,
  bounds = NULL,
  n_dens = NULL
)

ppc_dens_overlay_grouped(
  y,
  yrep,
  group,
  ...,
  size = 0.25,
  alpha = 0.7,
  trim = FALSE,
```

```
  bw = NULL,
  adjust = NULL,
  kernel = NULL,
  bounds = NULL,
  n_dens = NULL
)

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

ppc_ecdf_overlay_grouped(
  y,
  yrep,
  group,
  ...,
  discrete = deprecated(),
  pad = TRUE,
  size = 0.25,
  alpha = 0.7
)

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

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

ppc_freqpoly(
  y,
  yrep,
  ...,
  binwidth = NULL,
  bins = NULL,
  freq = TRUE,
  size = 0.5,
```

```
  alpha = 1
)

ppc_freqpoly_grouped(
  y,
  yrep,
  group,
  ...,
  binwidth = NULL,
  bins = NULL,
  freq = TRUE,
  size = 0.5,
  alpha = 1
)

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

ppc_dots(y, yrep, ..., binwidth = NA, quantiles = 100, freq = TRUE)

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
)

ppc_pit_ecdf(
  y,
  yrep,
  ...,
  pit = NULL,
  K = NULL,
  prob = 0.99,
  plot_diff = FALSE,
  interpolate_adj = NULL,
  method = NULL,
  test = NULL,
  gamma = NULL,
  linewidth = NULL,
  color = NULL,
  help_text = NULL,
```

```
  pareto_pit = NULL,
  help_text_shrinkage = NULL
)

ppc_pit_ecdf_grouped(
  y,
  yrep,
  group,
  ...,
  K = NULL,
  pit = NULL,
  prob = 0.99,
  plot_diff = FALSE,
  interpolate_adj = NULL,
  method = NULL,
  test = NULL,
  gamma = NULL,
  linewidth = NULL,
  color = NULL,
  help_text = NULL,
  pareto_pit = NULL,
  help_text_shrinkage = NULL
)
```

Arguments

y	A vector of observations. See **Details**.
yrep	An S by N matrix of draws from the posterior (or prior) predictive distribution, or a posterior::draws object. The number of rows, S, is the size of the posterior (or prior) sample used to generate yrep. The number of columns, N is the number of predicted observations (length(y)). The columns of yrep should be in the same order as the data points in y for the plots to make sense. See the **Details** and **Plot Descriptions** sections for additional advice specific to particular plots.
group	A grouping variable of the same length as y. Will be coerced to factor if not already a factor. Each value in group is interpreted as the group level pertaining to the corresponding observation.
...	For dot plots, optional additional arguments to pass to ggdist::stat_dots().
size, alpha	Passed to the appropriate geom to control the appearance of the predictive distributions.
trim	A logical scalar passed to ggplot2::geom_density().
bw, adjust, kernel, n_dens, bounds	Optional arguments passed to stats::density() (and bounds to ggplot2::stat_density()) to override default kernel density estimation parameters or truncate the density support. If NULL (default), bw is set to "nrd0", adjust to 1, kernel to "gaussian", and n_dens to 1024.
discrete	**[Deprecated]** The discrete argument is deprecated. The ECDF is a step function by definition, so geom_step() is now always used.

pad — A logical scalar passed to ggplot2::stat_ecdf().

binwidth — Passed to ggplot2::geom_histogram(), ggplot2::geom_area(), and ggdist::stat_dots() to override the default binwidth.

bins — Passed to ggplot2::geom_histogram() and ggplot2::geom_area() to override the default binning.

breaks — Passed to ggplot2::geom_histogram() as an alternative to binwidth.

freq — For histograms and frequency polygons, 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 yaxis_text() convenience function.)

notch — For the box plot, a logical scalar passed to ggplot2::geom_boxplot(). Note: unlike geom_boxplot(), the default is notch=TRUE.

quantiles — For dot plots, an optional integer passed to ggdist::stat_dots() specifying the number of quantiles to use for a quantile dot plot. If quantiles is NA then all data points are plotted. The default is quantiles=100 so that each dot represent one percent of posterior mass.

probs — A numeric vector of probabilities controlling where quantile lines are drawn. Set to NULL to remove the lines.

y_draw — For ppc_violin_grouped(), a string specifying how to draw y: "violin" (default), "points" (jittered points), or "both".

y_jitter, y_size, y_alpha — For ppc_violin_grouped(), if y_draw is "points" or "both" then y_size, y_alpha, and y_jitter are passed to to the size, alpha, and width arguments of ggplot2::geom_jitter() to control the appearance of y points. The default of y_jitter=NULL will let **ggplot2** determine the amount of jitter.

pit — For ppc_pit_ecdf() and ppc_pit_ecdf_grouped(), an optional vector of precomputed PIT values (length length(y), values in [0, 1]). If NULL (default), PIT values are computed internally.

K — An optional integer defining the number of equally spaced evaluation points for the PIT-ECDF. Reducing K when using interpolate_adj = FALSE makes computing the confidence bands faster. For ppc_pit_ecdf() and ppc_pit_ecdf_grouped() when method = 'independent'. If pit is supplied, defaults to length(pit), otherwise yrep determines the maximum accuracy of the estimated PIT values and K is set to min(nrow(yrep) + 1, 1000). For mcmc_rank_ecdf(), defaults to the number of iterations per chain in x.

prob — The desired simultaneous coverage level of the bands around the ECDF. A value in (0,1). For ppc_pit_ecdf() and ppc_pit_ecdf_grouped().

plot_diff — A boolean defining whether to plot the difference between the observed PIT-ECDF and the theoretical expectation for uniform PIT values rather than plotting the regular ECDF. For ppc_pit_ecdf() and ppc_pit_ecdf_grouped() when method = 'independent'. The default is FALSE, but for large samples we recommend setting plot_diff = TRUE to better use the plot area.

interpolate_adj — A boolean defining if the simultaneous confidence bands should be interpolated based on precomputed values rather than computed exactly. Computing the bands may be computationally intensive and the approximation gives

a fast method for assessing the ECDF trajectory. For `ppc_pit_ecdf()` and `ppc_pit_ecdf_grouped()` when `method = 'independent'` and for `mcmc_rank_ecdf()`. The default is to use interpolation if `K` is greater than 200.

`method` The method used to calculate the uniformity test:

- `"independent"`: assumes independent PIT values (Säilynoja et al., 2022).
- `"correlated"`: accounts for correlated PIT values (Tesso & Vehtari, 2026).

`test` When `method = "correlated"`, which dependence-aware test to use: `"POT"`, `"PRIT"`, or `"PIET"`. Defaults to `"POT"`.

`gamma` When `method = "correlated"`, tolerance threshold controlling how strongly suspicious points are flagged. Larger values (`gamma > 0`) emphasize points with larger deviations. If `NULL`, defaults to `0` and thus all suspicious points are flagged.

`linewidth` When `method = "correlated"`, the line width of the ECDF. Defaults to `0.3`.

`color` When `method = "correlated"`, a named character vector of plot colors with elements `ecdf` and `highlight`. `ecdf` is used for the main ECDF line; `highlight` for suspicious regions flagged by the uniformity test. Defaults to `c(ecdf = "grey60", highlight = "red")`. Values must be valid ggplot2 colors (e.g. R color names or hex codes). Ignored when `method = "independent"`.

`help_text` When `method = "correlated"`, a boolean defining whether to add information about p-value to the plot. Defaults to `TRUE`.

`pareto_pit` A boolean defining whether to compute PIT values using Pareto-PIT method. Defaults to `TRUE` if `test` is either `"POT"` or `"PIET"` and no `pit` values are provided otherwise `FALSE`. This argument should normally not be modified by the user, except for development purposes. If `pit` is non-`NULL`, `pareto_pit` cannot be simultaneously `TRUE`.

`help_text_shrinkage`
When `method = "correlated"`, a numeric value between 0 and 1 defining the factor by which the help-text (p-value information) is scaled. The default is `0.8`.

Details

For Binomial data, the plots may be more useful if the input contains the "success" *proportions* (not discrete "success" or "failure" counts).

Value

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

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_dots()` A dot plot 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_ecdf_overlay()`, `ppc_dens_overlay()`, `ppc_ecdf_overlay_grouped()`, `ppc_dens_overlay_grouped()` 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). For an example of `ppc_dens_overlay()` also see Gabry et al. (2019).

`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.

`ppc_pit_ecdf()`, `ppc_pit_ecdf_grouped()` The PIT-ECDF of empirical PIT values for `y` relative to corresponding draws in `yrep` (or precomputed values supplied via `pit`). With `method = "independent"`, the plot shows `100 * prob`% central simultaneous confidence intervals under an independence assumption. With `method = "correlated"`, the plot uses a dependence-aware uniformity assessment and can highlight suspicious regions. See Säilynoja et al. (2022), Säilynoja et al. (2026), and Tesso & Vehtari (2026) for details. Note that the default "independent" method is **superseded** by the "correlated" method (Tesso & Vehtari, 2026) which accounts for dependent LOO-PIT values.

`ppc_data()` This function prepares data for plotting with **ggplot2** and doesn't itself make any plots. Users can call it directly to obtain the underlying data frame that (in most cases) is passed to **ggplot2**. This is useful when you want to customize the appearance of PPC plots beyond what the built-in plotting functions allow, or when you want to construct new types of PPC visualizations based on the same underlying data.

References

Gabry, J. , Simpson, D. , Vehtari, A. , Betancourt, M. and Gelman, A. (2019), Visualization in Bayesian workflow. *J. R. Stat. Soc. A*, 182: 389-402. doi:10.1111/rssa.12378. (journal version, arXiv preprint, code on GitHub)

Säilynoja, T., Bürkner, P.-C., and Vehtari, A. (2022). Graphical test for discrete uniformity and its applications in goodness-of-fit evaluation and multiple sample comparison. *Statistics and Computing*. 32(2), 32. doi:10.1007/s11222-022-10090-6. arXiv preprint: https://arxiv.org/abs/2103.10522

Säilynoja, T., Johnson, A., Martin, O., and Vehtari, A. (2026). Recommendations for visual predictive checks in Bayesian workflow. *Journal of Visualization and Interaction*. 1(1). doi:10.54337/jovi.v1i1.11478.

Tesso, H., and Vehtari, A. (2026). LOO-PIT predictive model checking. *arXiv preprint arXiv:2603.02928*. doi:10.48550/arXiv.2603.02928.

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-calibration`, `PPC-censoring`, `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()
group <- example_group_data()
dim(yrep)

ppc_dens_overlay(y, yrep[1:25, ])

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

# PIT-ECDF and PIT-ECDF difference plot of the PIT values of y compared to
# yrep with 99% simultaneous confidence bands.
ppc_pit_ecdf(y, yrep, prob = 0.99, plot_diff = FALSE)
ppc_pit_ecdf(y, yrep, prob = 0.99, plot_diff = TRUE)


# for ppc_hist,dens,freqpoly,boxplot,dots 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, ])

# dot plot
ppc_dots(y, yrep[1:8, ])



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

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

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


# density and distribution overlays by group
ppc_dens_overlay_grouped(y, yrep[1:25, ], group = group)

ppc_ecdf_overlay_grouped(y, yrep[1:25, ], group = group)


# PIT-ECDF plots of the PIT values by group
# with 99% simultaneous confidence bands.
ppc_pit_ecdf_grouped(y, yrep, group=group, prob=0.99)
```

```
# 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 - yrep`. See the **Details** and **Plot Descriptions** sections, below.

Usage

```
ppc_error_hist(
  y,
  yrep,
  ...,
  facet_args = list(),
  binwidth = NULL,
  bins = NULL,
  breaks = NULL,
  freq = TRUE
)

ppc_error_hist_grouped(
  y,
  yrep,
  group,
  ...,
  facet_args = list(),
  binwidth = NULL,
  bins = NULL,
  breaks = NULL,
  freq = TRUE
)
```

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

ppc_error_scatter_avg(
  y,
  yrep,
  x = NULL,
  ...,
  stat = "mean",
  size = 2.5,
  alpha = 0.8
)

ppc_error_scatter_avg_grouped(
  y,
  yrep,
  group,
  ...,
  stat = "mean",
  facet_args = list(),
  size = 2.5,
  alpha = 0.8
)

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

ppc_error_binned(
  y,
  yrep,
  x = NULL,
  ...,
  facet_args = list(),
  bins = NULL,
  size = 1,
  alpha = 0.25
)

ppc_error_data(y, yrep, group = NULL)
```

Arguments

y	A vector of observations. See **Details**.
yrep	An S by N matrix of draws from the posterior (or prior) predictive distribution, or a posterior::draws object. The number of rows, S, is the size of the posterior (or prior) sample used to generate yrep. The number of columns, N is the number of predicted observations (length(y)). The columns of yrep should be in the same order as the data points in y for the plots to make sense. See the **Details** and **Plot Descriptions** sections for additional advice specific to particular plots.
...	Currently unused.
facet_args	A named list of arguments (other than facets) passed to ggplot2::facet_wrap() or ggplot2::facet_grid() to control faceting. Note: if scales is not included in facet_args then **bayesplot** may use scales="free" as the default (depending on the plot) instead of the **ggplot2** default of scales="fixed".
binwidth	Passed to ggplot2::geom_histogram(), ggplot2::geom_area(), and ggdist::stat_dots() to override the default binwidth.
bins	For ppc_error_binned(), the number of bins to use (approximately).
breaks	Passed to ggplot2::geom_histogram() as an alternative to binwidth.
freq	For histograms and frequency polygons, 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 yaxis_text() convenience function.)
group	A grouping variable of the same length as y. Will be coerced to factor if not already a factor. Each value in group is interpreted as the group level pertaining to the corresponding observation.
size, alpha	For scatterplots, arguments passed to ggplot2::geom_point() 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.
x	A numeric vector the same length as y to use as the x-axis variable.
stat	A function or a string naming a function for computing the posterior average. In both cases, the function should take a vector input and return a scalar statistic. The function name is displayed in the axis-label. Defaults to "mean".

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

The plotting functions return a ggplot object that can be further customized using the **ggplot2** package. The functions with suffix _data() return the data that would have been drawn by the plotting function.

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.

When the optional `x` argument is provided, the average error is plotted on the y-axis and the predictor variable `x` is plotted on the x-axis.

`ppc_error_scatter_avg_vs_x()` Deprecated. Use `ppc_error_scatter_avg(x = x)` instead.

`ppc_error_binned()` Intended for use with binomial data. A separate binned error plot (similar to `arm::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.

`ppc_error_data()` Data-preparation back end for the `ppc_error_*()` family of plotting functions. Users can call `ppc_error_data()` directly to obtain the data frame of predictive errors (`y - yrep`) and create custom error visualizations with **ggplot2**.

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-calibration`, `PPC-censoring`, `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(y, yrep, x)

## Not run:
# binned error plot with binomial model from rstanarm
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, ])

# plotting against a covariate on x-axis
herd <- as.numeric(example_model$data$herd)
ppc_error_binned(y_prop, yrep_prop[1:6, ], x = herd)

## 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 = 0.5,
  prob_outer = 0.9,
```

```
  alpha = 0.33,
  size = 1,
  fatten = 2.5,
  linewidth = 1
)

ppc_intervals_grouped(
  y,
  yrep,
  x = NULL,
  group,
  ...,
  facet_args = list(),
  prob = 0.5,
  prob_outer = 0.9,
  alpha = 0.33,
  size = 1,
  fatten = 2.5,
  linewidth = 1
)

ppc_ribbon(
  y,
  yrep,
  x = NULL,
  ...,
  prob = 0.5,
  prob_outer = 0.9,
  alpha = 0.33,
  size = 0.25,
  y_draw = c("line", "points", "both")
)

ppc_ribbon_grouped(
  y,
  yrep,
  x = NULL,
  group,
  ...,
  facet_args = list(),
  prob = 0.5,
  prob_outer = 0.9,
  alpha = 0.33,
  size = 0.25,
  y_draw = c("line", "points", "both")
)

ppc_intervals_data(
```

```
  y,
  yrep,
  x = NULL,
  group = NULL,
  ...,
  prob = 0.5,
  prob_outer = 0.9
)

ppc_ribbon_data(
  y,
  yrep,
  x = NULL,
  group = NULL,
  ...,
  prob = 0.5,
  prob_outer = 0.9
)
```

Arguments

y	A vector of observations. See **Details**.
yrep	An S by N matrix of draws from the posterior (or prior) predictive distribution, or a posterior::draws object. The number of rows, S, is the size of the posterior (or prior) sample used to generate yrep. The number of columns, N is the number of predicted observations (length(y)). The columns of yrep should be in the same order as the data points in y for the plots to make sense. See the **Details** and **Plot Descriptions** sections for additional advice specific to particular plots.
x	A numeric vector to use as the x-axis variable. For example, x could be a predictor variable from a regression model, a time variable for time-series models, etc. If x is missing or NULL then the observation index is used for the x-axis.
...	Currently unused.
prob, prob_outer	Values between 0 and 1 indicating the desired probability mass to include in the inner and outer intervals. The defaults are prob=0.5 and prob_outer=0.9.
alpha, size, fatten, linewidth	Arguments passed to geoms. For ribbon plots alpha is passed to ggplot2::geom_ribbon() to control the opacity of the outer ribbon and size is passed to ggplot2::geom_line() to control the size of the line representing the median prediction (size=0 will remove the line). For interval plots alpha, size, fatten, and linewidth are passed to ggplot2::geom_pointrange() (fatten=0 will remove the point estimates).
group	A grouping variable of the same length as y. Will be coerced to factor if not already a factor. Each value in group is interpreted as the group level pertaining to the corresponding observation.
facet_args	A named list of arguments (other than facets) passed to ggplot2::facet_wrap() or ggplot2::facet_grid() to control faceting. Note: if scales is not included

in `facet_args` then **bayesplot** may use `scales="free"` as the default (depending on the plot) instead of the **ggplot2** default of `scales="fixed"`.

`y_draw` For ribbon plots only, a string specifying how to draw `y`. Can be `"line"` (the default), `"points"`, or `"both"`.

Value

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

Plot Descriptions

`ppc_intervals()`, `ppc_ribbon()` `100*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 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.

`ppc_intervals_data()`, `ppc_ribbon_data()` Data-preparation back end for `ppc_intervals()`, `ppc_ribbon()`, and their grouped variants. `ppc_ribbon_data()` is an alias for `ppc_intervals_data()`. Users can call either function directly to obtain the prepared data frame and create custom interval or ribbon visualizations with **ggplot2**.

References

Gabry, J. , Simpson, D. , Vehtari, A. , Betancourt, M. and Gelman, A. (2019), Visualization in Bayesian workflow. *J. R. Stat. Soc. A*, 182: 389-402. doi:10.1111/rssa.12378. (journal version, arXiv preprint, code on GitHub)

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-calibration`, `PPC-censoring`, `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_intervals(y, yrep)
ppc_ribbon(y, yrep)
ppc_ribbon(y, yrep, y_draw = "points")
```

```
## Not run:
ppc_ribbon(y, yrep, y_draw = "both")

## End(Not run)

ppc_intervals(y, yrep, size = 1.5, fatten = 0) # remove the yrep point estimates

color_scheme_set("teal")
year <- 1950:1999
ppc_intervals(y, yrep, x = year, fatten = 1) + ggplot2::xlab("Year")
ppc_ribbon(y, yrep, x = year) + 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, y_draw = "both") +
  ggplot2::scale_x_continuous(breaks = pretty)

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

# get the data frames used to make the ggplots
ppc_dat <- ppc_intervals_data(y, yrep, x = year, prob = 0.5)
ppc_group_dat <- ppc_intervals_data(y, yrep, x = year, group = group, prob = 0.5)

## Not run:
library("rstanarm")
fit <- stan_glmer(mpg ~ wt + (1|cyl), data = mtcars, refresh = 0)
yrep <- posterior_predict(fit)

color_scheme_set("purple")
ppc_intervals(y = mtcars$mpg, yrep = yrep, x = mtcars$wt, prob = 0.8) +
 panel_bg(fill="gray90", color = NA) +
 grid_lines(color = "white")

ppc_ribbon(y = mtcars$mpg, yrep = yrep, x = mtcars$wt,
           prob = 0.6, prob_outer = 0.8)

ppc_ribbon_grouped(y = mtcars$mpg, yrep = yrep, x = mtcars$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) +
 xaxis_title(FALSE)
```

```
## End(Not run)
```

PPC-loo *LOO predictive checks*

Description

Leave-One-Out (LOO) predictive checks. See the **Plot Descriptions** section, below, and Gabry et al. (2019) for details.

Usage

```
ppc_loo_pit_overlay(
  y,
  yrep,
  lw = NULL,
  ...,
  psis_object = NULL,
  pit = NULL,
  samples = 100,
  size = 0.25,
  alpha = 0.7,
  boundary_correction = TRUE,
  grid_len = 512,
  bw = NULL,
  trim = FALSE,
  adjust = NULL,
  kernel = NULL,
  n_dens = NULL
)

ppc_loo_pit_data(
  y,
  yrep,
  lw = NULL,
  ...,
  psis_object = NULL,
  pit = NULL,
  samples = 100,
  bw = "nrd0",
  boundary_correction = TRUE,
  grid_len = 512
)
```

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

ppc_loo_pit_ecdf(
  y,
  yrep,
  lw = NULL,
  ...,
  psis_object = NULL,
  pit = NULL,
  K = NULL,
  prob = 0.99,
  plot_diff = FALSE,
  interpolate_adj = NULL,
  method = NULL,
  test = NULL,
  gamma = NULL,
  linewidth = NULL,
  color = NULL,
  help_text = NULL,
  pareto_pit = NULL,
  help_text_shrinkage = NULL
)

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

ppc_loo_intervals(
  y,
  yrep,
```

```
  psis_object,
  ...,
  subset = NULL,
  intervals = NULL,
  prob = 0.5,
  prob_outer = 0.9,
  alpha = 0.33,
  size = 1,
  fatten = 2.5,
  linewidth = 1,
  order = c("index", "median")
)

ppc_loo_ribbon(
  y,
  yrep,
  psis_object,
  ...,
  subset = NULL,
  intervals = NULL,
  prob = 0.5,
  prob_outer = 0.9,
  alpha = 0.33,
  size = 0.25
)
```

Arguments

y	A vector of observations. See **Details**.
yrep	An S by N matrix of draws from the posterior (or prior) predictive distribution, or a posterior::draws object. The number of rows, S, is the size of the posterior (or prior) sample used to generate yrep. The number of columns, N is the number of predicted observations (length(y)). The columns of yrep should be in the same order as the data points in y for the plots to make sense. See the **Details** and **Plot Descriptions** sections for additional advice specific to particular plots.
lw	A matrix of (smoothed) log weights with the same dimensions as yrep. See loo::psis() and the associated weights() method as well as the **Examples** section, below. If lw is not specified then psis_object can be provided and log weights will be extracted.
...	Currently unused.
psis_object	If using **loo** version 2.0.0 or greater, an object returned by the psis() function (or by the loo() function with argument save_psis set to TRUE).
pit	For ppc_loo_pit_overlay(), ppc_loo_pit_qq(), and ppc_loo_pit_ecdf(), an optional vector of precomputed LOO-PIT values (length length(y), values in [0, 1]). If supplied, y, yrep, and lw / psis_object are ignored. If NULL (default), LOO-PIT values are computed internally.

samples
: For ppc_loo_pit_overlay(), the number of data sets (each the same size as y) to simulate from the standard uniform distribution. The default is 100. The density estimate of each dataset is plotted as a thin line in the plot, with the density estimate of the LOO PITs overlaid as a thicker dark line.

alpha, size, fatten, linewidth
: Arguments passed to code geoms to control plot aesthetics. For ppc_loo_pit_qq() and ppc_loo_pit_overlay(),size and alpha are passed to ggplot2::geom_point() and ggplot2::geom_density(), respectively. For ppc_loo_intervals(), size linewidth and fatten are passed to ggplot2::geom_pointrange(). For ppc_loo_ribbon(), alpha and size are passed to ggplot2::geom_ribbon(). For ppc_loo_pit_ecdf(), linewidth for the ECDF plot. When method = "correlated", defaults to 0.3. When method = "independent", if NULL no linewidth is specified for the ECDF line.

boundary_correction
: For ppc_loo_pit_overlay(), when set to TRUE (the default) the function will compute boundary corrected density values via convolution and a Gaussian filter, also known as the reflection method (Boneva et al., 1971). As a result, parameters controlling the standard kernel density estimation such as adjust, kernel and n_dens are ignored. NOTE: The current implementation only works well for continuous observations.

grid_len
: For ppc_loo_pit_overlay(), when boundary_correction is set to TRUE this parameter specifies the number of points used to generate the estimations. This is set to 512 by default.

bw, adjust, kernel, n_dens
: Optional arguments passed to stats::density() to override the defaults.

trim
: Passed to ggplot2::stat_density().

compare
: For ppc_loo_pit_qq(), 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 compare="normal", the Q-Q plot compares standard normal quantiles calculated from the PIT values to the theoretical standard normal quantiles.

K
: An optional integer defining the number of equally spaced evaluation points for the PIT-ECDF. Reducing K when using interpolate_adj = FALSE makes computing the confidence bands faster. For ppc_loo_pit_ecdf(), when method = 'independent'. If pit is supplied, defaults to length(pit), otherwise yrep determines the maximum accuracy of the estimated PIT values and K is set to min(nrow(yrep) + 1, 1000).

prob, prob_outer
: Values between 0 and 1 indicating the desired probability mass to include in the inner and outer intervals. The defaults are prob=0.5 and prob_outer=0.9 for ppc_loo_intervals() and prob = 0.99 for ppc_loo_pit_ecdf().

plot_diff
: A boolean defining whether to plot the difference between the observed PIT-ECDF and the theoretical expectation for uniform PIT values rather than plotting the regular ECDF. For ppc_loo_pit_ecdf(), when method = 'independent'. The default is FALSE, but for large samples we recommend setting plot_diff = TRUE to better use the plot area.

interpolate_adj
: A boolean defining if the simultaneous confidence bands should be interpolated based on precomputed values rather than computed exactly. Computing the bands may be computationally intensive and the approximation gives a fast method for assessing the ECDF trajectory. For `ppc_loo_pit_ecdf()` when `method = 'independent'`. The default is to use interpolation if `K` is greater than 200.

method
: The method used to calculate the uniformity test:
  - `"independent"`: assumes independent PIT values (Säilynoja et al., 2022).
  - `"correlated"`: accounts for correlated PIT values (Tesso & Vehtari, 2026).

test
: When `method = "correlated"`, which dependence-aware test to use: `"POT"`, `"PRIT"`, or `"PIET"`. Defaults to `"POT"`.

gamma
: When `method = "correlated"`, tolerance threshold controlling how strongly suspicious points are flagged. Larger values (`gamma > 0`) emphasize points with larger deviations. If `NULL`, defaults to `0` and thus all suspicious points are flagged.

color
: When `method = "correlated"`, a named character vector of plot colors with elements `ecdf` and `highlight`. `ecdf` is used for the main ECDF line; `highlight` for suspicious regions flagged by the uniformity test. Defaults to `c(ecdf = "grey60", highlight = "red")`. Values must be valid ggplot2 colors (e.g. R color names or hex codes). Ignored when `method = "independent"`.

help_text
: When `method = "correlated"`, a boolean defining whether to add information about p-value to the plot. Defaults to `TRUE`.

pareto_pit
: A boolean defining whether to compute PIT values using Pareto-PIT method. Defaults to `TRUE` if `test` is either `"POT"` or `"PIET"` and no `pit` values are provided otherwise `FALSE`. This argument should normally not be modified by the user, except for development purposes. If `pit` is non-`NULL`, `pareto_pit` cannot be simultaneously `TRUE`.

help_text_shrinkage
: When `method = "correlated"`, a numeric value between 0 and 1 defining the factor by which the help-text (p-value information) is scaled. The default is `0.8`.

subset
: For `ppc_loo_intervals()` and `ppc_loo_ribbon()`, an optional integer vector indicating which observations in `y` (and `yrep`) to include. Dropping observations from `y` and `yrep` manually before passing them to the plotting function will not work because the dimensions will not match up with the dimensions of `psis_object`, but if all of `y` and `yrep` are passed along with `subset` then **bayesplot** can do the subsetting internally for `y`, `yrep` *and* `psis_object`. See the **Examples** section for a demonstration.

intervals
: For `ppc_loo_intervals()` and `ppc_loo_ribbon()`, optionally a matrix of precomputed LOO predictive intervals that can be specified instead of `yrep` (ignored if `intervals` is specified). If not specified the intervals are computed internally before plotting. If specified, `intervals` must be a matrix with number of rows equal to the number of data points and five columns in the following order: lower outer interval, lower inner interval, median (50%), upper inner interval and upper outer interval (column names are ignored).

order
: For `ppc_loo_intervals()`, 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

The plotting functions return a ggplot object that can be further customized using the **ggplot2** package. The functions with suffix _data() return the data that would have been drawn by the plotting function.

Plot Descriptions

ppc_loo_pit_data() This function prepares LOO-PIT data for plotting with **ggplot2**. It is the data-preparation back end for the LOO-PIT plotting functions (ppc_loo_pit_overlay(), ppc_loo_pit_qq(), and ppc_loo_pit_ecdf()), and users can call it directly to create custom LOO-PIT plots using ggplot2. The function computes the leave-one-out probability integral transform (LOO-PIT) values and returns a data frame that can be used to build ggplot objects. This is useful when you want to create custom visualizations of LOO-PIT values beyond what the built-in plotting functions provide.

ppc_loo_pit_overlay(), ppc_loo_pit_qq(), ppc_loo_pit_ecdf() 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) and section 5 of Gabry et al. (2019) for an example of using **bayesplot** for these checks.

The LOO PIT values are asymptotically uniform (for continuous data) if the model is calibrated. The ppc_loo_pit_overlay() function creates a plot comparing the density of the LOO PITs (thick line) to the density estimates of many simulated data sets from the standard uniform distribution (thin lines). See Gabry et al. (2019) for an example of interpreting the shape of the miscalibration that can be observed in these plots.

The ppc_loo_pit_qq() function provides an alternative visualization of the miscalibration with a quantile-quantile (Q-Q) plot comparing the LOO PITs to the standard uniform distribution. Comparing to the uniform is not good for extreme probabilities close to 0 and 1, so it can sometimes be useful to set the compare argument to "normal", which will produce a Q-Q plot comparing standard normal quantiles calculated from the PIT values to the theoretical standard normal quantiles. This can help see the (mis)calibration better for the extreme values. However, in most cases we have found that the overlaid density plot (ppc_loo_pit_overlay()) function will provide a clearer picture of calibration problems than the Q-Q plot.

The ppc_loo_pit_ecdf() function visualizes the empirical cumulative distribution function (ECDF) of the LOO PIT values. With method = "independent", the plot overlays 100 * prob% simultaneous confidence intervals for a standard uniform sample. With method = "correlated", the plot uses a dependence-aware uniformity assessment and can highlight suspicious regions. Setting plot_diff = TRUE displays the ECDF minus the theoretical expectation, which can improve visual assessment of calibration. Note that the default "independent" method is **superseded** by the "correlated" method (Tesso & Vehtari, 2026) which accounts for dependent LOO-PIT 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)

Gabry, J. , Simpson, D. , Vehtari, A. , Betancourt, M. and Gelman, A. (2019), Visualization in Bayesian workflow. *J. R. Stat. Soc. A*, 182: 389-402. doi:10.1111/rssa.12378. (journal version, arXiv preprint, code on GitHub)

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: https://arxiv.org/abs/1507.04544

Säilynoja, T., Bürkner, P.-C., and Vehtari, A. (2022). Graphical test for discrete uniformity and its applications in goodness-of-fit evaluation and multiple sample comparison. *Statistics and Computing*. 32(2), 32. doi:10.1007/s11222-022-10090-6. arXiv preprint: https://arxiv.org/abs/2103.10522

Tesso, H., and Vehtari, A. (2026). LOO-PIT predictive model checking. *arXiv preprint arXiv:2603.02928*. doi:10.48550/arXiv.2603.02928.

Boneva, L. I., Kendall, D., & Stefanov, I. (1971). Spline transformations: Three new diagnostic aids for the statistical data-analyst. *J. R. Stat. Soc. B* (Methodological), 33(1), 1-71. https://www.jstor.org/stable/2986005.

See Also

Other PPCs: PPC-calibration, PPC-censoring, 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,
  iter = 100,
  chains = 2,
  cores = 2
)
y <- radon$log_radon
yrep <- posterior_predict(fit)

loo1 <- loo(fit, save_psis = TRUE, cores = 4)
psis1 <- loo1$psis_object
lw <- weights(psis1) # normalized log weights

# marginal predictive check using LOO probability integral transform
color_scheme_set("orange")
ppc_loo_pit_overlay(y, yrep, lw = lw)
```

```
ppc_loo_pit_qq(y, yrep, lw = lw)
ppc_loo_pit_qq(y, yrep, lw = lw, compare = "normal")

# predictive calibration check using LOO probability integral transform
ppc_loo_pit_ecdf(y, yrep, lw)

# With `plot_diff = TRUE` it is easier to assess the calibration.
ppc_loo_pit_ecdf(y, yrep, lw, plot_diff = TRUE)

# can use the psis object instead of lw
ppc_loo_pit_qq(y, yrep, psis_object = psis1)

# loo predictive intervals vs observations
keep_obs <- 1:50
ppc_loo_intervals(y, yrep, psis_object = psis1, subset = keep_obs)

color_scheme_set("gray")
ppc_loo_intervals(y, yrep,
  psis_object = psis1, subset = keep_obs,
  order = "median"
)

## End(Not run)
```

PPC-overview *Graphical posterior predictive checking*

Description

The **bayesplot** PPC module provides various plotting functions for creating graphical displays comparing observed data to simulated data from the posterior (or prior) predictive distribution. See the sections below for a brief discussion of the ideas behind posterior predictive checking, an overview of the available PPC plots, and tips on providing an interface to **bayesplot** from another package.

For plots of posterior (or prior) predictive distributions that do *not* include observed data see PPD-overview instead.

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 θ in the model. For each draw of the parameters θ from the posterior distribution $p(\theta \mid y, X)$ we generate an entire vector of outcomes. The result is an $S \times N$ matrix of simulations, where S is the 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).

Prior predictive checking: To use **bayesplot** for *prior* predictive checks you can simply use draws from the prior predictive distribution instead of the posterior predictive distribution. See Gabry et al. (2019) for more on prior predictive checking and when it is reasonable to compare the prior predictive distribution to the observed data. If you want to avoid using the observed data for prior predictive checks then you can use the **bayesplot** PPD plots instead, which do not take a `y` argument, or you can use the PPC plots but provide plausible or implausible `y` values that you want to compare to the prior predictive realizations.

PPC plotting functions

The plotting functions for prior and posterior predictive checking all have the prefix `ppc_` and all require the arguments `y`, a vector of observations, and `yrep`, a matrix of replications (in-sample predictions). The plots are organized into several categories, each with its own documentation:

- PPC-distributions: Histograms, kernel density estimates, boxplots, and other plots comparing the empirical distribution of data `y` to the distributions of individual simulated datasets (rows) in `yrep`.
- PPC-test-statistics: The distribution of a statistic, or a pair of statistics, over the simulated datasets (rows) in `yrep` compared to value of the statistic(s) computed from `y`.
- PPC-intervals: Interval estimates of `yrep` with `y` overlaid. The x-axis variable can be optionally specified by the user (e.g. to plot against a predictor variable or over time).
- PPC-errors: Plots of predictive errors (`y - yrep`) computed from `y` and each of the simulated datasets (rows) in `yrep`. For binomial models binned error plots are also available.
- PPC-scatterplots: Scatterplots (and similar visualizations) of the data `y` vs. individual simulated datasets (rows) in `yrep`, or vs. the average value of the distributions of each data point (columns) in `yrep`.
- PPC-discrete: 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.
- PPC-loo: PPC functions for predictive checks based on (approximate) leave-one-out (LOO) cross-validation. '
- PPC-censoring: PPC functions comparing the empirical distribution of censored data `y` to the distributions of individual simulated datasets (rows) in `yrep`.

- PPC-calibration: PPC functions to assess the calibration of predictions or predictive probabilities in relation to binary observations.

Providing an interface for 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. and Gelman, A. (2019), Visualization in Bayesian workflow. *J. R. Stat. Soc. A*, 182: 389-402. doi:10.1111/rssa.12378. (journal version, arXiv preprint, code on GitHub)

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-calibration`, `PPC-censoring`, `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 `yrep` from the posterior predictive distribution. See the **Plot Descriptions** and **Details** sections, below.

Usage

```
ppc_scatter(
  y,
  yrep,
  ...,
  facet_args = list(),
  size = 2.5,
  alpha = 0.8,
  ref_line = TRUE
)

ppc_scatter_avg(
  y,
  yrep,
```

```
  ...,
  stat = "mean",
  size = 2.5,
  alpha = 0.8,
  ref_line = TRUE
)

ppc_scatter_avg_grouped(
  y,
  yrep,
  group,
  ...,
  stat = "mean",
  facet_args = list(),
  size = 2.5,
  alpha = 0.8,
  ref_line = TRUE
)

ppc_scatter_data(y, yrep)

ppc_scatter_avg_data(y, yrep, group = NULL, stat = "mean")
```

Arguments

y	A vector of observations. See **Details**.
yrep	An S by N matrix of draws from the posterior (or prior) predictive distribution, or a posterior::draws object. The number of rows, S, is the size of the posterior (or prior) sample used to generate yrep. The number of columns, N is the number of predicted observations (length(y)). The columns of yrep should be in the same order as the data points in y for the plots to make sense. See the **Details** and **Plot Descriptions** sections for additional advice specific to particular plots.
...	Currently unused.
facet_args	A named list of arguments (other than facets) passed to ggplot2::facet_wrap() or ggplot2::facet_grid() to control faceting. Note: if scales is not included in facet_args then **bayesplot** may use scales="free" as the default (depending on the plot) instead of the **ggplot2** default of scales="fixed".
size, alpha	Arguments passed to ggplot2::geom_point() to control the appearance of the points.
ref_line	If TRUE (the default) a dashed line with intercept 0 and slope 1 is drawn behind the scatter plot.
stat	A function or a string naming a function for computing the posterior average. In both cases, the function should take a vector input and return a scalar statistic. The function name is displayed in the axis-label, and the underlying $rep_label for ppc_scatter_avg_data() includes the function name. Defaults to "mean".
group	A grouping variable of the same length as y. Will be coerced to factor if not already a factor. Each value in group is interpreted as the group level pertaining

to the corresponding observation.

Details

For Binomial data, the plots may be more useful if the input contains the "success" *proportions* (not discrete "success" or "failure" counts).

Value

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

Plot Descriptions

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

`ppc_scatter_avg()` A single scatterplot of `y` against the average values of `yrep`, i.e., the points `(x,y) = (average(yrep[, n]), y[n])`, where each `yrep[, n]` is a vector of length equal to the number of posterior draws and `average()` is a summary statistic. Unlike for `ppc_scatter()`, for `ppc_scatter_avg()` `yrep` should contain many draws (rows).

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

`ppc_scatter_data()`, `ppc_scatter_avg_data()` Data-preparation back ends for the `ppc_scatter*()` family of plotting functions. `ppc_scatter_data()` returns a data frame with one row per observation per `yrep` draw, while `ppc_scatter_avg_data()` returns a data frame with one row per observation summarising `yrep` draws with the chosen `stat`. Users can call these functions directly to create custom visualizations with **ggplot2**.

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-calibration`, `PPC-censoring`, `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

# don't draw line x=y
ppc_scatter_avg(y, yrep, ref_line = FALSE)

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

# "average" function is customizable
ppc_scatter_avg(y, yrep, stat = "median", ref_line = FALSE)

# for ppc_scatter_avg_grouped the default is to allow the facets
# to have different x and y axes
group <- example_group_data()
ppc_scatter_avg_grouped(y, yrep, group)

# let x-axis vary but force y-axis to be the same
ppc_scatter_avg_grouped(y, yrep, group, facet_args = list(scales = "free_x"))
```

PPC-test-statistics *PPC test statistics*

Description

The distribution of a (test) statistic `T(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, as well as Gabry et al. (2019).

NOTE: Although the default test statistic is the mean, this is unlikely to detect anything interesting in most cases. In general we recommend using some other test statistic as discussed in Section 5 of Gabry et al. (2019).

Usage

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

ppc_stat_grouped(
  y,
  yrep,
```

```
  group,
  stat = "mean",
  ...,
  discrete = FALSE,
  facet_args = list(),
  binwidth = NULL,
  bins = NULL,
  breaks = NULL,
  freq = TRUE
)

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

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

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

ppc_stat_data(y, yrep, group = NULL, stat)
```

Arguments

y	A vector of observations. See **Details**.
yrep	An S by N matrix of draws from the posterior (or prior) predictive distribution, or a posterior::draws object. The number of rows, S, is the size of the posterior (or prior) sample used to generate yrep. The number of columns, N is the number of predicted observations (length(y)). The columns of yrep should be in the same order as the data points in y for the plots to make sense. See the **Details** and **Plot Descriptions** sections for additional advice specific to particular plots.
stat	A single function or a string naming a function, except for the 2D plot which

	requires a vector of exactly two names or functions. In all cases the function(s) should take a vector input and return a scalar statistic. If specified as a string (or strings) then the legend will display the function name(s). If specified as a function (or functions) then generic naming is used in the legend.
`...`	Currently unused.
`discrete`	For `ppc_stat()` and `ppc_stat_grouped()`, if `TRUE` then a bar chart is used instead of a histogram.
`binwidth`	Passed to `ggplot2::geom_histogram()`, `ggplot2::geom_area()`, and `ggdist::stat_dots()` to override the default binwidth.
`bins`	Passed to `ggplot2::geom_histogram()` and `ggplot2::geom_area()` to override the default binning.
`breaks`	Passed to `ggplot2::geom_histogram()` as an alternative to `binwidth`.
`freq`	For histograms and frequency polygons, `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 `yaxis_text()` convenience function.)
`group`	A grouping variable of the same length as `y`. Will be coerced to factor if not already a factor. Each value in `group` is interpreted as the group level pertaining to the corresponding observation.
`facet_args`	A named list of arguments (other than `facets`) passed to `ggplot2::facet_wrap()` or `ggplot2::facet_grid()` to control faceting. Note: if `scales` is not included in `facet_args` then **bayesplot** may use `scales="free"` as the default (depending on the plot) instead of the **ggplot2** default of `scales="fixed"`.
`size, alpha`	For the 2D plot only, arguments passed to `ggplot2::geom_point()` to control the appearance of scatterplot points.

Details

For Binomial data, the plots may be more useful if the input contains the "success" *proportions* (not discrete "success" or "failure" counts).

Value

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

Plot Descriptions

`ppc_stat()`, `ppc_stat_freqpoly()` A histogram/bar plot or frequency polygon of the distribution of a 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. More details and example usage of `ppc_stat()` can be found in Gabry et al. (2019).

`ppc_stat_grouped()`,`ppc_stat_freqpoly_grouped()` The same as `ppc_stat()` and `ppc_stat_freqpoly()`, but a separate plot is generated for each level of a grouping variable. More details and example usage of `ppc_stat_grouped()` can be found in Gabry et al. (2019).

`ppc_stat_2d()` A scatterplot showing the joint distribution of two statistics computed over the datasets (rows) in `yrep`. The value of the statistics in the observed data is overlaid as large point.

`ppc_stat_data()` Data-preparation back end for `ppc_stat()`, `ppc_stat_freqpoly()`, and their grouped variants. Users can call `ppc_stat_data()` directly to obtain the data frame of test-statistic values computed from `y` and each row of `yrep`, enabling custom test-statistic visualizations with **ggplot2**.

References

Gabry, J. , Simpson, D. , Vehtari, A. , Betancourt, M. and Gelman, A. (2019), Visualization in Bayesian workflow. *J. R. Stat. Soc. A*, 182: 389-402. doi:10.1111/rssa.12378. (journal version, arXiv preprint, code on GitHub)

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-calibration`, `PPC-censoring`, `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, stat = "median")
ppc_stat(y, yrep, stat = "sd") + legend_none()

# discrete data example
set.seed(0)
y_discrete <- rbinom(20, 1, 0.2)
yrep_discrete <- matrix(rbinom(2000, 1, prob = 0.4), 1000, 20, byrow = TRUE)
ppc_stat(y_discrete, yrep_discrete, stat = "mean", discrete = TRUE)

# use your own function for the 'stat' argument
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 instead of
# using its name but then the legend doesn't include the function name
ppc_stat(y, yrep, stat = function(y) quantile(y, 0.25))

# plots by group
color_scheme_set("teal")
group <- example_group_data()
ppc_stat_grouped(y, yrep, group, stat = "median")
ppc_stat_grouped(y, yrep, group, stat = "mad") + yaxis_text()

# discrete data example with groups
group_discrete <- rep(c("First Half","Second Half"), each = 10)
```

```
ppc_stat_grouped(y_discrete, yrep_discrete, group_discrete, stat = "mean", discrete = TRUE)

# force y-axes to have same scales, allow x axis to vary
ppc_stat_grouped(y, yrep, group, facet_args = list(scales = "free_x")) + yaxis_text()

# the freqpoly plots use frequency polygons instead of histograms
ppc_stat_freqpoly(y, yrep, stat = "median")
ppc_stat_freqpoly_grouped(y, yrep, group, stat = "median", facet_args = list(nrow = 2))

# ppc_stat_2d allows 2 statistics and makes a scatterplot
bayesplot_theme_set(ggplot2::theme_linedraw())
color_scheme_set("viridisE")
ppc_stat_2d(y, yrep, stat = c("mean", "sd"))

bayesplot_theme_set(ggplot2::theme_grey())
color_scheme_set("brewer-Paired")
ppc_stat_2d(y, yrep, stat = c("median", "mad"))

# reset aesthetics
color_scheme_set()
bayesplot_theme_set()
```

PPD-distributions *PPD distributions*

Description

Plot posterior or prior predictive distributions. Each of these functions makes the same plot as the corresponding ppc_ function but without plotting any observed data y. The **Plot Descriptions** section at PPC-distributions has details on the individual plots.

Usage

```
ppd_data(ypred, group = NULL)

ppd_dens_overlay(
  ypred,
  show_marginal = FALSE,
  ...,
  size = 0.25,
  alpha = 0.7,
  trim = FALSE,
  bw = NULL,
  adjust = NULL,
  kernel = NULL,
  bounds = NULL,
  n_dens = NULL
)
```

```
ppd_ecdf_overlay(
  ypred,
  show_marginal = FALSE,
  ...,
  discrete = deprecated(),
  pad = TRUE,
  size = 0.25,
  alpha = 0.7
)

ppd_dens(
  ypred,
  show_marginal = FALSE,
  ...,
  trim = FALSE,
  size = 0.5,
  alpha = 1,
  bounds = NULL
)

ppd_hist(
  ypred,
  show_marginal = FALSE,
  ...,
  binwidth = NULL,
  bins = NULL,
  breaks = NULL,
  freq = !show_marginal
)

ppd_dots(
  ypred,
  show_marginal = FALSE,
  ...,
  binwidth = NA,
  quantiles = 100,
  freq = TRUE
)

ppd_freqpoly(
  ypred,
  show_marginal = FALSE,
  ...,
  binwidth = NULL,
  bins = NULL,
  freq = !show_marginal,
  size = 0.5,
```

```
  alpha = 1
)

ppd_freqpoly_grouped(
  ypred,
  group,
  show_marginal = FALSE,
  ...,
  binwidth = NULL,
  bins = NULL,
  freq = !show_marginal,
  size = 0.5,
  alpha = 1
)

ppd_boxplot(
  ypred,
  show_marginal = FALSE,
  ...,
  notch = TRUE,
  size = 0.5,
  alpha = 1
)
```

Arguments

ypred	An S by N matrix of draws from the posterior (or prior) predictive distribution, or a posterior::draws object. The number of rows, S, is the size of the posterior (or prior) sample used to generate ypred. The number of columns, N, is the number of predicted observations.
group	A grouping variable of the same length as y. Will be coerced to factor if not already a factor. Each value in group is interpreted as the group level pertaining to the corresponding observation.
show_marginal	Plot the marginal PPD along with the ypreds.
...	For dot plots, optional additional arguments to pass to ggdist::stat_dots().
size, alpha	Passed to the appropriate geom to control the appearance of the predictive distributions.
trim	A logical scalar passed to ggplot2::geom_density().
bw, adjust, kernel, n_dens, bounds	Optional arguments passed to stats::density() (and bounds to ggplot2::stat_density()) to override default kernel density estimation parameters or truncate the density support. If NULL (default), bw is set to "nrd0", adjust to 1, kernel to "gaussian", and n_dens to 1024.
discrete	**[Deprecated]** The discrete argument is deprecated. The ECDF is a step function by definition, so geom_step() is now always used.
pad	A logical scalar passed to ggplot2::stat_ecdf().

binwidth	Passed to ggplot2::geom_histogram(), ggplot2::geom_area(), and ggdist::stat_dots() to override the default binwidth.
bins	Passed to ggplot2::geom_histogram() and ggplot2::geom_area() to override the default binning.
breaks	Passed to ggplot2::geom_histogram() as an alternative to binwidth.
freq	For histograms and frequency polygons, 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 yaxis_text() convenience function.)
quantiles	For dot plots, an optional integer passed to ggdist::stat_dots() specifying the number of quantiles to use for a quantile dot plot. If quantiles is NA then all data points are plotted. The default is quantiles=100 so that each dot represent one percent of posterior mass.
notch	For the box plot, a logical scalar passed to ggplot2::geom_boxplot(). Note: unlike geom_boxplot(), the default is notch=TRUE.

Details

For Binomial data, the plots may be more useful if the input contains the "success" *proportions* (not discrete "success" or "failure" counts).

Value

The plotting functions return a ggplot object that can be further customized using the **ggplot2** package. The functions with suffix _data() return the data that would have been drawn by the plotting function.

References

Säilynoja, T., Johnson, A., Martin, O., and Vehtari, A. (2026). Recommendations for visual predictive checks in Bayesian workflow. *Journal of Visualization and Interaction*. 1(1). doi:10.54337/jovi.v1i1.11478.

See Also

Other PPDs: PPD-intervals, PPD-overview, PPD-test-statistics

Examples

```
# difference between ppd_dens_overlay() and ppc_dens_overlay()
color_scheme_set("brightblue")
preds <- example_yrep_draws()
ppd_dens_overlay(ypred = preds[1:50, ])
ppd_dens_overlay(ypred = preds[1:50, ], show_marginal = TRUE)
ppc_dens_overlay(y = example_y_data(), yrep = preds[1:50, ])
```

PPD-intervals *PPD intervals*

Description

Medians and central interval estimates of posterior or prior predictive distributions. Each of these functions makes the same plot as the corresponding ppc_ function but without plotting any observed data y. The **Plot Descriptions** section at PPC-intervals has details on the individual plots.

Usage

```
ppd_intervals(
  ypred,
  x = NULL,
  ...,
  prob = 0.5,
  prob_outer = 0.9,
  alpha = 0.33,
  size = 1,
  fatten = 2.5,
  linewidth = 1
)

ppd_intervals_grouped(
  ypred,
  x = NULL,
  group,
  ...,
  facet_args = list(),
  prob = 0.5,
  prob_outer = 0.9,
  alpha = 0.33,
  size = 1,
  fatten = 2.5,
  linewidth = 1
)

ppd_ribbon(
  ypred,
  x = NULL,
  ...,
  prob = 0.5,
  prob_outer = 0.9,
  alpha = 0.33,
  size = 0.25
)
```

```
ppd_ribbon_grouped(
  ypred,
  x = NULL,
  group,
  ...,
  facet_args = list(),
  prob = 0.5,
  prob_outer = 0.9,
  alpha = 0.33,
  size = 0.25
)

ppd_intervals_data(
  ypred,
  x = NULL,
  group = NULL,
  ...,
  prob = 0.5,
  prob_outer = 0.9
)

ppd_ribbon_data(
  ypred,
  x = NULL,
  group = NULL,
  ...,
  prob = 0.5,
  prob_outer = 0.9
)
```

Arguments

`ypred`	An `S` by `N` matrix of draws from the posterior (or prior) predictive distribution, or a `posterior::draws` object. The number of rows, `S`, is the size of the posterior (or prior) sample used to generate `ypred`. The number of columns, `N`, is the number of predicted observations.
`x`	A numeric vector to use as the x-axis variable. For example, `x` could be a predictor variable from a regression model, a time variable for time-series models, etc. If `x` is missing or `NULL` then the observation index is used for the x-axis.
`...`	Currently unused.
`prob`, `prob_outer`	Values between `0` and `1` indicating the desired probability mass to include in the inner and outer intervals. The defaults are `prob=0.5` and `prob_outer=0.9`.
`alpha`, `size`, `fatten`, `linewidth`	Arguments passed to geoms. For ribbon plots `alpha` is passed to `ggplot2::geom_ribbon()` to control the opacity of the outer ribbon and `size` is passed to `ggplot2::geom_line()` to control the size of the line representing the median prediction (`size=0` will remove the line). For interval plots `alpha`, `size`, `fatten`, and `linewidth` are

passed to ggplot2::geom_pointrange() (fatten=0 will remove the point estimates).

group — A grouping variable of the same length as y. Will be coerced to factor if not already a factor. Each value in group is interpreted as the group level pertaining to the corresponding observation.

facet_args — A named list of arguments (other than facets) passed to ggplot2::facet_wrap() or ggplot2::facet_grid() to control faceting. Note: if scales is not included in facet_args then **bayesplot** may use scales="free" as the default (depending on the plot) instead of the **ggplot2** default of scales="fixed".

Value

The plotting functions return a ggplot object that can be further customized using the **ggplot2** package. The functions with suffix _data() return the data that would have been drawn by the plotting function.

References

Gabry, J. , Simpson, D. , Vehtari, A. , Betancourt, M. and Gelman, A. (2019), Visualization in Bayesian workflow. *J. R. Stat. Soc. A*, 182: 389-402. doi:10.1111/rssa.12378. (journal version, arXiv preprint, code on GitHub)

See Also

Other PPDs: PPD-distributions, PPD-overview, PPD-test-statistics

Examples

```
color_scheme_set("brightblue")
ypred <- example_yrep_draws()
x <- example_x_data()
group <- example_group_data()

ppd_intervals(ypred[, 1:50])
ppd_intervals(ypred[, 1:50], fatten = 0)
ppd_intervals(ypred[, 1:50], fatten = 0, linewidth = 2)
ppd_intervals(ypred[, 1:50], prob_outer = 0.75, fatten = 0, linewidth = 2)

# put a predictor variable on the x-axis
ppd_intervals(ypred[, 1:100], x = x[1:100], fatten = 1) +
  ggplot2::labs(y = "Prediction", x = "Some variable of interest")

# with a grouping variable too
ppd_intervals_grouped(
  ypred = ypred[, 1:100],
  x = x[1:100],
  group = group[1:100],
  size = 2,
  fatten = 0,
  facet_args = list(nrow = 2)
)
```

```
# even reducing size, ppd_intervals is too cluttered when there are many
# observations included (ppd_ribbon is better)
ppd_intervals(ypred, size = 0.5, fatten = 0.1, linewidth = 0.5)
ppd_ribbon(ypred)
ppd_ribbon(ypred, size = 0) # remove line showing median prediction
```

PPD-overview *Plots of posterior or prior predictive distributions*

Description

The **bayesplot** PPD module provides various plotting functions for creating graphical displays of simulated data from the posterior or prior predictive distribution. These plots are essentially the same as the corresponding PPC plots but without showing any observed data. Because these are not "checks" compared to data we use PPD (for prior/posterior predictive distribution) instead of PPC (for prior/posterior predictive check).

PPD plotting functions

The functions for plotting prior and posterior predictive distributions without observed data each have the prefix `ppd_` and all have a required argument `ypred` (a matrix of predictions). The plots are organized into several categories, each with its own documentation:

- PPD-distributions: Histograms, kernel density estimates, boxplots, and other plots of multiple simulated datasets (rows) in `ypred`. These are the same as the plots in PPC-distributions but without including any comparison to `y`.
- PPD-intervals: Interval estimates for each predicted observations (columns) in `ypred`. The x-axis variable can be optionally specified by the user (e.g. to plot against against a predictor variable or over time).These are the same as the plots in PPC-intervals but without including any comparison to `y`.
- PPD-test-statistics: The distribution of a statistic, or a pair of statistics, over the simulated datasets (rows) in `ypred`. These are the same as the plots in PPC-test-statistics but without including any comparison to `y`.

References

Gabry, J. , Simpson, D. , Vehtari, A. , Betancourt, M. and Gelman, A. (2019), Visualization in Bayesian workflow. *J. R. Stat. Soc. A*, 182: 389-402. doi:10.1111/rssa.12378. (journal version, arXiv preprint, code on GitHub)

See Also

Other PPDs: `PPD-distributions`, `PPD-intervals`, `PPD-test-statistics`

PPD-test-statistics *PPD test statistics*

Description

The distribution of a (test) statistic `T(ypred)`, or a pair of (test) statistics, over the simulations from the posterior or prior predictive distribution. Each of these functions makes the same plot as the corresponding ppc_ function but without comparing to any observed data `y`. The **Plot Descriptions** section at PPC-test-statistics has details on the individual plots.

Usage

```
ppd_stat(
  ypred,
  stat = "mean",
  show_marginal = FALSE,
  ...,
  discrete = FALSE,
  binwidth = NULL,
  bins = NULL,
  breaks = NULL,
  freq = !show_marginal
)

ppd_stat_grouped(
  ypred,
  group,
  stat = "mean",
  show_marginal = FALSE,
  ...,
  discrete = FALSE,
  facet_args = list(),
  binwidth = NULL,
  bins = NULL,
  breaks = NULL,
  freq = !show_marginal
)

ppd_stat_freqpoly(
  ypred,
  stat = "mean",
  show_marginal = FALSE,
  ...,
  facet_args = list(),
  binwidth = NULL,
  bins = NULL,
  freq = TRUE
```

```
)

ppd_stat_freqpoly_grouped(
  ypred,
  group,
  stat = "mean",
  show_marginal = FALSE,
  ...,
  facet_args = list(),
  binwidth = NULL,
  bins = NULL,
  freq = TRUE
)

ppd_stat_2d(
  ypred,
  stat = c("mean", "sd"),
  show_marginal = FALSE,
  ...,
  size = 2.5,
  alpha = 0.7
)

ppd_stat_data(ypred, group = NULL, stat, show_marginal = FALSE)
```

Arguments

ypred	An S by N matrix of draws from the posterior (or prior) predictive distribution, or a posterior::draws object. The number of rows, S, is the size of the posterior (or prior) sample used to generate ypred. The number of columns, N, is the number of predicted observations.
stat	A single function or a string naming a function, except for the 2D plot which requires a vector of exactly two names or functions. In all cases the function(s) should take a vector input and return a scalar statistic. If specified as a string (or strings) then the legend will display the function name(s). If specified as a function (or functions) then generic naming is used in the legend.
show_marginal	Plot the marginal PPD along with the ypreds.
...	Currently unused.
discrete	For ppc_stat() and ppc_stat_grouped(), if TRUE then a bar chart is used instead of a histogram.
binwidth	Passed to ggplot2::geom_histogram(), ggplot2::geom_area(), and ggdist::stat_dots() to override the default binwidth.
bins	Passed to ggplot2::geom_histogram() and ggplot2::geom_area() to override the default binning.
breaks	Passed to ggplot2::geom_histogram() as an alternative to binwidth.
freq	For histograms and frequency polygons, 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 `yaxis_text()` convenience function.)
`group`	A grouping variable of the same length as `y`. Will be coerced to factor if not already a factor. Each value in `group` is interpreted as the group level pertaining to the corresponding observation.
`facet_args`	A named list of arguments (other than `facets`) passed to `ggplot2::facet_wrap()` or `ggplot2::facet_grid()` to control faceting. Note: if `scales` is not included in `facet_args` then **bayesplot** may use `scales="free"` as the default (depending on the plot) instead of the **ggplot2** default of `scales="fixed"`.
`size, alpha`	For the 2D plot only, arguments passed to `ggplot2::geom_point()` to control the appearance of scatterplot points.

Details

For Binomial data, the plots may be more useful if the input contains the "success" *proportions* (not discrete "success" or "failure" counts).

Value

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

References

Gabry, J. , Simpson, D. , Vehtari, A. , Betancourt, M. and Gelman, A. (2019), Visualization in Bayesian workflow. *J. R. Stat. Soc. A*, 182: 389-402. doi:10.1111/rssa.12378. (journal version, arXiv preprint, code on GitHub)

See Also

Other PPDs: `PPD-distributions`, `PPD-intervals`, `PPD-overview`

Examples

```
yrep <- example_yrep_draws()
ppd_stat(yrep)
ppd_stat(yrep, stat = "sd") + legend_none()
ppd_stat(yrep, show_marginal = TRUE)

# use your own function for the 'stat' argument
color_scheme_set("brightblue")
q25 <- function(y) quantile(y, 0.25)
ppd_stat(yrep, stat = "q25") # legend includes function name
ppd_stat(yrep, stat = "q25", show_marginal = TRUE)
```

theme_default *Default* **bayesplot** *plotting theme*

Description

The theme_default() function returns the default ggplot theme used by the **bayesplot** plotting functions. See bayesplot_theme_set() for details on setting and updating the plotting theme.

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 ggplot2::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.

Value

A ggplot theme object.

See Also

bayesplot_theme_set() to change the ggplot theme.

bayesplot-colors to set or view the color scheme used for plotting.

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

Examples

```
class(theme_default())

bayesplot_theme_set() # defaults to setting theme_default()
x <- example_mcmc_draws()
mcmc_hist(x)

# change the default font size and family for bayesplots
bayesplot_theme_set(theme_default(base_size = 8, base_family = "sans"))
mcmc_hist(x)
mcmc_areas(x, regex_pars = "beta")

# change back
bayesplot_theme_set()
```

```
mcmc_areas(x, regex_pars = "beta")
```

`tidy-params` *Tidy parameter selection*

Description

Parameter selection in the style of **dplyr** and other tidyverse packages.

Usage

```
param_range(prefix, range, vars = NULL)

param_glue(pattern, ..., vars = NULL)
```

Arguments

`prefix`, `range`	For `param_range()` only, `prefix` is a string naming a parameter and `range` is an integer vector providing the indices of a subset of elements to select. For example, using `param_range("beta", c(1,2,8))` would select parameters named `beta[1]`, `beta[2]`, and `beta[8]`. `param_range()` is only designed for the case that the indices are integers surrounded by brackets. If there are no brackets use num_range().
`vars`	`NULL` or a character vector of parameter names to choose from. This is only needed for the atypical use case of calling the function as a standalone function outside of `vars()`, `select()`, etc. Typically this is left as `NULL` and will be set automatically for the user.

`pattern`, `...`	For `param_glue()` only, `pattern` is a string containing expressions enclosed in braces and `...` should be named arguments providing one character vector per expression in braces in `pattern`. It is easiest to describe how to use these arguments with an example:
	
	`param_glue("beta_{var}[{level}]",`
	`  var = c("age", "income"),`
	`  level = c(3,8))`
	
	would select parameters with names `"beta_age[3]"`, `"beta_income[3]"`, `"beta_age[8]"`, `"beta_income[8]"`.

Details

As of version 1.7.0, **bayesplot** allows the pars argument for MCMC plots to use "tidy" variable selection (in the style of the **dplyr** package). The vars() function is re-exported from **dplyr** for this purpose.

Features of tidy selection includes direct selection (vars(alpha, sigma)), everything-but selection (vars(-alpha)), ranged selection (vars(`beta[1]`:`beta[3]`)), support for selection functions (vars(starts_with("beta"))), and combinations of these features. See the **Examples** section, below.

When using pars for tidy parameter selection, the regex_pars argument is ignored because **bayesplot** supports using tidyselect helper functions (starts_with(), contains(), num_range(), etc.) for the same purpose. **bayesplot** also exports some additional helper functions to help with parameter selection:

- param_range(): like num_range() but used when parameter indexes are in brackets (e.g. beta[2]).
- param_glue(): for more complicated parameter names with multiple indexes (including variable names) inside the brackets (e.g., beta[(Intercept) age_group:3]).

These functions can be used inside of vars(), dplyr::select(), and similar functions, just like the tidyselect helper functions.

Extra Advice

Parameter names in vars() are not quoted. When the names contain special characters like brackets, they should be wrapped in backticks, as in vars(`beta[1]`).

To exclude a range of variables, wrap the sequence in parentheses and then negate it. For example, (vars(-(`beta[1]`:`beta[3]`))) would exclude beta[1], beta[2], and beta[3].

vars() is a helper function. It holds onto the names and expressions used to select columns. When selecting variables inside a **bayesplot** function, use vars(...): mcmc_hist(data, pars = vars(alpha)). When using select() to prepare a dataframe for a **bayesplot** function, do not use vars(): data %>% select(alpha) %>% mcmc_hist().

Internally, tidy selection works by converting names and expressions into position numbers. As a result, integers will select parameters; vars(1, 3) selects the first and third ones. We do not endorse this approach because positions might change as variables are added and removed from models. To select a parameter that happens to be called 1, use backticks to escape it vars(`1`).

See Also

glue::glue()

Examples

```
x <- example_mcmc_draws(params = 6)
dimnames(x)
mcmc_hex(x, pars = vars(alpha, `beta[2]`))
mcmc_dens(x, pars = vars(sigma, contains("beta")))
mcmc_hist(x, pars = vars(-contains("beta")))
```

```
# using the param_range() helper
mcmc_hist(x, pars = vars(param_range("beta", c(1, 3, 4))))


############################
## Examples using rstanarm ##
############################
if (requireNamespace("rstanarm", quietly = TRUE)) {
  # see ?rstanarm::example_model
  fit <- example("example_model", package = "rstanarm", local=TRUE)$value
  print(fit)
  posterior <- as.data.frame(fit)
  str(posterior)

  color_scheme_set("brightblue")
  mcmc_hist(posterior, pars = vars(size, contains("period")))

  # same as previous but using dplyr::select() and piping
  library("dplyr")
  posterior %>%
    select(size, contains("period")) %>%
    mcmc_hist()

  mcmc_intervals(posterior, pars = vars(contains("herd")))
  mcmc_intervals(posterior, pars = vars(contains("herd"), -contains("Sigma")))

  bayesplot_theme_set(ggplot2::theme_dark())
  color_scheme_set("viridisC")
  mcmc_areas_ridges(posterior, pars = vars(starts_with("b[")))

  bayesplot_theme_set()
  color_scheme_set("purple")
  not_789 <- vars(starts_with("b["), -matches("[7-9]"))
  mcmc_intervals(posterior, pars = not_789)

  # using the param_glue() helper
  just_149 <- vars(param_glue("b[(Intercept) herd:{level}]", level = c(1,4,9)))
  mcmc_intervals(posterior, pars = just_149)

  # same but using param_glue() with dplyr::select()
  # before passing to bayesplot
  posterior %>%
    select(param_glue("b[(Intercept) herd:{level}]",
                      level = c(1, 4, 9))) %>%
    mcmc_intervals()
}

## Not run:
###################################
## More examples of param_glue() ##
###################################
library(dplyr)
posterior <- tibble(
```

```
  b_Intercept = rnorm(1000),
  sd_condition__Intercept = rexp(1000),
  sigma = rexp(1000),
  `r_condition[A,Intercept]` = rnorm(1000),
  `r_condition[B,Intercept]` = rnorm(1000),
  `r_condition[C,Intercept]` = rnorm(1000),
  `r_condition[A,Slope]` = rnorm(1000),
  `r_condition[B,Slope]` = rnorm(1000)
)
posterior

# using one expression in braces
posterior %>%
  select(
    param_glue("r_condition[{level},Intercept]", level = c("A", "B"))
  ) %>%
  mcmc_hist()

# using multiple expressions in braces
posterior %>%
   select(
     param_glue(
       "r_condition[{level},{type}]",
        level = c("A", "B"),
        type = c("Intercept", "Slope"))
   ) %>%
   mcmc_hist()

## End(Not run)
```

Index

∗ **MCMC**
MCMC-combos, 21
MCMC-diagnostics, 22
MCMC-distributions, 26
MCMC-intervals, 32
MCMC-nuts, 39
MCMC-overview, 42
MCMC-parcoord, 44
MCMC-recover, 47
MCMC-scatterplots, 50
MCMC-traces, 57
∗ **PPCs**
PPC-calibration, 65
PPC-censoring, 70
PPC-discrete, 73
PPC-distributions, 78
PPC-errors, 86
PPC-intervals, 90
PPC-loo, 95
PPC-overview, 102
PPC-scatterplots, 104
PPC-test-statistics, 107
∗ **PPDs**
PPD-distributions, 111
PPD-intervals, 115
PPD-overview, 118
PPD-test-statistics, 119

abline_01 *(*bayesplot-helpers*)*, 11
available_mcmc *(*available_ppc*)*, 5
available_ppc, 5
available_ppd *(*available_ppc*)*, 5

base::grep(), *6*
bayesplot *(*bayesplot-package*)*, 3
bayesplot-colors, *4*, 6, *20*, *122*
bayesplot-extractors, 9
bayesplot-helpers, 11, *20*, *122*
bayesplot-package, 3
bayesplot_grid, 17
bayesplot_grid(), *53, 54*
bayesplot_theme_get, 19
bayesplot_theme_replace *(*bayesplot_theme_get*)*, 19
bayesplot_theme_set *(*bayesplot_theme_get*)*, 19
bayesplot_theme_set(), *4*, *8*, *122*
bayesplot_theme_update *(*bayesplot_theme_get*)*, 19

color_scheme_get *(*bayesplot-colors*)*, 6
color_scheme_set *(*bayesplot-colors*)*, 6
color_scheme_view *(*bayesplot-colors*)*, 6
colors, *8*
Combos, *43*
Comparisons to true values, *43*

dplyr::select(...), *24*, *29*, *35*, *44*, *52*, *59*

facet_bg *(*bayesplot-helpers*)*, 11
facet_text *(*bayesplot-helpers*)*, 11
factor, *68*, *72*, *75*, *81*, *88*, *92*, *105*, *109*, *113*, *117*, *121*

General MCMC diagnostics, *43*
ggdist::stat_dots(), *23*, *30*, *40*, *49*, *81*, *82*, *88*, *109*, *113*, *114*, *120*
ggplot2::aes(), *14*
ggplot2::element_blank(), *13*
ggplot2::element_line(), *13*
ggplot2::element_rect(), *13*
ggplot2::element_text(), *13*
ggplot2::facet_grid(), *24*, *30*, *48*, *60*, *88*, *92*, *105*, *109*, *117*, *121*
ggplot2::facet_wrap(), *24*, *30*, *48*, *60*, *75*, *88*, *92*, *105*, *109*, *117*, *121*
ggplot2::geom_abline(), *13*
ggplot2::geom_area(), *23*, *30*, *40*, *49*, *82*, *88*, *109*, *114*, *120*
ggplot2::geom_bar(), *74*

ggplot2::geom_boxplot(), *82*, *114*
ggplot2::geom_density(), *30*, *81*, *98*, *113*
ggplot2::geom_hex(), *52*
ggplot2::geom_histogram(), *23*, *30*, *40*, *49*, *82*, *88*, *109*, *114*, *120*
ggplot2::geom_hline(), *13*
ggplot2::geom_jitter(), *82*
ggplot2::geom_line(), *23*, *45*, *75*, *92*, *116*
ggplot2::geom_point(), *23*, *49*, *52, 53*, *88*, *98*, *105*, *109*, *121*
ggplot2::geom_pointrange(), *75*, *92*, *98*, *117*
ggplot2::geom_ribbon(), *92*, *98*, *116*
ggplot2::geom_rug(), *61*
ggplot2::geom_vline(), *13*
ggplot2::ggsave(), *4*
ggplot2::labs(), *14*
ggplot2::stat_density(), *30*, *36*, *81*, *98*, *113*
ggplot2::stat_ecdf(), *82*, *113*
ggplot2::stat_function(), *13, 14*
ggplot2::theme(), *14*, *21*
ggplot2::theme_bw(), *122*
ggplot2::theme_grey(), *19*
ggplot2::theme_set(), *19*
glue::glue(), *124*
grid_lines *(*bayesplot-helpers*)*, 11
gridExtra::arrangeGrob(), *17, 18*, *21*, *40*

hline_0 *(*bayesplot-helpers*)*, 11
hline_at *(*bayesplot-helpers*)*, 11

integer, *75*
invisibly, *7*

lbub *(*bayesplot-helpers*)*, 11
legend_move *(*bayesplot-helpers*)*, 11
legend_none *(*bayesplot-helpers*)*, 11
legend_none(), *21*
legend_text *(*bayesplot-helpers*)*, 11
log_posterior *(*bayesplot-extractors*)*, 9
log_posterior(), *40*, *53*
loo::psis(), *97*

match.fun(), *29*, *35*, *45*, *52*, *60*
MCMC, *3*, *6*, *21*
MCMC *(*MCMC-overview*)*, 42
MCMC plots, *124*
MCMC-combos, 21
MCMC-diagnostics, *11*, 22
MCMC-distributions, 26
MCMC-intervals, 32
MCMC-nuts, *11*, *22*, *25*, 39
MCMC-overview, *21*, *23*, *29*, *35*, 42, *44*, *48*, *51*, *52*, *59*
MCMC-parcoord, 44
MCMC-recover, 47
MCMC-scatterplots, 50
MCMC-traces, 57
mcmc_acf *(*MCMC-diagnostics*)*, 22
mcmc_acf_bar *(*MCMC-diagnostics*)*, 22
mcmc_areas *(*MCMC-intervals*)*, 32
mcmc_areas_data *(*MCMC-intervals*)*, 32
mcmc_areas_ridges *(*MCMC-intervals*)*, 32
mcmc_areas_ridges_data *(*MCMC-intervals*)*, 32
mcmc_combo *(*MCMC-combos*)*, 21
mcmc_dens *(*MCMC-distributions*)*, 26
mcmc_dens_chains *(*MCMC-distributions*)*, 26
mcmc_dens_chains_data *(*MCMC-distributions*)*, 26
mcmc_dens_overlay *(*MCMC-distributions*)*, 26
mcmc_dots *(*MCMC-distributions*)*, 26
mcmc_dots_by_chain *(*MCMC-distributions*)*, 26
mcmc_hex *(*MCMC-scatterplots*)*, 50
mcmc_hist *(*MCMC-distributions*)*, 26
mcmc_hist_by_chain *(*MCMC-distributions*)*, 26
mcmc_intervals *(*MCMC-intervals*)*, 32
mcmc_intervals_data *(*MCMC-intervals*)*, 32
mcmc_neff *(*MCMC-diagnostics*)*, 22
mcmc_neff_data *(*MCMC-diagnostics*)*, 22
mcmc_neff_hist *(*MCMC-diagnostics*)*, 22
mcmc_nuts_acceptance *(*MCMC-nuts*)*, 39
mcmc_nuts_divergence *(*MCMC-nuts*)*, 39
mcmc_nuts_energy *(*MCMC-nuts*)*, 39
mcmc_nuts_stepsize *(*MCMC-nuts*)*, 39
mcmc_nuts_treedepth *(*MCMC-nuts*)*, 39
mcmc_pairs *(*MCMC-scatterplots*)*, 50
mcmc_pairs(), *42*
mcmc_parcoord *(*MCMC-parcoord*)*, 44
mcmc_parcoord(), *42*
mcmc_parcoord_data *(*MCMC-parcoord*)*, 44
mcmc_rank_ecdf *(*MCMC-traces*)*, 57

mcmc_rank_hist (MCMC-traces), 57
mcmc_rank_overlay (MCMC-traces), 57
mcmc_recover_hist (MCMC-recover), 47
mcmc_recover_intervals (MCMC-recover), 47
mcmc_recover_scatter (MCMC-recover), 47
mcmc_rhat (MCMC-diagnostics), 22
mcmc_rhat_data (MCMC-diagnostics), 22
mcmc_rhat_hist (MCMC-diagnostics), 22
mcmc_scatter (MCMC-scatterplots), 50
mcmc_scatter(), *42*
mcmc_trace (MCMC-traces), 57
mcmc_trace(), *41*
mcmc_trace_data (MCMC-traces), 57
mcmc_trace_highlight (MCMC-traces), 57
mcmc_violin (MCMC-distributions), 26

neff_ratio (bayesplot-extractors), 9
neff_ratio(), *23*
No-U-Turn Sampler (NUTS), *3*
num_range(), *123, 124*
NUTS, *45*, *54*, *60, 61*
NUTS (MCMC-nuts), 39
NUTS diagnostics, *43*
NUTS(), *54*
nuts_params (bayesplot-extractors), 9
nuts_params(), *40*, *45*, *52*, *60*

options(), *122*
overlay_function (bayesplot-helpers), 11

pairs_condition (MCMC-scatterplots), 50
pairs_style_np (MCMC-scatterplots), 50
panel_bg (bayesplot-helpers), 11
Parallel coordinates plots, *43*
param_glue (tidy-params), 123
param_range (tidy-params), 123
parcoord_style_np (MCMC-parcoord), 44
plot_bg (bayesplot-helpers), 11
Posterior distributions, *43*
posterior::draws, *68*, *71*, *74*, *81*, *88*, *92*, *97*, *105*, *108*, *113*, *116*, *120*
pp_check, 64
pp_check(), *104*
PPC, *3*, *6*, *64*, *73*, *118*
PPC (PPC-overview), 102
PPC-calibration, 65, *104*
PPC-censoring, 70, *103*
PPC-discrete, 73, *103*
PPC-distributions, 78, *103*, *111*, *118*
PPC-errors, 86, *103*
PPC-intervals, 90, *103*, *115*, *118*
PPC-loo, 95, *103*
PPC-overview, 102
PPC-scatterplots, *103*, 104
PPC-statistics (PPC-test-statistics), 107
PPC-test-statistics, *103*, 107, *118, 119*
ppc_, *111*, *115*, *119*
ppc_bars (PPC-discrete), 73
ppc_bars_data (PPC-discrete), 73
ppc_bars_grouped (PPC-discrete), 73
ppc_boxplot (PPC-distributions), 78
ppc_calibration (PPC-calibration), 65
ppc_calibration_data (PPC-calibration), 65
ppc_calibration_grouped (PPC-calibration), 65
ppc_calibration_overlay (PPC-calibration), 65
ppc_calibration_overlay_grouped (PPC-calibration), 65
ppc_data (PPC-distributions), 78
ppc_dens (PPC-distributions), 78
ppc_dens_overlay (PPC-distributions), 78
ppc_dens_overlay_grouped (PPC-distributions), 78
ppc_dots (PPC-distributions), 78
ppc_ecdf_overlay, *76*
ppc_ecdf_overlay (PPC-distributions), 78
ppc_ecdf_overlay_grouped (PPC-distributions), 78
ppc_error_binned (PPC-errors), 86
ppc_error_data (PPC-errors), 86
ppc_error_hist (PPC-errors), 86
ppc_error_hist_grouped (PPC-errors), 86
ppc_error_scatter (PPC-errors), 86
ppc_error_scatter_avg (PPC-errors), 86
ppc_error_scatter_avg_grouped (PPC-errors), 86
ppc_error_scatter_avg_vs_x (PPC-errors), 86
ppc_freqpoly (PPC-distributions), 78
ppc_freqpoly_grouped (PPC-distributions), 78
ppc_hist (PPC-distributions), 78
ppc_intervals (PPC-intervals), 90

ppc_intervals(), *100*
ppc_intervals_data *(*PPC-intervals*)*, 90
ppc_intervals_grouped *(*PPC-intervals*)*, 90
ppc_km_overlay *(*PPC-censoring*)*, 70
ppc_km_overlay_grouped *(*PPC-censoring*)*, 70
ppc_loo_calibration *(*PPC-calibration*)*, 65
ppc_loo_calibration_grouped *(*PPC-calibration*)*, 65
ppc_loo_intervals *(*PPC-loo*)*, 95
ppc_loo_pit *(*PPC-loo*)*, 95
ppc_loo_pit_data *(*PPC-loo*)*, 95
ppc_loo_pit_ecdf *(*PPC-loo*)*, 95
ppc_loo_pit_overlay *(*PPC-loo*)*, 95
ppc_loo_pit_qq *(*PPC-loo*)*, 95
ppc_loo_ribbon *(*PPC-loo*)*, 95
ppc_pit_ecdf *(*PPC-distributions*)*, 78
ppc_pit_ecdf(), *76*
ppc_pit_ecdf_grouped *(*PPC-distributions*)*, 78
ppc_pit_ecdf_grouped(), *76*
ppc_ribbon *(*PPC-intervals*)*, 90
ppc_ribbon(), *100*
ppc_ribbon_data *(*PPC-intervals*)*, 90
ppc_ribbon_grouped *(*PPC-intervals*)*, 90
ppc_rootogram *(*PPC-discrete*)*, 73
ppc_rootogram_grouped *(*PPC-discrete*)*, 73
ppc_scatter *(*PPC-scatterplots*)*, 104
ppc_scatter_avg *(*PPC-scatterplots*)*, 104
ppc_scatter_avg_data *(*PPC-scatterplots*)*, 104
ppc_scatter_avg_grouped *(*PPC-scatterplots*)*, 104
ppc_scatter_data *(*PPC-scatterplots*)*, 104
ppc_stat *(*PPC-test-statistics*)*, 107
ppc_stat(), *76*
ppc_stat_2d *(*PPC-test-statistics*)*, 107
ppc_stat_data *(*PPC-test-statistics*)*, 107
ppc_stat_freqpoly *(*PPC-test-statistics*)*, 107
ppc_stat_freqpoly_grouped *(*PPC-test-statistics*)*, 107
ppc_stat_grouped *(*PPC-test-statistics*)*, 107
ppc_stat_grouped(), *76*
ppc_violin_grouped *(*PPC-distributions*)*, 78
PPD, *3*, *6*, *103*
PPD *(*PPD-overview*)*, 118
PPD-distributions, 111, *118*
PPD-intervals, 115, *118*
PPD-overview, *102*, 118
PPD-statistics *(*PPD-test-statistics*)*, 119
PPD-test-statistics, *118*, 119
ppd_boxplot *(*PPD-distributions*)*, 111
ppd_data *(*PPD-distributions*)*, 111
ppd_dens *(*PPD-distributions*)*, 111
ppd_dens_overlay *(*PPD-distributions*)*, 111
ppd_dots *(*PPD-distributions*)*, 111
ppd_ecdf_overlay *(*PPD-distributions*)*, 111
ppd_freqpoly *(*PPD-distributions*)*, 111
ppd_freqpoly_grouped *(*PPD-distributions*)*, 111
ppd_hist *(*PPD-distributions*)*, 111
ppd_intervals *(*PPD-intervals*)*, 115
ppd_intervals_data *(*PPD-intervals*)*, 115
ppd_intervals_grouped *(*PPD-intervals*)*, 115
ppd_ribbon *(*PPD-intervals*)*, 115
ppd_ribbon_data *(*PPD-intervals*)*, 115
ppd_ribbon_grouped *(*PPD-intervals*)*, 115
ppd_stat *(*PPD-test-statistics*)*, 119
ppd_stat(), *76*
ppd_stat_2d *(*PPD-test-statistics*)*, 119
ppd_stat_data *(*PPD-test-statistics*)*, 119
ppd_stat_freqpoly *(*PPD-test-statistics*)*, 119
ppd_stat_freqpoly_grouped *(*PPD-test-statistics*)*, 119
ppd_stat_grouped *(*PPD-test-statistics*)*, 119
ppd_stat_grouped(), *76*

RColorBrewer::brewer.pal(), *8*
regular expression, *11*, *24*, *29*, *35*, *45*, *52*, *59*
reshape2::melt(), *11*
rhat *(*bayesplot-extractors*)*, 9
rhat(), *36*

scatter_style_np *(*MCMC-scatterplots*)*, 50
Scatterplots, *43*

select helpers, *11*, *24*, *29*, *35*, *45*, *52*, *60*
stats::density(), *30*, *36*, *81*, *98*, *113*

theme, *13*, *21*, *122*
theme_default, 122
theme_default(), *4*, *8*, *14*, *20*, *122*
Tidy parameter selection, *24*, *29*, *35*, *44*, *52*, *59*
tidy-params, 123
tidyselect helper functions, *124*
Trace plots, *43*
trace_style_np *(*MCMC-traces*)*, 57

Uncertainty intervals, *43*

vars(), *124*
vline_0 *(*bayesplot-helpers*)*, 11
vline_at *(*bayesplot-helpers*)*, 11

xaxis_text *(*bayesplot-helpers*)*, 11
xaxis_ticks *(*bayesplot-helpers*)*, 11
xaxis_title *(*bayesplot-helpers*)*, 11

yaxis_text *(*bayesplot-helpers*)*, 11
yaxis_text(), *30*, *82*, *88*, *109*, *114*, *121*
yaxis_ticks *(*bayesplot-helpers*)*, 11
yaxis_title *(*bayesplot-helpers*)*, 11