Package ‘rstanarm’

April 29, 2017

Type Package

Title Bayesian Applied Regression Modeling via Stan

Version 2.15.3

Date 2017-04-27

Description Estimates previously compiled regression models using the 'rstan' package, which provides the R interface to the Stan C++ library for Bayesian estimation. Users specify models via the customary R syntax with a formula and data.frame plus some additional arguments for priors.

License GPL (>= 3)

Depends R (>= 3.0.2), Rcpp (>= 0.12.0), methods

Imports bayesplot (>= 1.2.0), ggplot2 (>= 2.2.1), lme4 (>= 1.1-8), loo (>= 1.1.0), Matrix, nlme (>= 3.1-124), rstan (>= 2.14.2), rstantools (>= 1.2.0), shinystan (>= 2.3.0), stats, utils

Suggests arm, betareg, digest, gridExtra, HSAUR3, knitr (>= 1.15.1), MASS, mgcv, rmarkdown, roxygen2, testthat (>= 1.0.2)

LinkingTo StanHeaders (>= 2.14.0), rstan (>= 2.14.2), BH (>= 1.62.0), Rcpp (>= 0.12.0), RcppEigen

SystemRequirements pandoc

VignetteBuilder knitr

LazyData true

NeedsCompilation yes

URL `https://groups.google.com/forum/#!forum/stan-users`, `http://mc-stan.org/`

BugReports `https://github.com/stan-dev/rstanarm/issues`

RoxygenNote 5.0.1

Author Jonah Gabry [aut],
Imad Ali [ctb],
Trustees of Columbia University [cph],
R Core Deveopment Team [cph] (R/stan_aov.R),
Douglas Bates [cph] (R/pp_data.R),

Martin Maechler [cph] (R/pp_data.R),
Ben Bolker [cph] (R/pp_data.R),
Steve Walker [cph] (R/pp_data.R),
Brian Ripley [cph] (R/stan_aov.R, R/stan_polr.R),
William Venables [cph] (R/stan_polr.R),
Ben Goodrich [cre, aut]

Maintainer Ben Goodrich <benjamin.goodrich@columbia.edu>

Repository CRAN

Date/Publication 2017-04-29 12:58:17 UTC

R topics documented:

`rstanarm-package` *Applied Regression Modeling via RStan*

Description

The **rstanarm** package is an appendage to the **rstan** package that enables many of the most common applied regression models to be estimated using Markov Chain Monte Carlo, variational approximations to the posterior distribution, or optimization. The **rstanarm** package allows these models to be specified using the customary R modeling syntax (e.g., like that of `glm` with a `formula` and a `data.frame`).

The set of models supported by **rstanarm** is large (and will continue to grow), but also limited enough so that it is possible to integrate them tightly with the `pp_check` function for graphical posterior predictive checks and the `posterior_predict` function to easily estimate the effect of specific manipulations of predictor variables or to predict the outcome in a training set.

The objects returned by the **rstanarm** modeling functions are called `stanreg` objects. In addition to all of the typical `methods` defined for fitted model objects, stanreg objects can be passed to the `loo` function in the **loo** package for model comparison or to the `launch_shinystan` function in the **shinystan** package in order to visualize the posterior distribution using the ShinyStan graphical user interface. See the **rstanarm** vignettes for more details about the entire process.

Estimation algorithms

The modeling functions in the **rstanarm** package take an `algorithm` argument that can be one of the following:

Sampling (`algorithm="sampling"`) Uses Markov Chain Monte Carlo (MCMC) — in particular, Hamiltonian Monte Carlo (HMC) with a tuned but diagonal mass matrix — to draw from the posterior distribution of the parameters. See `sampling` for more details. This is the slowest but most reliable of the available estimation algorithms and it is **the default and recommended algorithm for statistical inference.**

Mean-field (`algorithm="meanfield"`) Uses mean-field variational inference to draw from an approximation to the posterior distribution. In particular, this algorithm finds the set of independent normal distributions in the unconstrained space that — when transformed into the constrained space — most closely approximate the posterior distribution. Then it draws repeatedly from these independent normal distributions and transforms them into the constrained space. The entire process is much faster than HMC and yields independent draws but **is not recommended for final statistical inference**. It can be useful to narrow the set of candidate models in large problems, particularly when specifying `QR=TRUE` in `stan_glm`, `stan_glmer`, and `stan_gamm4`, but is **only an approximation to the posterior distribution**.

Full-rank (`algorithm="fullrank"`) Uses full-rank variational inference to draw from an approximation to the posterior distribution by finding the multivariate normal distribution in the unconstrained space that — when transformed into the constrained space — most closely approximates the posterior distribution. Then it draws repeatedly from this multivariate normal distribution and transforms the draws into the constrained space. This process is slower than meanfield variational inference but is faster than HMC. Although still an approximation to the posterior distribution and thus **not recommended for final statistical inference**, the approximation is more realistic than that of mean-field variational inference because the parameters are not assumed to be independent in the unconstrained space. Nevertheless, fullrank variational inference is a more difficult optimization problem and the algorithm is more prone to non-convergence or convergence to a local optimum.

Optimizing (`algorithm="optimizing"`) Finds the posterior mode using a C++ implementation of the LBGFS algorithm. See `optimizing` for more details. If there is no prior information, then this is equivalent to maximum likelihood, in which case there is no great reason to use the functions in the **rstanarm** package over the emulated functions in other packages. However, if priors are specified, then the estimates are penalized maximum likelihood estimates, which may have some redeeming value. Currently, optimization is only supported for `stan_glm`.

Modeling functions

The model estimating functions are described in greater detail in their individual help pages and vignettes. Here we provide a very brief overview:

`stan_lm`, `stan_aov`, `stan_biglm` Similar to `lm` or `aov` but with novel regularizing priors on the model parameters that are driven by prior beliefs about R^2, the proportion of variance in the outcome attributable to the predictors in a linear model.

`stan_glm`, `stan_glm.nb` Similar to `glm` but with various possible prior distributions for the coefficients and, if applicable, a prior distribution for any auxiliary parameter in a Generalized Linear Model (GLM) that is characterized by a `family` object (e.g. the shape parameter in Gamma models). It is also possible to estimate a negative binomial model in a similar way to the `glm.nb` function in the **MASS** package.

`stan_glmer`, `stan_glmer.nb`, `stan_lmer` Similar to the `glmer`, `glmer.nb` and `lmer` functions in the **lme4** package in that GLMs are augmented to have group-specific terms that deviate from the common coefficients according to a mean-zero multivariate normal distribution with a highly-structured but unknown covariance matrix (for which **rstanarm** introduces an innovative prior distribution). MCMC provides more appropriate estimates of uncertainty for models that consist of a mix of common and group-specific parameters.

`stan_gamm4` Similar to `gamm4` in the **gamm4** package, which augments a GLM (possibly with group-specific terms) with nonlinear smooth functions of the predictors to form a Generalized Additive Mixed Model (GAMM). Rather than calling `glmer` like `gamm4` does, `stan_gamm4` essentially calls `stan_glmer`, which avoids the optimization issues that often crop up with GAMMs and provides better estimates for the uncertainty of the parameter estimates.

`stan_polr` Similar to `polr` in the **MASS** package in that it models an ordinal response, but the Bayesian model also implies a prior distribution on the unknown cutpoints. Can also be used to model binary outcomes, possibly while estimating an unknown exponent governing the probability of success.

`stan_betareg` Similar to `betareg` in that it models an outcome that is a rate (proportion) but, rather than performing maximum likelihood estimation, full Bayesian estimation is performed by default, with customizable prior distributions for all parameters.

Prior distributions

See priors help page for an overview of the various choices the user can make for prior distributions. The package vignettes also provide examples of using many of the available priors as well as more detailed descriptions of some of the novel priors used by **rstanarm**.

References

Bates, D., Maechler, M., Bolker, B., and Walker, S. (2015). Fitting linear mixed-Effects models using lme4. *Journal of Statistical Software*. 67(1), 1–48.

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. `http://stat.columbia.edu/~gelman/book/`

Gelman, A. and Hill, J. (2007). *Data Analysis Using Regression and Multilevel/Hierarchical Models.* Cambridge University Press, Cambridge, UK. `http://stat.columbia.edu/~gelman/arm/`

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

Vehtari, A., Gelman, A., and Gabry, J. (2016a). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*. Advance online publication. doi:10.1007/s11222-016-9696-4. arXiv preprint: `http://arxiv.org/abs/1507.04544/`

See Also

- `stanreg-objects` and `stanreg-methods` for details on the fitted model objects returned by the modeling functions.
- The custom `plot` and `pp_check` methods for the various plots that can be used to explore and check fitted models.
- `http://mc-stan.org/` for more information on the Stan C++ package used by **rstanarm** for model fitting.
- `https://github.com/stan-dev/rstanarm/issues/` to submit a bug report or feature request.
- `https://groups.google.com/forum/#!forum/stan-users/` to ask a question about **rstanarm** on the Stan-users forum.

`adapt_delta` *Target average acceptance probability*

Description

Target average acceptance probability

Details

For the No-U-Turn Sampler (NUTS), the variant of Hamiltonian Monte Carlo used used by **rstanarm**, `adapt_delta` is the target average proposal acceptance probability for adaptation. `adapt_delta` is ignored if `algorithm` is not `"sampling"`.

The default value of `adapt_delta` is 0.95, except when the prior for the regression coefficients is `R2`, `hs`, or `hs_plus`, in which case the default is 0.99.

In general you should not need to change adapt_delta unless you see a warning message about divergent transitions, in which case you can increase adapt_delta from the default to a value *closer* to 1 (e.g. from 0.95 to 0.99, or from 0.99 to 0.999, etc). The step size used by the numerical integrator is a function of adapt_delta in that increasing adapt_delta will result in a smaller step size and fewer divergences. Increasing adapt_delta will typically result in a slower sampler, but it will always lead to a more robust sampler.

References

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

as.matrix.stanreg   *Extract the posterior sample*

Description

For models fit using MCMC (algorithm="sampling"), the posterior sample —the post-warmup draws from the posterior distribution— can be extracted from a fitted model object as a matrix, data frame, or array. The as.matrix and as.data.frame methods merge all chains together, whereas the as.array method keeps the chains separate. For models fit using optimization ("optimizing") or variational inference ("meanfield" or "fullrank"), there is no posterior sample but rather a matrix (or data frame) of 1000 draws from either the asymptotic multivariate Gaussian sampling distribution of the parameters or the variational approximation to the posterior distribution.

Usage

```
## S3 method for class 'stanreg'
as.matrix(x, ..., pars = NULL, regex_pars = NULL)

## S3 method for class 'stanreg'
as.array(x, ..., pars = NULL, regex_pars = NULL)

## S3 method for class 'stanreg'
as.data.frame(x, ..., pars = NULL, regex_pars = NULL)
```

Arguments

x	A fitted model object returned by one of the **rstanarm** modeling functions. See stanreg-objects.
...	Ignored.
pars	An optional character vector of parameter names.
regex_pars	An optional character vector of regular expressions to use for parameter selection. regex_pars can be used in place of pars or in addition to pars. Currently, all functions that accept a regex_pars argument ignore it for models fit using optimization.

Value

A matrix, data.frame, or array, the dimensions of which depend on `pars` and `regex_pars`, as well as the model and estimation algorithm (see the Description section above).

See Also

`stanreg-methods`

Examples

```
if (!exists("example_model")) example(example_model)
# Extract posterior sample after MCMC
draws <- as.matrix(example_model)
print(dim(draws))

# For example, we can see that the median of the draws for the intercept
# is the same as the point estimate rstanarm uses
print(median(draws[, "(Intercept)"]))
print(example_model$coefficients[["(Intercept)"]])

# The as.array method keeps the chains separate
draws_array <- as.array(example_model)
print(dim(draws_array)) # iterations x chains x parameters

# Extract draws from asymptotic Gaussian sampling distribution
# after optimization
fit <- stan_glm(mpg ~ wt, data = mtcars, algorithm = "optimizing")
draws <- as.data.frame(fit)
print(colnames(draws))
print(nrow(draws)) # 1000 draws are taken

# Extract draws from variational approximation to the posterior distribution
fit2 <- update(fit, algorithm = "meanfield")
draws <- as.data.frame(fit2, pars = "wt")
print(colnames(draws))
print(nrow(draws)) # 1000 draws are taken
```

`example_model` *Example model*

Description

A model for use in **rstanarm** examples.

Format

Calling `example("example_model")` will run the model in the Examples section, below, and the resulting stanreg object will then be available in the global environment. The `chains` and `iter` arguments are specified to make this example be small in size. In practice, we recommend that they be left unspecified in order to use the default values (4 and 2000 respectively) or increased if there are convergence problems. The `cores` argument is optional and on a multicore system, the user may well want to set that equal to the number of chains being executed.

See Also

`cbpp` for a description of the data.

Examples

```
example_model <-
  stan_glmer(cbind(incidence, size - incidence) ~ size + period + (1|herd),
             data = lme4::cbpp, family = binomial,
             # this next line is only to keep the example small in size!
             chains = 2, cores = 1, seed = 12345, iter = 500)
example_model
```

`log_lik.stanreg` *Pointwise log-likelihood matrix*

Description

For models fit using MCMC only, the `log_lik` method returns the S by N pointwise log-likelihood matrix, where S is the size of the posterior sample and N is the number of data points.

Usage

```
## S3 method for class 'stanreg'
log_lik(object, newdata = NULL, offset = NULL, ...)
```

Arguments

`object`	A fitted model object returned by one of the **rstanarm** modeling functions. See `stanreg-objects`.
`newdata`	An optional data frame of new data (e.g. holdout data) to use when evaluating the log-likelihood. See the description of `newdata` for `posterior_predict`.
`offset`	A vector of offsets. Only required if `newdata` is specified and an `offset` was specified when fitting the model.
`...`	Currently ignored.

Value

An S by N matrix, where S is the size of the posterior sample and N is the number of data points.

Examples

```
roaches$roach100 <- roaches$roach1 / 100
fit <- stan_glm(
   y ~ roach100 + treatment + senior,
   offset = log(exposure2),
   data = roaches,
   family = poisson(link = "log"),
   prior = normal(0, 2.5),
   prior_intercept = normal(0, 10),
   iter = 500 # to speed up example
)
ll <- log_lik(fit)
dim(ll)
all.equal(ncol(ll), nobs(fit))

# using newdata argument
nd <- roaches[1:2, ]
nd$treatment[1:2] <- c(0, 1)
ll2 <- log_lik(fit, newdata = nd, offset = c(0, 0))
head(ll2)
dim(ll2)
all.equal(ncol(ll2), nrow(nd))
```

`loo.stanreg` *Information criteria and cross-validation*

Description

For models fit using MCMC, compute approximate leave-one-out cross-validation (LOO, LOOIC) or, less preferably, the Widely Applicable Information Criterion (WAIC) using the **loo** package. Exact K-fold cross-validation is also available. Compare two or more models using the `compare_models` function. **Note:** these functions are not guaranteed to work properly unless the `data` argument was specified when the model was fit.

Usage

```
## S3 method for class 'stanreg'
loo(x, ..., k_threshold = NULL)

## S3 method for class 'stanreg'
waic(x, ...)

kfold(x, K = 10, save_fits = FALSE)

compare_models(..., loos = list())
```

Arguments

x	A fitted model object returned by one of the **rstanarm** modeling functions. See `stanreg-objects`.
...	For the `loo` method, `...` can be used to pass optional arguments (e.g. `cores`) to `psislw`. For `compare_models`, `...` should contain two or more objects returned by the `loo`, `kfold`, or `waic` method (see the **Examples** section, below).
k_threshold	Threshold for flagging estimates of the Pareto shape parameters k estimated by `loo`. See the *How to proceed when* `loo` *gives warnings* section, below, for details.
K	For `kfold`, the number of subsets of equal (if possible) size into which the data will be randomly partitioned for performing K-fold cross-validation. The model is refit `K` times, each time leaving out one of the `K` subsets. If `K` is equal to the total number of observations in the data then K-fold cross-validation is equivalent to exact leave-one-out cross-validation.
save_fits	If `TRUE`, a component 'fits' is added to the returned object to store the cross-validated stanreg objects and the indices of the omitted observations for each fold. Defaults to `FALSE`.
loos	For `compare_models`, a list of two or more objects returned by the `loo`, `kfold`, or `waic` method. This argument can be used as an alternative to passing these objects via `...`.

Value

The `loo` and `waic` methods return an object of class 'loo'. See the **Value** section in `loo` and `waic` (from the **loo** package) for details on the structure of these objects.

`kfold` returns an object with has classes 'kfold' and 'loo' that has a similar structure as the objects returned by the `loo` and `waic` methods.

`compare_models` returns a vector or matrix with class 'compare.loo'. See the **Comparing models** section below for more details.

Approximate LOO CV

The `loo` method for stanreg objects provides an interface to the **loo** package for approximate leave-one-out cross-validation (LOO). The LOO Information Criterion (LOOIC) has the same purpose as the Akaike Information Criterion (AIC) that is used by frequentists. Both are intended to estimate the expected log predictive density (ELPD) for a new dataset. However, the AIC ignores priors and assumes that the posterior distribution is multivariate normal, whereas the functions from the **loo** package do not make this distributional assumption and integrate over uncertainty in the parameters. This only assumes that any one observation can be omitted without having a major effect on the posterior distribution, which can be judged using the diagnostic plot provided by the `plot.loo` method and the warnings provided by the `print.loo` method (see the *How to Use the rstanarm Package* vignette for an example of this process).

How to proceed when `loo` gives warnings (k_threshold):

The k_threshold argument to the loo method for **rstanarm** models is provided as a possible remedy when the diagnostics reveal problems stemming from the posterior's sensitivity to particular observations. Warnings about Pareto k estimates indicate observations for which the approximation to LOO is problematic (this is described in detail in Vehtari, Gelman, and Gabry (2016) and the **loo** package documentation). The k_threshold argument can be used to set the k value above which an observation is flagged. If k_threshold is not NULL and there are J observations with k estimates above k_threshold then when loo is called it will refit the original model J times, each time leaving out one of the J problematic observations. The pointwise contributions of these observations to the total ELPD are then computed directly and substituted for the previous estimates from these J observations that are stored in the object created by loo.

Note: in the warning messages issued by loo about large Pareto k estimates we recommend setting k_threshold to at least 0.7. There is a theoretical reason, explained in Vehtari, Gelman, and Gabry (2016), for setting the threshold to the stricter value of 0.5, but in practice they find that errors in the LOO approximation start to increase non-negligibly when $k > 0.7$.

K-fold CV

The kfold function performs exact K-fold cross-validation. First the data are randomly partitioned into K subsets of equal (or as close to equal as possible) size. Then the model is refit K times, each time leaving out one of the K subsets. If K is equal to the total number of observations in the data then K-fold cross-validation is equivalent to exact leave-one-out cross-validation (to which loo is an efficient approximation). The compare_models function is also compatible with the objects returned by kfold.

Comparing models

compare_models is a wrapper around the compare function in the **loo** package. Before calling compare, compare_models performs some extra checks to make sure the **rstanarm** models are suitable for comparison. These extra checks include verifying that all models to be compared were fit using the same outcome variable and likelihood family.

If exactly two models are being compared then compare_models returns a vector containing the difference in expected log predictive density (ELPD) between the models and the standard error of that difference (the documentation for compare has additional details about the calculation of the standard error of the difference). The difference in ELPD will be negative if the expected out-of-sample predictive accuracy of the first model is higher. If the difference is be positive then the second model is preferred.

If more than two models are being compared then compare_models returns a matrix with one row per model. This matrix summarizes the objects and arranges them in descending order according to expected out-of-sample predictive accuracy. That is, the first row of the matrix will be for the model with the largest ELPD (smallest LOOIC).

References

Vehtari, A., Gelman, A., and Gabry, J. (2016a). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*. Advance online publication. doi:10.1007/s11222-016-9696-4. arXiv preprint: http://arxiv.org/abs/1507.04544/

See Also

- The various **rstanarm** vignettes for more examples of using `loo` and `compare_models`.
- `loo-package` (in particular the *PSIS-LOO* section) for details on the computations implemented by the **loo** package and the interpretation of the Pareto k estimates displayed when using the `plot.loo` method.
- `log_lik.stanreg` to directly access the pointwise log-likelihood matrix.

Examples

```
fit1 <- stan_glm(mpg ~ wt, data = mtcars)
fit2 <- stan_glm(mpg ~ wt + cyl, data = mtcars)

# compare on LOOIC
(loo1 <- loo(fit1, cores = 2))
loo2 <- loo(fit2, cores = 2)
plot(loo2)

# when comparing exactly two models, the reported 'elpd_diff' will be
# positive if the expected predictive accuracy for the second model is higher
compare_models(loo1, loo2) # or compare_models(loos = list(loo1, loo2))

# when comparing three or more models they are ordered by expected
# predictive accuracy
fit3 <- stan_glm(mpg ~ ., data = mtcars)
loo3 <- loo(fit3, k_threshold = 0.7, cores = 2)
compare_models(loo1, loo2, loo3)

# 10-fold cross-validation
(kfold1 <- kfold(fit1, K = 10))
kfold2 <- kfold(fit2, K = 10)
compare_models(kfold1, kfold2)
```

`loo_predict.stanreg` *Compute weighted expectations using LOO*

Description

These functions are wrappers around the `E_loo` function (**loo** package).

Usage

```
## S3 method for class 'stanreg'
loo_predict(object, type = c("mean", "var", "quantile"),
  probs = 0.5, ..., lw)
```

```
## S3 method for class 'stanreg'
loo_linpred(object, type = c("mean", "var", "quantile"),
  probs = 0.5, transform = FALSE, ..., lw)

## S3 method for class 'stanreg'
loo_predictive_interval(object, prob = 0.9, ..., lw)
```

Arguments

`object`	A fitted model object returned by one of the **rstanarm** modeling functions. See `stanreg-objects`.
`type`	The type of expectation to compute. The options are `"mean"`, `"var"` (variance), and `"quantile"`.
`probs`	A vector of probabilities. Ignored unless `type` is `"quantile"`.
`...`	Optional arguments passed to `psislw`. If `lw` is specified these arguments are ignored.
`lw`	An optional matrix of (smoothed) log-weights. If `lw` is missing then `psislw` is executed internally, which may be time consuming for models fit to very large datasets.
`transform`	Passed to `posterior_linpred`.
`prob`	For `loo_predictive_interval`, a scalar in $(0, 1)$ indicating the desired probability mass to include in the intervals. The default is `prob=0.9` (90% intervals).

Value

`loo_predict` and `loo_linpred` return a vector with one element per observation. The only exception is if `type="quantile"` and `length(probs) >= 2`, in which case a separate vector for each element of `probs` is computed and they are returned in a matrix with `length(probs)` rows and one column per observation.

`loo_predictive_interval` returns a matrix with one row per observation and two columns (like `predictive_interval`). `loo_predictive_interval(..., prob = p)` is equivalent to `loo_predict(..., type = "quan`
with `a = (1 - p)/2`, except it transposes the result and adds informative column names.

Examples

```
## Not run:
if (!exists("example_model")) example(example_model)
head(loo_predictive_interval(example_model, prob = 0.8, cores = 2))

# optionally, log-weights can be pre-computed and reused
psis <- loo::psislw(-log_lik(example_model), cores = 2)
loo_predictive_interval(example_model, prob = 0.8, lw = psis$lw_smooth)
loo_predict(example_model, type = "var", lw = psis$lw_smooth)

## End(Not run)
```

neg_binomial_2 *Family function for negative binomial GLMs*

Description

Specifies the information required to fit a Negative Binomial GLM in a similar way to `negative.binomial`. However, here the overdispersion parameter `theta` is not specified by the user and always estimated (really the *reciprocal* of the dispersion parameter is estimated). A call to this function can be passed to the `family` argument of `stan_glm` or `stan_glmer` to estimate a Negative Binomial model. Alternatively, the `stan_glm.nb` and `stan_glmer.nb` wrapper functions may be used, which call `neg_binomial_2` internally.

Usage

```
neg_binomial_2(link = "log")
```

Arguments

`link`	The same as for `poisson`, typically a character vector of length one among `"log"`, `"identity"`, and `"sqrt"`.

Value

An object of class `family` very similar to that of `poisson` but with a different family name.

Examples

```
if (!grepl("^sparc",  R.version$platform))
stan_glm(Days ~ Sex/(Age + Eth*Lrn), data = MASS::quine, seed = 123,
         family = neg_binomial_2, QR = TRUE, algorithm = "optimizing")

# or, equivalently, call stan_glm.nb() without specifying the family
```

pairs.stanreg *Pairs method for stanreg objects*

Description

Interface to **bayesplot**'s `mcmc_pairs` function for use with **rstanarm** models. Be careful not to specify too many parameters to include or the plot will be both hard to read and slow to render.

Usage

```
## S3 method for class 'stanreg'
pairs(x, pars = NULL, regex_pars = NULL, ...)
```

Arguments

`x`	A fitted model object returned by one of the **rstanarm** modeling functions. See `stanreg-objects`.
`pars`	An optional character vetor of parameter names. All parameters are included by default, but for models with more than just a few parameters it may be far too many to visualize on a small computer screen and also may require substantial computing time.
`regex_pars`	An optional character vector of regular expressions to use for parameter selection. `regex_pars` can be used in place of `pars` or in addition to `pars`. Currently, all functions that accept a `regex_pars` argument ignore it for models fit using optimization.
`...`	Optional arguments passed to `mcmc_pairs`. The `np`, `lp`, and `max_treedepth` arguments to `mcmc_pairs` are handled automatically by **rstanarm** and do not need to be specified by the user in `....` The arguments that can be specified in `...` include `transformations`, `diag_fun`, `off_diag_fun`, `diag_args`, `off_diag_args`, `condition`, and `np_style`. These arguments are documented thoroughly on the help page for `mcmc_pairs`.

Details

By default, the `mcmc_pairs` function in the **bayesplot** package plots some of the Markov chains (half, in the case of an even number of chains) in the panels above the diagonal and the other half in the panels below the diagonal. This can be changed using the `condition` argument along with the `pairs_condition` helper function. We provide an example in the **Examples** section, below, but for full details see the `mcmc_pairs` help page. In particular, if when you fit your model **rstanarm** issues warnings about convergence, divergent transitions, or transitions hitting the maximum treedepth, then it can sometimes be useful to use one of the NUTS sampler parameters/diagnostics for `condition`. The last few examples below demonstrate this feature.

Examples

```
if (!exists("example_model")) example(example_model)

bayesplot::color_scheme_set("purple")
pairs(example_model, pars = c("(Intercept)", "log-posterior"))

pairs(
  example_model,
  regex_pars = "herd:[2,7,9]",
  diag_fun = "dens",
  off_diag_fun = "hex"
)



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

pairs(
  fit,
  pars = c("wt", "sigma", "log-posterior"),
  transformations = list(sigma = "log"), # show log(sigma) instead of sigma
  off_diag_fun = "hex" # use hexagonal heatmaps instead of scatterplots
)


bayesplot::color_scheme_set("brightblue")
pairs(
  fit,
  pars = c("(Intercept)", "wt", "sigma", "log-posterior"),
  transformations = list(sigma = "log"),
 off_diag_args = list(size = 3/4, alpha = 1/3), # size and transparency of scatterplot points
 np_style = pairs_style_np(div_color = "black", div_shape = 2) # color and shape of the divergences
)

# Using the condition argument to show divergences above the diagonal
pairs(
  fit,
  pars = c("(Intercept)", "wt", "log-posterior"),
  condition = pairs_condition(nuts = "divergent__")
)

# Using the condition argument to divide iterations by whether NUTS
# accept_stat__ is at least the median accept_stat__ (above diagonal) or less
# than the median accept_stat__ (below diagonal). divergences are still
# marked in red.
pairs(
  fit,
  pars = c("(Intercept)", "wt", "log-posterior"),
  condition = pairs_condition(nuts = "accept_stat__")
)
```

plot.stanreg   *Plot method for stanreg objects*

Description

The `plot` method for stanreg-objects provides a convenient interface to the MCMC module in the **bayesplot** package for plotting MCMC draws and diagnostics. It is also straightforward to use the functions from the **bayesplot** package directly rather than via the `plot` method. Examples of both methods of plotting are given below.

Usage

```
## S3 method for class 'stanreg'
plot(x, plotfun = "intervals", pars = NULL,
  regex_pars = NULL, ...)
```

Arguments

`x`	A fitted model object returned by one of the **rstanarm** modeling functions. See `stanreg-objects`.
`plotfun`	A character string naming the **bayesplot** MCMC function to use. The default is to call `mcmc_intervals`. `plotfun` can be specified either as the full name of a **bayesplot** plotting function (e.g. `"mcmc_hist"`) or can be abbreviated to the part of the name following the `"mcmc_"` prefix (e.g. `"hist"`). To get the names of all available MCMC functions see `available_mcmc`.
`pars`	An optional character vector of parameter names.
`regex_pars`	An optional character vector of regular expressions to use for parameter selection. `regex_pars` can be used in place of `pars` or in addition to `pars`. Currently, all functions that accept a `regex_pars` argument ignore it for models fit using optimization.
`...`	Additional arguments to pass to `plotfun` for customizing the plot. These are described on the help pages for the individual plotting functions. For example, the arguments accepted for the default `plotfun="intervals"` can be found at `mcmc_intervals`.

Value

Either a ggplot object that can be further customized using the **ggplot2** package, or an object created from multiple ggplot objects (e.g. a gtable object created by `arrangeGrob`).

See Also

- The vignettes in the **bayesplot** package for many examples.
- `MCMC-overview` (**bayesplot**) for links to the documentation for all the available plotting functions.
- `color_scheme_set` (**bayesplot**) to change the color scheme used for plotting.
- `pp_check` for graphical posterior predictive checks.
- `plot_nonlinear` for models with nonlinear smooth functions fit using `stan_gamm4`.

Examples

```
# Use rstanarm example model
if (!exists("example_model")) example(example_model)
fit <- example_model

#####################################
### Intervals and point estimates ###
```

```
#####################################
plot(fit) # same as plot(fit, "intervals"), plot(fit, "mcmc_intervals")

p <- plot(fit, pars = "size", regex_pars = "period",
          prob = 0.5, prob_outer = 0.9)
p + ggplot2::ggtitle("Posterior medians \n with 50% and 90% intervals")

# Shaded areas under densities
bayesplot::color_scheme_set("brightblue")
plot(fit, "areas", regex_pars = "period",
     prob = 0.5, prob_outer = 0.9)

# Make the same plot by extracting posterior draws and calling
# bayesplot::mcmc_areas directly
x <- as.array(fit, regex_pars = "period")
bayesplot::mcmc_areas(x, prob = 0.5, prob_outer = 0.9)


##################################
### Histograms & density plots ###
##################################
plot_title <- ggplot2::ggtitle("Posterior Distributions")
plot(fit, "hist", regex_pars = "period") + plot_title
plot(fit, "dens_overlay", pars = "(Intercept)",
     regex_pars = "period") + plot_title

####################
### Scatterplots ###
####################
bayesplot::color_scheme_set("teal")
plot(fit, "scatter", pars = paste0("period", 2:3))
plot(fit, "scatter", pars = c("(Intercept)", "size"),
     size = 3, alpha = 0.5) +
     ggplot2::stat_ellipse(level = 0.9)


####################################################
### Rhat, effective sample size, autocorrelation ###
####################################################
bayesplot::color_scheme_set("red")

# rhat
plot(fit, "rhat")
plot(fit, "rhat_hist")

# ratio of effective sample size to total posterior sample size
plot(fit, "neff")
plot(fit, "neff_hist")

# autocorrelation by chain
plot(fit, "acf", pars = "(Intercept)", regex_pars = "period")
plot(fit, "acf_bar", pars = "(Intercept)", regex_pars = "period")
```

```
##################
### Traceplots ###
##################
# NOTE: rstanarm doesn't store the warmup draws (to save space because they
# are not so essential for diagnosing the particular models implemented in
# rstanarm) so the iterations in the traceplot are post-warmup iterations

bayesplot::color_scheme_set("pink")
(trace <- plot(fit, "trace", pars = "(Intercept)"))

# change traceplot colors to ggplot defaults or custom values
trace + ggplot2::scale_color_discrete()
trace + ggplot2::scale_color_manual(values = c("maroon", "skyblue2"))

# changing facet layout
plot(fit, "trace", pars = c("(Intercept)", "period2"),
     facet_args = list(nrow = 2))
# same plot by calling bayesplot::mcmc_trace directly
x <- as.array(fit, pars = c("(Intercept)", "period2"))
bayesplot::mcmc_trace(x, facet_args = list(nrow = 2))


############
### More ###
############

# regex_pars examples
plot(fit, regex_pars = "herd:1\\]")
plot(fit, regex_pars = "herd:[279]")
plot(fit, regex_pars = "herd:[279]|period2")
plot(fit, regex_pars = c("herd:[279]", "period2"))


# For graphical posterior predictive checks see
# help("pp_check.stanreg")
```

posterior_interval.stanreg
Posterior uncertainty intervals

Description

For models fit using MCMC (`algorithm="sampling"`) or one of the variational approximations (`"meanfield"` or `"fullrank"`), the `posterior_interval` function computes Bayesian posterior uncertainty intervals. These intervals are often referred to as *credible* intervals, but we use the term *uncertainty* intervals to highlight the fact that wider intervals correspond to greater uncertainty.

Usage

```
## S3 method for class 'stanreg'
posterior_interval(object, prob = 0.9, type = "central",
  pars = NULL, regex_pars = NULL, ...)
```

Arguments

`object`	A fitted model object returned by one of the **rstanarm** modeling functions. See `stanreg-objects`.
`prob`	A number $p \in (0, 1)$ indicating the desired probability mass to include in the intervals. The default is to report 90% intervals (`prob=0.9`) rather than the traditionally used 95% (see Details).
`type`	The type of interval to compute. Currently the only option is `"central"` (see Details). A central $100p\%$ interval is defined by the $\alpha/2$ and $1 - \alpha/2$ quantiles, where $\alpha = 1 - p$.
`pars`	An optional character vector of parameter names.
`regex_pars`	An optional character vector of regular expressions to use for parameter selection. `regex_pars` can be used in place of `pars` or in addition to `pars`. Currently, all functions that accept a `regex_pars` argument ignore it for models fit using optimization.
`...`	Currently ignored.

Details

Interpretation: Unlike for a frenquentist confidence interval, it is valid to say that, conditional on the data and model, we believe that with probability p the value of a parameter is in its $100p\%$ posterior interval. This intuitive interpretation of Bayesian intervals is often erroneously applied to frequentist confidence intervals. See Morey et al. (2015) for more details on this issue and the advantages of using Bayesian posterior uncertainty intervals (also known as credible intervals).

Default 90% intervals: We default to reporting 90% intervals rather than 95% intervals for several reasons:

- Computational stability: 90% intervals are more stable than 95% intervals (for which each end relies on only 2.5% of the posterior draws).
- Relation to Type-S errors (Gelman and Carlin, 2014): 95% of the mass in a 90% central interval is above the lower value (and 95% is below the upper value). For a parameter θ, it is therefore easy to see if the posterior probability that $\theta > 0$ (or $\theta < 0$) is larger or smaller than 95%.

Of course, if 95% intervals are desired they can be computed by specifying `prob=0.95`.

Types of intervals: Currently `posterior_interval` only computes central intervals because other types of intervals are rarely useful for the models that **rstanarm** can estimate. Additional possibilities may be provided in future releases as more models become available.

Value

A matrix with two columns and as many rows as model parameters (or the subset of parameters specified by `pars` and/or `regex_pars`). For a given value of `prob`, p, the columns correspond to the lower and upper $100p\%$ interval limits and have the names $100\alpha/2\%$ and $100(1-\alpha/2)\%$, where $\alpha = 1 - p$. For example, if `prob=0.9` is specified (a 90% interval), then the column names will be `"5%"` and `"95%"`, respectively.

References

Gelman, A. and Carlin, J. (2014). Beyond power calculations: assessing Type S (sign) and Type M (magnitude) errors. *Perspectives on Psychological Science*. 9(6), 641–51.

Morey, R. D., Hoekstra, R., Rouder, J., Lee, M. D., and Wagenmakers, E. (2016). The fallacy of placing confidence in confidence intervals. *Psychonomic Bulletin & Review*. 23(1), 103–123.

See Also

`confint.stanreg`, which, for models fit using optimization, can be used to compute traditional confidence intervals.

`predictive_interval` for predictive intervals.

Examples

```
if (!exists("example_model")) example(example_model)
posterior_interval(example_model)
posterior_interval(example_model, regex_pars = "herd")
posterior_interval(example_model, pars = "period2", prob = 0.5)
```

---

posterior_linpred.stanreg
Posterior distribution of the linear predictor

Description

Extract the posterior draws of the linear predictor, possibly transformed by the inverse-link function. This function is occasionally useful, but it should be used sparingly. Inference and model checking should generally be carried out using the posterior predictive distribution (i.e., using `posterior_predict`).

Usage

```
## S3 method for class 'stanreg'
posterior_linpred(object, transform = FALSE,
  newdata = NULL, re.form = NULL, offset = NULL, XZ = FALSE, ...)
```

Arguments

`object`	A fitted model object returned by one of the **rstanarm** modeling functions. See `stanreg-objects`.
`transform`	Should the linear predictor be transformed using the inverse-link function? The default is `FALSE`, in which case the untransformed linear predictor is returned.
`newdata, re.form, offset`	Same as for `posterior_predict`.
`XZ`	If `TRUE` then instead of computing the linear predictor the design matrix `X` (or `cbind(X,Z)` for models with group-specific terms) constructed from `newdata` is returned. The default is `FALSE`.
`...`	Currently ignored.

Value

The default is to return a `draws` by `nrow(newdata)` matrix of simulations from the posterior distribution of the (possibly transformed) linear predictor. The exception is if the argument `XZ` is set to `TRUE` (see the `XZ` argument description above).

See Also

`posterior_predict` to draw from the posterior predictive distribution of the outcome, which is typically preferable.

Examples

```
if (!exists("example_model")) example(example_model)
print(family(example_model))

# linear predictor on log-odds scale
linpred <- posterior_linpred(example_model)
colMeans(linpred)

# probabilities
probs <- posterior_linpred(example_model, transform = TRUE)
colMeans(probs)

# not conditioning on any group-level parameters
probs2 <- posterior_linpred(example_model, transform = TRUE, re.form = NA)
apply(probs2, 2, median)
```

`posterior_predict.stanreg` *Draw from posterior predictive distribution*

Description

The posterior predictive distribution is the distribution of the outcome implied by the model after using the observed data to update our beliefs about the unknown parameters in the model. Simulating data from the posterior predictive distribution using the observed predictors is useful for checking the fit of the model. Drawing from the posterior predictive distribution at interesting values of the predictors also lets us visualize how a manipulation of a predictor affects (a function of) the outcome(s). With new observations of predictor variables we can use the posterior predictive distribution to generate predicted outcomes.

Usage

```
## S3 method for class 'stanreg'
posterior_predict(object, newdata = NULL, draws = NULL,
  re.form = NULL, fun = NULL, seed = NULL, offset = NULL, ...)
```

Arguments

`object`	A fitted model object returned by one of the **rstanarm** modeling functions. See `stanreg-objects`.
`newdata`	Optionally, a data frame in which to look for variables with which to predict. If omitted, the model matrix is used. If `newdata` is provided and any variables were transformed (e.g. rescaled) in the data used to fit the model, then these variables must also be transformed in `newdata`. This only applies if variables were transformed before passing the data to one of the modeling functions and *not* if transformations were specified inside the model formula. Also see the Note section below for a note about using the `newdata` argument with with binomial models.
`draws`	An integer indicating the number of draws to return. The default and maximum number of draws is the size of the posterior sample.
`re.form`	If `object` contains `group-level` parameters, a formula indicating which group-level parameters to condition on when making predictions. `re.form` is specified in the same form as for `predict.merMod`. The default, `NULL`, indicates that all estimated group-level parameters are conditioned on. To refrain from conditioning on any group-level parameters, specify `NA` or `~0`. The `newdata` argument may include new *levels* of the grouping factors that were specified when the model was estimated, in which case the resulting posterior predictions marginalize over the relevant variables.
`fun`	An optional function to apply to the results. `fun` is found by a call to `match.fun` and so can be specified as a function object, a string naming a function, etc.
`seed`	An optional `seed` to use.
`offset`	A vector of offsets. Only required if `newdata` is specified and an `offset` argument was specified when fitting the model.
`...`	Currently ignored.

Value

A draws by nrow(newdata) matrix of simulations from the posterior predictive distribution. Each row of the matrix is a vector of predictions generated using a single draw of the model parameters from the posterior distribution. The returned matrix will also have class "ppd" to indicate it contains draws from the posterior predictive distribution.

Note

For binomial models with a number of trials greater than one (i.e., not Bernoulli models), if newdata is specified then it must include all variables needed for computing the number of binomial trials to use for the predictions. For example if the left-hand side of the model formula is cbind(successes, failures) then both successes and failures must be in newdata. The particular values of successes and failures in newdata do not matter so long as their sum is the desired number of trials. If the left-hand side of the model formula were cbind(successes, trials - successes) then both trials and successes would need to be in newdata, probably with successes set to 0 and trials specifying the number of trials. See the Examples section below and the *How to Use the rstanarm Package* for examples.

See Also

pp_check for graphical posterior predictive checks. Examples of posterior predictive checking can also be found in the **rstanarm** vignettes and demos.

predictive_error and predictive_interval.

Examples

```
if (!exists("example_model")) example(example_model)
yrep <- posterior_predict(example_model)
table(yrep)


# Using newdata
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
fit3 <- stan_glm(counts ~ outcome + treatment, family = poisson(link="log"),
                prior = normal(0, 1), prior_intercept = normal(0, 5))
nd <- data.frame(treatment = factor(rep(1,3)), outcome = factor(1:3))
ytilde <- posterior_predict(fit3, nd, draws = 500)
print(dim(ytilde))  # 500 by 3 matrix (draws by nrow(nd))
ytilde <- data.frame(count = c(ytilde),
                     outcome = rep(nd$outcome, each = 500))
ggplot2::ggplot(ytilde, ggplot2::aes(x=outcome, y=count)) +
  ggplot2::geom_boxplot() +
  ggplot2::ylab("predicted count")


# Using newdata with a binomial model.
# example_model is binomial so we need to set
# the number of trials to use for prediction.
# This could be a different number for each
# row of newdata or the same for all rows.
# Here we'll use the same value for all.
nd <- lme4::cbpp
print(formula(example_model))  # cbind(incidence, size - incidence) ~ ...
nd$size <- max(nd$size) + 1L   # number of trials
nd$incidence <- 0  # set to 0 so size - incidence = number of trials
ytilde <- posterior_predict(example_model, newdata = nd)


# Using fun argument to transform predictions
mtcars2 <- mtcars
mtcars2$log_mpg <- log(mtcars2$mpg)
fit <- stan_glm(log_mpg ~ wt, data = mtcars2)
ytilde <- posterior_predict(fit, fun = exp)
```

`posterior_vs_prior` *Juxtapose prior and posterior*

Description

Plot medians and central intervals comparing parameter draws from the prior and posterior distributions. If the plotted priors look different than the priors you think you specified it is likely either because of internal rescaling or the use of the `QR` argument (see the documentation for the `prior_summary` method for details on these special cases).

Usage

```
posterior_vs_prior(object, ...)

## S3 method for class 'stanreg'
posterior_vs_prior(object, pars = NULL, regex_pars = NULL,
  prob = 0.9, color_by = c("parameter", "vs", "none"),
  group_by_parameter = FALSE, facet_args = list(), ...)
```

Arguments

`object`	A fitted model object returned by one of the **rstanarm** modeling functions. See `stanreg-objects`.
`...`	The S3 generic uses `...` to pass arguments to any defined methods. For the method for stanreg objects, `...` is for arguments (other than `color`) passed to `geom_pointrange` to control the appearance of the plotted intervals.
`pars`	An optional character vector specifying a subset of parameters to display. Parameters can be specified by name or several shortcuts can be used. Using `pars="beta"` will restrict the displayed parameters to only the regression coefficients (without the intercept). `"alpha"` can also be used as a shortcut for

"(Intercept)". If the model has varying intercepts and/or slopes they can be selected using pars = "varying". If pars is NULL all parameters are selected. See Examples.

regex_pars	An optional character vector of regular expressions to use for parameter selection. regex_pars can be used in place of pars or in addition to pars. Currently, all functions that accept a regex_pars argument ignore it for models fit using optimization.
prob	A number $p \in (0, 1)$ indicating the desired posterior probability mass to include in the (central posterior) interval estimates displayed in the plot. The default is 0.9.
color_by	How should the estimates be colored? Use "parameter" to color by parameter name, "vs" to color the prior one color and the posterior another, and "none" to use no color. Except when color_by="none", a variable is mapped to the color aesthetic and it is therefore also possible to change the default colors by adding one of the various discrete color scales available in ggplot2 (scale_color_manual, scale_color_brewer, etc.). See Examples.
group_by_parameter	Should estimates be grouped together by parameter (TRUE) or by posterior and prior (FALSE, the default)?
facet_args	A named list of arguments passed to facet_wrap (other than the facets argument), e.g., nrow or ncol to change the layout, scales to allow axis scales to vary across facets, etc. See Examples.

Value

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

Examples

```
## Not run:
if (!exists("example_model")) example(example_model)
# display non-varying (i.e. not group-level) coefficients
posterior_vs_prior(example_model, pars = "beta")

# show group-level (varying) parameters and group by parameter
posterior_vs_prior(example_model, pars = "varying",
                   group_by_parameter = TRUE, color_by = "vs")

# group by parameter and allow axis scales to vary across facets
posterior_vs_prior(example_model, regex_pars = "period",
                   group_by_parameter = TRUE, color_by = "none",
                   facet_args = list(scales = "free"))

# assign to object and customize with functions from ggplot2
(gg <- posterior_vs_prior(example_model, pars = c("beta", "varying"), prob = 0.8))

gg +
 ggplot2::geom_hline(yintercept = 0, size = 0.3, linetype = 3) +
 ggplot2::coord_flip() +
 ggplot2::ggtitle("Comparing the prior and posterior")

# compare very wide and very narrow priors using roaches example
# (see help(roaches, "rstanarm") for info on the dataset)
roaches$roach100 <- roaches$roach1 / 100
wide_prior <- normal(0, 10)
narrow_prior <- normal(0, 0.1)
fit_pois_wide_prior <- stan_glm(y ~ treatment + roach100 + senior,
                                offset = log(exposure2),
                                family = "poisson", data = roaches,
                                prior = wide_prior)
posterior_vs_prior(fit_pois_wide_prior, pars = "beta", prob = 0.5,
                   group_by_parameter = TRUE, color_by = "vs",
                   facet_args = list(scales = "free"))

fit_pois_narrow_prior <- update(fit_pois_wide_prior, prior = narrow_prior)
posterior_vs_prior(fit_pois_narrow_prior, pars = "beta", prob = 0.5,
                   group_by_parameter = TRUE, color_by = "vs",
                   facet_args = list(scales = "free"))


# look at cutpoints for ordinal model
fit_polr <- stan_polr(tobgp ~ agegp, data = esoph, method = "probit",
                      prior = R2(0.2, "mean"), init_r = 0.1)
(gg_polr <- posterior_vs_prior(fit_polr, regex_pars = "\\|", color_by = "vs",
                               group_by_parameter = TRUE))
# flip the x and y axes
gg_polr + ggplot2::coord_flip()

## End(Not run)
```

pp_check.stanreg *Graphical posterior predictive checks*

Description

Interface to the PPC (posterior predictive checking) module in the **bayesplot** package, providing various plots comparing the observed outcome variable y to simulated datasets y^{rep} from the posterior predictive distribution. The pp_check method for stanreg-objects prepares the arguments required for the specified **bayesplot** PPC plotting function and then calls that function. It is also straightforward to use the functions from the **bayesplot** package directly rather than via the pp_check method. Examples of both are given below.

Usage

```
## S3 method for class 'stanreg'
pp_check(object, plotfun = "dens_overlay", nreps = NULL,
  seed = NULL, ...)
```

Arguments

object	A fitted model object returned by one of the **rstanarm** modeling functions. See stanreg-objects.
plotfun	A character string naming the **bayesplot** PPC function to use. The default is to call ppc_dens_overlay. plotfun can be specified either as the full name of a **bayesplot** plotting function (e.g. "ppc_hist") or can be abbreviated to the part of the name following the "ppc_" prefix (e.g. "hist"). To get the names of all available PPC functions see available_ppc.
nreps	The number of y^{rep} datasets to generate from the posterior predictive distribution and show in the plots. The default depends on plotfun. For functions that plot each yrep dataset separately (e.g. ppc_hist), nreps defaults to a small value to make the plots readable. For functions that overlay many yrep datasets (e.g., ppc_dens_overlay) a larger number is used by default, and for other functions (e.g. ppc_stat) the default is to set nreps equal to the posterior sample size.
seed	An optional seed to pass to posterior_predict.
...	Additonal arguments passed to the **bayesplot** function called. For many plotting functions ... is optional, however for functions that require a group or x argument, these arguments should be specified in .... If specifying group and/or x, they can be provided as either strings naming variables (in which case they are searched for in the model frame) or as vectors containing the actual values of the variables. See the **Examples** section, below.

Value

pp_check returns a ggplot object that can be further customized using the **ggplot2** package.

Note

For binomial data, plots of y and y^{rep} show the proportion of 'successes' rather than the raw count. Also for binomial models see ppc_error_binned for binned residual plots.

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

- The vignettes in the **bayesplot** package for many examples. Examples of posterior predictive checks can also be found in the **rstanarm** vignettes and demos.
- PPC-overview (**bayesplot**) for links to the documentation for all the available plotting functions.
- posterior_predict for drawing from the posterior predictive distribution.
- color_scheme_set to change the color scheme of the plots.

Examples

```
fit <- stan_glmer(mpg ~ wt + am + (1|cyl), data = mtcars,
                  iter = 400, chains = 2) # just to keep example quick

# Compare distribution of y to distributions of multiple yrep datasets
pp_check(fit)
pp_check(fit, plotfun = "boxplot", nreps = 10, notch = FALSE)
pp_check(fit, plotfun = "hist", nreps = 3)


# Same plot (up to RNG noise) using bayesplot package directly
bayesplot::ppc_hist(y = mtcars$mpg, yrep = posterior_predict(fit, draws = 3))

# Check histograms of test statistics by level of grouping variable 'cyl'
pp_check(fit, plotfun = "stat_grouped", stat = "median", group = "cyl")

# Defining a custom test statistic
q25 <- function(y) quantile(y, probs = 0.25)
pp_check(fit, plotfun = "stat_grouped", stat = "q25", group = "cyl")

# Scatterplot of two test statistics
pp_check(fit, plotfun = "stat_2d", test = c("mean", "sd"))

# Scatterplot of y vs. average yrep
pp_check(fit, plotfun = "scatter_avg") # y vs. average yrep
# Same plot (up to RNG noise) using bayesplot package directly
bayesplot::ppc_scatter_avg(y = mtcars$mpg, yrep = posterior_predict(fit))

# Scatterplots of y vs. several individual yrep datasets
pp_check(fit, plotfun = "scatter", nreps = 3)

# Same plot (up to RNG noise) using bayesplot package directly
bayesplot::ppc_scatter(y = mtcars$mpg, yrep = posterior_predict(fit, draws = 3))

# yrep intervals with y points overlaid
# by default 1:length(y) used on x-axis but can also specify an x variable
pp_check(fit, plotfun = "intervals")
pp_check(fit, plotfun = "intervals", x = "wt") + ggplot2::xlab("wt")

# Same plot (up to RNG noise) using bayesplot package directly
bayesplot::ppc_intervals(y = mtcars$mpg, yrep = posterior_predict(fit),
                         x = mtcars$wt) + ggplot2::xlab("wt")

# predictive errors
pp_check(fit, plotfun = "error_hist", nreps = 6)
pp_check(fit, plotfun = "error_scatter_avg_vs_x", x = "wt") +
  ggplot2::xlab("wt")

# Example of a PPC for ordinal models (stan_polr)
fit2 <- stan_polr(tobgp ~ agegp, data = esoph, method = "probit",
                  prior = R2(0.2, "mean"), init_r = 0.1)
pp_check(fit2, plotfun = "bars", nreps = 500, prob = 0.5)
pp_check(fit2, plotfun = "bars_grouped", group = esoph$agegp,
         nreps = 500, prob = 0.5)
```

---

pp_validate *Model validation via simulation*

Description

The pp_validate function is based on the methods described in Cook, Gelman, and Rubin (2006) for validating software developed to fit particular Bayesian models. Here we take the perspective that models themselves are software and thus it is useful to apply this validation approach to individual models.

Usage

```
pp_validate(object, nreps = 20, seed = 12345, ...)
```

Arguments

object	A fitted model object returned by one of the **rstanarm** modeling functions. See stanreg-objects.
nreps	The number of replications to be performed. nreps must be sufficiently large so that the statistics described below in Details are meaningful. Depending on the model and the size of the data, running pp_validate may be slow. See also the Note section below for advice on avoiding numerical issues.
seed	A seed passed to Stan to use when refitting the model.
...	Currently ignored.

Details

We repeat nreps times the process of simulating parameters and data from the model and refitting the model to this simulated data. For each of the nreps replications we do the following:

1. Refit the model but *without* conditioning on the data (setting prior_PD=TRUE), obtaining draws θ^{true} from the *prior* distribution of the model parameters.
2. Given θ^{true}, simulate data y^* from the *prior* predictive distribution (calling posterior_predict on the fitted model object obtained in step 1).
3. Fit the model to the simulated outcome y^*, obtaining parameters θ^{post}.

For any individual parameter, the quantile of the "true" parameter value with respect to its posterior distribution *should* be uniformly distributed. The validation procedure entails looking for deviations from uniformity by computing statistics for a test that the quantiles are uniformly distributed. The absolute values of the computed test statistics are plotted for batches of parameters (e.g., non-varying coefficients are grouped into a batch called "beta", parameters that vary by group level are in batches named for the grouping variable, etc.). See Cook, Gelman, and Rubin (2006) for more details on the validation procedure.

Value

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

Note

In order to make it through `nreps` replications without running into numerical difficulties you may have to restrict the range for randomly generating initial values for parameters when you fit the *original* model. With any of **rstanarm**'s modeling functions this can be done by specifying the optional argument `init_r` as some number less than the default of 2.

References

Cook, S., Gelman, A., and Rubin, D. (2006). Validation of software for Bayesian models using posterior quantiles. *Journal of Computational and Graphical Statistics*. 15(3), 675–692.

See Also

`pp_check` for graphical posterior predictive checks and `posterior_predict` to draw from the posterior predictive distribution.

`color_scheme_set` to change the color scheme of the plot.

Examples

```
## Not run:
if (!exists("example_model")) example(example_model)
try(pp_validate(example_model)) # fails with default seed / priors

## End(Not run)
```

`predict.stanreg` *Predict method for stanreg objects*

Description

This method is primarily intended to be used only for models fit using optimization. For models fit using MCMC or one of the variational approximations, see `posterior_predict`.

Usage

```
## S3 method for class 'stanreg'
predict(object, ..., newdata = NULL, type = c("link",
  "response"), se.fit = FALSE)
```

Arguments

`object`	A fitted model object returned by one of the **rstanarm** modeling functions. See `stanreg-objects`.
`...`	Ignored.
`newdata`	Optionally, a data frame in which to look for variables with which to predict. If omitted, the model matrix is used.
`type`	The type of prediction. The default `'link'` is on the scale of the linear predictors; the alternative `'response'` is on the scale of the response variable.
`se.fit`	A logical scalar indicating if standard errors should be returned. The default is `FALSE`.

Value

A vector if `se.fit` is `FALSE` and a list if `se.fit` is `TRUE`.

See Also

`posterior_predict`

`predictive_error.stanreg`

In-sample or out-of-sample predictive errors

Description

This is a convenience function for computing $y - y^{rep}$ (in-sample, for observed y) or $y - \tilde{y}$ (out-of-sample, for new or held-out y). The method for stanreg objects calls `posterior_predict` internally, whereas the method for objects with class "ppd" accepts the matrix returned by `posterior_predict` as input and can be used to avoid multiple calls to `posterior_predict`.

Usage

```
## S3 method for class 'stanreg'
predictive_error(object, newdata = NULL, draws = NULL,
  re.form = NULL, seed = NULL, offset = NULL, ...)

## S3 method for class 'ppd'
predictive_error(object, y, ...)
```

Arguments

`object`	Either a fitted model object returned by one of the **rstanarm** modeling functions (a stanreg object) or, for the "ppd" method, a matrix of draws from the posterior predictive distribution returned by `posterior_predict`.

`newdata, draws, seed, offset, re.form`
: Optional arguments passed to `posterior_predict`. For binomial models, please see the **Note** section below if `newdata` will be specified.

`...`
: Currently ignored.

`y`
: For the `"ppd"` method only, a vector of y values the same length as the number of columns in the matrix used as `object`. The method for stanreg objects takes `y` directly from the fitted model object.

Value

A `draws` by `nrow(newdata)` matrix. If `newdata` is not specified then it will be `draws` by `nobs(object)`.

Note

The **Note** section in `posterior_predict` about `newdata` for binomial models also applies for `predictive_error`, with one important difference. For `posterior_predict` if the left-hand side of the model formula is `cbind(successes, failures)` then the particular values of `successes` and `failures` in `newdata` don't matter, only that they add to the desired number of trials. **This is not the case for** `predictive_error`. For `predictive_error` the particular value of `successes` matters because it is used as y when computing the error.

See Also

`posterior_predict` to draw from the posterior predictive distribution without computing predictive errors.

Examples

```
if (!exists("example_model")) example(example_model)
err1 <- predictive_error(example_model, draws = 50)
hist(err1)

# Using newdata with a binomial model
formula(example_model)
nd <- data.frame(
 size = c(10, 20),
 incidence = c(5, 10),
 period = factor(c(1,2)),
 herd = c(1, 15)
)
err2 <- predictive_error(example_model, newdata = nd, draws = 10, seed = 1234)

# stanreg vs ppd methods
fit <- stan_glm(mpg ~ wt, data = mtcars, iter = 300)
preds <- posterior_predict(fit, seed = 123)
all.equal(
  predictive_error(fit, seed = 123),
  predictive_error(preds, y = fit$y)
)
```

predictive_interval.stanreg

Predictive intervals

Description

For models fit using MCMC (`algorithm="sampling"`) or one of the variational approximations (`"meanfield"` or `"fullrank"`), the `predictive_interval` function computes Bayesian predictive intervals. The method for stanreg objects calls `posterior_predict` internally, whereas the method for objects of class `"ppd"` accepts the matrix returned by `posterior_predict` as input and can be used to avoid multiple calls to `posterior_predict`.

Usage

```
## S3 method for class 'stanreg'
predictive_interval(object, prob = 0.9, newdata = NULL,
  draws = NULL, re.form = NULL, fun = NULL, seed = NULL,
  offset = NULL, ...)

## S3 method for class 'ppd'
predictive_interval(object, prob = 0.9, ...)
```

Arguments

`object`	Either a fitted model object returned by one of the **rstanarm** modeling functions (a stanreg object) or, for the `"ppd"` method, a matrix of draws from the posterior predictive distribution returned by `posterior_predict`.
`prob`	A number $p \in (0, 1)$ indicating the desired probability mass to include in the intervals. The default is to report 90% intervals (`prob=0.9`) rather than the traditionally used 95% (see Details).
`newdata, draws, fun, offset, re.form, seed`	Passed to `posterior_predict`.
`...`	Currently ignored.

Value

A matrix with two columns and as many rows as are in `newdata`. If `newdata` is not provided then the matrix will have as many rows as the data used to fit the model. For a given value of `prob`, p, the columns correspond to the lower and upper $100p\%$ central interval limits and have the names $100\alpha/2\%$ and $100(1-\alpha/2)\%$, where $\alpha = 1 - p$. For example, if `prob=0.9` is specified (a 90% interval), then the column names will be `"5%"` and `"95%"`, respectively.

See Also

`predictive_error`, `posterior_predict`, `posterior_interval`

Examples

```
fit <- stan_glm(mpg ~ wt, data = mtcars, iter = 300)
predictive_interval(fit)
predictive_interval(fit, newdata = data.frame(wt = range(mtcars$wt)),
                    prob = 0.5)

# stanreg vs ppd methods
preds <- posterior_predict(fit, seed = 123)
all.equal(
  predictive_interval(fit, seed = 123),
  predictive_interval(preds)
)
```

`print.stanreg` *Print method for stanreg objects*

Description

The `print` method for stanreg objects displays a compact summary of the fitted model. See the Details section below for a description of the printed output. For additional summary statistics and diagnostics use the `summary` method.

Usage

```
## S3 method for class 'stanreg'
print(x, digits = 1, ...)
```

Arguments

`x`	A fitted model object returned by one of the **rstanarm** modeling functions. See `stanreg-objects`.
`digits`	Number of digits to use for formatting numbers.
`...`	Ignored.

Details

Point estimates: Regardless of the estimation algorithm, point estimates are medians computed from simulations. For models fit using MCMC (`"sampling"`) the posterior sample is used. For optimization (`"optimizing"`), the simulations are generated from the asymptotic Gaussian sampling distribution of the parameters. For the `"meanfield"` and `"fullrank"` variational approximations, draws from the variational approximation to the posterior are used. In all cases, the point estimates reported are the same as the values returned by `coef`.

Uncertainty estimates: The standard deviations reported (labeled MAD_SD in the print output) are computed from the same set of draws described above and are proportional to the median absolute deviation (`mad`) from the median. Compared to the raw posterior standard deviation, the MAD_SD will be more robust for long-tailed distributions. These are the same as the values returned by `se`.

Additional output: For models fit using MCMC or a variational approximation, the median and MAD_SD are also reported for mean_PPD, the sample average ($X = \bar{X}$) posterior predictive distribution of the outcome.

For GLMs with group-specific terms (see `stan_glmer`) the printed output also shows point estimates of the standard deviations of the group effects (and correlations if there are both intercept and slopes that vary by group).

For analysis of variance models (see `stan_aov`) models, an ANOVA-like table is also displayed.

Value

Returns `x`, invisibly.

See Also

`summary.stanreg`, `stanreg-methods`

`priors` *Prior distributions and options*

Description

The functions described on this page are used to specify the prior-related arguments of the various modeling functions in the **rstanarm** package (to view the priors used for an existing model see `prior_summary`). The default priors used in the various **rstanarm** modeling functions are intended to be *weakly informative* in that they provide moderate regularlization and help stabilize computation. For many applications the defaults will perform well, but prudent use of more informative priors is encouraged. Uniform prior distributions are possible (e.g. by setting `stan_glm`'s `prior` argument to `NULL`) but, unless the data is very strong, they are not recommended and are *not* non-informative, giving the same probability mass to implausible values as plausible ones.

Usage

```
normal(location = 0, scale = NULL, autoscale = TRUE)

student_t(df = 1, location = 0, scale = NULL, autoscale = TRUE)

cauchy(location = 0, scale = NULL, autoscale = TRUE)

hs(df = 3, global_df = 1, global_scale = 1)

hs_plus(df1 = 3, df2 = 3, global_df = 1, global_scale = 1)

laplace(location = 0, scale = NULL, autoscale = TRUE)

lasso(df = 1, location = 0, scale = NULL, autoscale = TRUE)

product_normal(df = 2, location = 0, scale = 1)
```

```
exponential(rate = 1, autoscale = TRUE)

decov(regularization = 1, concentration = 1, shape = 1, scale = 1)

dirichlet(concentration = 1)

R2(location = NULL, what = c("mode", "mean", "median", "log"))
```

Arguments

`location`	Prior location. In most cases, this is the prior mean, but for `cauchy` (which is equivalent to `student_t` with `df=1`), the mean does not exist and `location` is the prior median. The default value is 0, except for `R2` which has no default value for `location`. For `R2`, `location` pertains to the prior location of the R^2 under a Beta distribution, but the interpretation of the `location` parameter depends on the specified value of the `what` argument (see the *R2 family* section in **Details**).
`scale`	Prior scale. The default depends on the family (see **Details**).
`autoscale`	A logical scalar, defaulting to `TRUE`. If `TRUE` then the scales of the priors on the intercept and regression coefficients may be additionally modified internally by **rstanarm** in the following cases. First, for Gaussian models only, the prior scales for the intercept, coefficients, and the auxiliary parameter `sigma` (error standard deviation) are multiplied by `sd(y)`. Additionally — not only for Gaussian models — if the `QR` argument to the model fitting function (e.g. `stan_glm`) is `FALSE` then: for a predictor with only one value nothing is changed; for a predictor `x` with exactly two unique values, we take the user-specified (or default) scale(s) for the selected priors and divide by the range of `x`; for a predictor `x` with more than two unique values, we divide the prior scale(s) by `sd(x)`.
`df, df1, df2`	Prior degrees of freedom. The default is 1 for `student_t`, in which case it is equivalent to `cauchy`. For the hierarchical shrinkage priors (`hs` and `hs_plus`) the degrees of freedom parameter(s) default to 3. For the `product_normal` prior, the degrees of freedom parameter must be an integer (vector) that is at least 2 (the default).
`global_df, global_scale`	Optional arguments for the hierarchical shrinkage priors. See the *Hierarchical shrinkage family* section below.
`rate`	Prior rate for the exponential distribution. Defaults to `1`. For the exponential distribution, the rate parameter is the *reciprocal* of the mean.
`regularization`	Exponent for an LKJ prior on the correlation matrix in the `decov` prior. The default is 1, implying a joint uniform prior.
`concentration`	Concentration parameter for a symmetric Dirichlet distribution. The default is 1, implying a joint uniform prior.
`shape`	Shape parameter for a gamma prior on the scale parameter in the `decov` prior. If `shape` and `scale` are both 1 (the default) then the gamma prior simplifies to the unit-exponential distribution.

`what`	A character string among `'mode'` (the default), `'mean'`, `'median'`, or `'log'` indicating how the `location` parameter is interpreted in the `LKJ` case. If `'log'`, then `location` is interpreted as the expected logarithm of the R^2 under a Beta distribution. Otherwise, `location` is interpreted as the `what` of the R^2 under a Beta distribution. If the number of predictors is less than or equal to two, the mode of this Beta distribution does not exist and an error will prompt the user to specify another choice for `what`.

Details

The details depend on the family of the prior being used:

Student t family: Family members:

- `normal(location, scale)`
- `student_t(df, location, scale)`
- `cauchy(location, scale)`

Each of these functions also takes an argument `autoscale`.

For the prior distribution for the intercept, `location`, `scale`, and `df` should be scalars. For the prior for the other coefficients they can either be vectors of length equal to the number of coefficients (not including the intercept), or they can be scalars, in which case they will be recycled to the appropriate length. As the degrees of freedom approaches infinity, the Student t distribution approaches the normal distribution and if the degrees of freedom are one, then the Student t distribution is the Cauchy distribution.

If `scale` is not specified it will default to 10 for the intercept and 2.5 for the other coefficients, unless the probit link function is used, in which case these defaults are scaled by a factor of `dnorm(0)/dlogis(0)`, which is roughly 1.6.

If the `autoscale` argument is `TRUE` (the default), then the scales will be further adjusted as described above in the documentation of the `autoscale` argument in the **Arguments** section.

Hierarchical shrinkage family: Family members:

- `hs(df, global_df, global_scale)`
- `hs_plus(df1, df2, global_df, global_scale)`

The hierarchical shrinkage priors are normal with a mean of zero and a standard deviation that is also a random variable. The traditional hierarchical shrinkage prior utilizes a standard deviation that is distributed half Cauchy with a median of zero and a scale parameter that is also half Cauchy. This is called the "horseshoe prior". The hierarchical shrinkage (`hs`) prior in the **rstanarm** package instead utilizes a half Student t distribution for the standard deviation (with 3 degrees of freedom by default), as described by Piironen and Vehtari (2015). It is possible to change the `df` argument, the prior degrees of freedom, to obtain less or more shrinkage. Traditionally the standard deviation parameter is then scaled by the square root of a *global* half Cauchy parameter, although **rstanarm** allows setting `global_df` and `global_scale` arguments, in which case this global parameter is distributed half Student t with degrees of freedom `global_df` and scale `global_scale`.

The hierarhical shrinkpage plus (`hs_plus`) prior is a normal with a mean of zero and a standard deviation that is distributed as the product of two independent half Student t parameters (both with 3 degrees of freedom (`df1`, `df2`) by default) that are each scaled in a similar way to the `hs` prior.

The hierarchical shrinkage priors have very tall modes and very fat tails. Consequently, they tend to produce posterior distributions that are very concentrated near zero, unless the predictor has a strong influence on the outcome, in which case the prior has little influence. Hierarchical shrinkage priors often require you to increase the `adapt_delta` tuning parameter in order to diminish the number of divergent transitions. For more details on tuning parameters and divergent transitions see the Troubleshooting section of the *How to Use the rstanarm Package* vignette.

Laplace family: Family members:

- `laplace(location, scale)`
- `lasso(df, location, scale)`

Each of these functions also takes an argument `autoscale`.

The Laplace distribution is also known as the double-exponential distribution. It is a symmetric distribution with a sharp peak at its mean / median / mode and fairly long tails. This distribution can be motivated as a scale mixture of normal distributions and the remarks above about the normal distribution apply here as well.

The lasso approach to supervised learning can be expressed as finding the posterior mode when the likelihood is Gaussian and the priors on the coefficients have independent Laplace distributions. It is commonplace in supervised learning to choose the tuning parameter by cross-validation, whereas a more Bayesian approach would be to place a prior on "it", or rather its reciprocal in our case (i.e. *smaller* values correspond to more shrinkage toward the prior location vector). We use a chi-square prior with degrees of freedom equal to that specified in the call to `lasso` or, by default, 1. The expectation of a chi-square random variable is equal to this degrees of freedom and the mode is equal to the degrees of freedom minus 2, if this difference is positive.

It is also common in supervised learning to standardize the predictors before training the model. We do not recommend doing so. Instead, it is better to specify `autoscale = TRUE` (the default value), which will adjust the scales of the priors according to the dispersion in the variables. See the documentation of the `autoscale` argument above and also the `prior_summary` page for more information.

Product-normal family: Family members:

- `product_normal(df, location, scale)`

The product-normal distribution is the product of at least two independent normal variates each with mean zero, shifted by the `location` parameter. It can be shown that the density of a product-normal variate is symmetric and infinite at `location`, so this prior resembles a "spike-and-slab" prior for sufficiently large values of the `scale` parameter. For better or for worse, this prior may be appropriate when it is strongly believed (by someone) that a regression coefficient "is" equal to the `location`, parameter even though no true Bayesian would specify such a prior.

Each element of `df` must be an integer of at least 2 because these "degrees of freedom" are interpreted as the number of normal variates being multiplied and then shifted by `location` to yield the regression coefficient. Higher degrees of freedom produce a sharper spike at `location`.

Each element of `scale` must be a non-negative real number that is interpreted as the standard deviation of the normal variates being multiplied and then shifted by `location` to yield the regression coefficient. In other words, the elements of `scale` may differ, but the k-th standard deviation is presumed to hold for all the normal deviates that are multiplied together and shifted by the k-th element of `location` to yield the k-th regression coefficient. The elements of `scale` are not the prior standard deviations of the regression coefficients. The prior variance of the regression coefficients is equal to the scale raised to the power of 2 times the corresponding element of `df`. Thus, larger values of `scale` put more prior volume on values of the regression coefficient that are far from zero.

Dirichlet family: Family members:

- `dirichlet(concentration)`

The Dirichlet distribution is a multivariate generalization of the beta distribution. It is perhaps the easiest prior distribution to specify because the concentration parameters can be interpreted as prior counts (although they need not be integers) of a multinomial random variable.

The Dirichlet distribution is used in `stan_polr` for an implicit prior on the cutpoints in an ordinal regression model. More specifically, the Dirichlet prior pertains to the prior probability of observing each category of the ordinal outcome when the predictors are at their sample means. Given these prior probabilities, it is straightforward to add them to form cumulative probabilities and then use an inverse CDF transformation of the cumulative probabilities to define the cutpoints.

If a scalar is passed to the `concentration` argument of the `dirichlet` function, then it is replicated to the appropriate length and the Dirichlet distribution is symmetric. If `concentration` is a vector and all elements are 1, then the Dirichlet distribution is jointly uniform. If all concentration parameters are equal but greater than 1 then the prior mode is that the categories are equiprobable, and the larger the value of the identical concentration parameters, the more sharply peaked the distribution is at the mode. The elements in `concentration` can also be given different values to represent that not all outcome categories are a priori equiprobable.

Covariance matrices: Family members:

- `decov(regularization, concentration, shape, scale)`

(Also see vignette for `stan_glmer`)

Covariance matrices are decomposed into correlation matrices and variances. The variances are in turn decomposed into the product of a simplex vector and the trace of the matrix. Finally, the trace is the product of the order of the matrix and the square of a scale parameter. This prior on a covariance matrix is represented by the `decov` function.

The prior for a correlation matrix is called LKJ whose density is proportional to the determinant of the correlation matrix raised to the power of a positive regularization parameter minus one. If `regularization = 1` (the default), then this prior is jointly uniform over all correlation matrices of that size. If `regularization > 1`, then the identity matrix is the mode and in the unlikely case that `regularization < 1`, the identity matrix is the trough.

The trace of a covariance matrix is equal to the sum of the variances. We set the trace equal to the product of the order of the covariance matrix and the *square* of a positive scale parameter. The particular variances are set equal to the product of a simplex vector — which is non-negative and sums to 1 — and the scalar trace. In other words, each element of the simplex vector represents the proportion of the trace attributable to the corresponding variable.

A symmetric Dirichlet prior is used for the simplex vector, which has a single (positive) `concentration` parameter, which defaults to 1 and implies that the prior is jointly uniform over the space of simplex vectors of that size. If `concentration > 1`, then the prior mode corresponds to all variables having the same (proportion of total) variance, which can be used to ensure the the posterior variances are not zero. As the `concentration` parameter approaches infinity, this mode becomes more pronounced. In the unlikely case that `concentration < 1`, the variances are more polarized.

If all the variables were multiplied by a number, the trace of their covariance matrix would increase by that number squared. Thus, it is reasonable to use a scale-invariant prior distribution for the positive scale parameter, and in this case we utilize a Gamma distribution, whose `shape` and `scale` are both 1 by default, implying a unit-exponential distribution. Set the `shape` hyperparameter to some value greater than 1 to ensure that the posterior trace is not zero.

If `regularization`, `concentration`, `shape` and / or `scale` are positive scalars, then they are recycled to the appropriate length. Otherwise, each can be a positive vector of the appropriate length, but the appropriate length depends on the number of covariance matrices in the model and their sizes. A one-by-one covariance matrix is just a variance and thus does not have `regularization` or `concentration` parameters, but does have `shape` and `scale` parameters for the prior standard deviation of that variable.

R2 family: Family members:

- `R2(location, what)`

The `stan_lm`, `stan_aov`, and `stan_polr` functions allow the user to utilize a function called `R2` to convey prior information about all the parameters. This prior hinges on prior beliefs about the location of R^2, the proportion of variance in the outcome attributable to the predictors, which has a `Beta` prior with first shape hyperparameter equal to half the number of predictors and second shape hyperparameter free. By specifying `what` to be the prior mode (the default), mean, median, or expected log of R^2, the second shape parameter for this Beta distribution is determined internally. If `what = 'log'`, location should be a negative scalar; otherwise it should be a scalar on the $(0, 1)$ interval.

For example, if $R^2 = 0.5$, then the mode, mean, and median of the `Beta` distribution are all the same and thus the second shape parameter is also equal to half the number of predictors. The second shape parameter of the `Beta` distribution is actually the same as the shape parameter in the LKJ prior for a correlation matrix described in the previous subsection. Thus, the smaller is R^2, the larger is the shape parameter, the smaller are the prior correlations among the outcome and predictor variables, and the more concentrated near zero is the prior density for the regression coefficients. Hence, the prior on the coefficients is regularizing and should yield a posterior distribution with good out-of-sample predictions *if* the prior location of R^2 is specified in a reasonable fashion.

Value

A named list to be used internally by the **rstanarm** model fitting functions.

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. http://stat.columbia.edu/~gelman/book/

Gelman, A., Jakulin, A., Pittau, M. G., and Su, Y. (2008). A weakly informative default prior distribution for logistic and other regression models. *Annals of Applied Statistics*. 2(4), 1360–1383.

Piironen, J., and Vehtari, A. (2015). Projection predictive variable selection using Stan+R. http://arxiv.org/abs/1508.02502/

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

See Also

The various vignettes for the **rstanarm** package also discuss and demonstrate the use of some of the supported prior distributions.

Examples

```
fmla <- mpg ~ wt + qsec + drat + am

# Draw from prior predictive distribution (by setting prior_PD = TRUE)
prior_pred_fit <- stan_glm(fmla, data = mtcars, prior_PD = TRUE,
                           chains = 1, seed = 12345, iter = 250, # for speed only
                           prior = student_t(df = 4, 0, 2.5),
                           prior_intercept = cauchy(0,10),
                           prior_aux = exponential(1/2))
plot(prior_pred_fit, "hist")


# Can assign priors to names
N05 <- normal(0, 5)
fit <- stan_glm(fmla, data = mtcars, prior = N05, prior_intercept = N05)


# Visually compare normal, student_t, cauchy, laplace, and product_normal
compare_priors <- function(scale = 1, df_t = 2, xlim = c(-10, 10)) {
  dt_loc_scale <- function(x, df, location, scale) {
    1/scale * dt((x - location)/scale, df)
  }
  dlaplace <- function(x, location, scale) {
    0.5 / scale * exp(-abs(x - location) / scale)
  }
  dproduct_normal <- function(x, scale) {
    besselK(abs(x) / scale ^ 2, nu = 0) / (scale ^ 2 * pi)
  }
  stat_dist <- function(dist, ...) {
    ggplot2::stat_function(ggplot2::aes_(color = dist), ...)
  }
  ggplot2::ggplot(data.frame(x = xlim), ggplot2::aes(x)) +
    stat_dist("normal", size = .75, fun = dnorm,
              args = list(mean = 0, sd = scale)) +
    stat_dist("student_t", size = .75, fun = dt_loc_scale,
              args = list(df = df_t, location = 0, scale = scale)) +
    stat_dist("cauchy", size = .75, linetype = 2, fun = dcauchy,
              args = list(location = 0, scale = scale)) +
    stat_dist("laplace", size = .75, linetype = 2, fun = dlaplace,
              args = list(location = 0, scale = scale)) +
    stat_dist("product_normal", size = .75, linetype = 2, fun = dproduct_normal,
              args = list(scale = 1))
}
# Cauchy has fattest tails, followed by student_t, laplace, and normal
compare_priors()

# The student_t with df = 1 is the same as the cauchy
```

```
compare_priors(df_t = 1)

# Even a scale of 5 is somewhat large. It gives plausibility to rather
# extreme values
compare_priors(scale = 5, xlim = c(-20,20))

# If you use a prior like normal(0, 1000) to be "non-informative" you are
# actually saying that a coefficient value of e.g. -500 is quite plausible
compare_priors(scale = 1000, xlim = c(-1000,1000))
```

`prior_summary.stanreg` *Summarize the priors used for an rstanarm model*

Description

The `prior_summary` method provides a summary of the prior distributions used for the parameters in a given model. In some cases the user-specified prior does not correspond exactly to the prior used internally by **rstanarm** (see the sections below). Especially in these cases, but also in general, it can be much more useful to visualize the priors. Visualizing the priors can be done using the `posterior_vs_prior` function, or alternatively by fitting the model with the `prior_PD` argument set to `TRUE` (to draw from the prior predictive distribution instead of conditioning on the outcome) and then plotting the parameters.

Usage

```
## S3 method for class 'stanreg'
prior_summary(object, digits = 2, ...)
```

Arguments

`object`	A fitted model object returned by one of the **rstanarm** modeling functions. See `stanreg-objects`.
`digits`	Number of digits to use for rounding.
`...`	Currently ignored by the method for stanreg objects. The S3 generic uses `...` to pass arguments to any defined methods.

Value

A list of class "prior_summary.stanreg", which has its own print method.

Intercept (after predictors centered)

For **rstanarm** modeling functions that accept a `prior_intercept` argument, the specified prior for the intercept term applies to the intercept after **rstanarm** internally centers the predictors so they each have mean zero. The estimate of the intercept returned to the user correspond to the intercept with the predictors as specified by the user (unmodified by **rstanarm**), but when *specifying* the prior the intercept can be thought of as the expected outcome when the predictors are set to their means.

The only exception to this is for models fit with the `sparse` argument set to `TRUE` (which is only possible with a subset of the modeling functions and never the default).

Adjusted scales

For some models you may see "`adjusted scale`" in the printed output and adjusted scales included in the object returned by `prior_summary`. These adjusted scale values are the prior scales actually used by **rstanarm** and are computed by adjusting the prior scales specified by the user to account for the scales of the predictors (as described in the documentation for the `autoscale` argument). To disable internal prior scale adjustments set the `autoscale` argument to `FALSE` when setting a prior using one of the distributions that accepts an `autoscale` argument. For example, `normal(0, 5, autoscale=FALSE)` instead of just `normal(0, 5)`.

Coefficients in Q-space

For the models fit with an **rstanarm** modeling function that supports the `QR` argument (see e.g, `stan_glm`), if `QR` is set to `TRUE` then the prior distributions for the regression coefficients specified using the `prior` argument are not relative to the original predictor variables X but rather to the variables in the matrix Q obtained from the QR decomposition of X.

In particular, if `prior = normal(location,scale)`, then this prior on the coefficients in Q-space can be easily translated into a joint multivariate normal (MVN) prior on the coefficients on the original predictors in X. Letting θ denote the coefficients on Q and β the coefficients on X then if $\theta \sim N(\mu, \sigma)$ the corresponding prior on β is $\beta \sim MVN(R\mu, R'R\sigma^2)$, where μ and σ are vectors of the appropriate length. Technically **rstanarm** uses a scaled QR decomposition to ensure that the columns of the predictor matrix used to fit the model all have unit scale. The matrices actually used are $Q^* = Q\sqrt{n-1}$ and $R^* = \frac{1}{\sqrt{n-1}}R$ (see the documentation for the `QR` argument).

If you are interested in the prior on β implied by the prior on θ, we recommend visualizing it as described above in the **Description** section, which is simpler than working it out analytically.

See Also

`posterior_vs_prior`, `priors`

Examples

```
if (!exists("example_model")) example(example_model)
prior_summary(example_model)

priors <- prior_summary(example_model)
names(priors)
priors$prior$scale
priors$prior$adjusted_scale

# for a glm with adjusted scales (see Details, above), compare
# the default (rstanarm adjusting the scales) to setting
# autoscale=FALSE for prior on coefficients
fit <- stan_glm(mpg ~ wt + am, data = mtcars,
                prior = normal(0, c(2.5, 4)),
                prior_intercept = normal(0, 5),
                iter = 10, chains = 1) # only for demonstration
```

```
prior_summary(fit)

fit2 <- update(fit, prior = normal(0, c(2.5, 4), autoscale=FALSE),
               prior_intercept = normal(0, 5, autoscale=FALSE))
prior_summary(fit2)
```

---

rstanarm-datasets *Datasets for rstanarm examples*

Description

Small datasets for use in **rstanarm** examples and vignettes.

Format

`bball1970` Data on hits and at-bats from the 1970 Major League Baseball season for 18 players.

Source: Efron and Morris (1975).

18 obs. of 5 variables

- `Player` Player's last name
- `Hits` Number of hits in the first 45 at-bats of the season
- `AB` Number of at-bats (45 for all players)
- `RemainingAB` Number of remaining at-bats (different for most players)
- `RemainingHits` Number of remaining hits

`bball2006` Hits and at-bats for the entire 2006 American League season of Major League Baseball.

Source: Carpenter (2009)

302 obs. of 2 variables

- `y` Number of hits
- `K` Number of at-bats

`kidiq` Data from a survey of adult American women and their children (a subsample from the National Longitudinal Survey of Youth).

Source: Gelman and Hill (2007)

434 obs. of 4 variables

- `kid_score` Child's IQ score
- `mom_hs` Indicator for whether the mother has a high school degree
- `mom_iq` Mother's IQ score
- `mom_age` Mother's age

`mortality` Surgical mortality rates in 12 hospitals performing cardiac surgery in babies.

Source: Spiegelhalter et al. (1996).

12 obs. of 2 variables

- `y` Number of deaths
- `K` Number of surgeries

`radon` Data on radon levels in houses in the state of Minnesota.

Source: Gelman and Hill (2007)

919 obs. of 4 variables

- `log_radon` Radon measurement from the house (log scale)
- `log_uranium` Uranium level in the county (log scale)
- `floor` Indicator for radon measurement made on the first floor of the house (0 = basement, 1 = first floor)
- `county` County name (`factor`)

`roaches` Data on the efficacy of a pest management system at reducing the number of roaches in urban apartments.

Source: Gelman and Hill (2007)

262 obs. of 6 variables

- `y` Number of roaches caught
- `roach1` Pretreatment number of roaches
- `treatment` Treatment indicator
- `senior` Indicator for only eldery residents in building
- `exposure2` Number of days for which the roach traps were used

`tumors` Tarone (1982) provides a data set of tumor incidence in historical control groups of rats; specifically endometrial stromal polyps in female lab rats of type F344.

Source: Gelman and Hill (2007)

71 obs. of 2 variables

- `y` Number of rats with tumors
- `K` Number of rats

`wells` A survey of 3200 residents in a small area of Bangladesh suffering from arsenic contamination of groundwater. Respondents with elevated arsenic levels in their wells had been encouraged to switch their water source to a safe public or private well in the nearby area and the survey was conducted several years later to learn which of the affected residents had switched wells.

Souce: Gelman and Hill (2007)

3020 obs. of 5 variables

- `switch` Indicator for well-switching
- `arsenic` Arsenic level in respondent's well
- `dist` Distance (meters) from the respondent's house to the nearest well with safe drinking water.
- `association` Indicator for member(s) of household participate in community organizations
- `educ` Years of education (head of household)

References

Carpenter, B. (2009) Bayesian estimators for the beta-binomial model of batting ability. `http://lingpipe-blog.com/2009/09/23/`

Efron, B. and Morris, C. (1975) Data analysis using Stein's estimator and its generalizations. *Journal of the American Statistical Association* **70**(350), 311–319.

Gelman, A. and Hill, J. (2007). *Data Analysis Using Regression and Multilevel/Hierarchical Models.* Cambridge University Press, Cambridge, UK. http://stat.columbia.edu/~gelman/arm/

Spiegelhalter, D., Thomas, A., Best, N., & Gilks, W. (1996) BUGS 0.5 Examples. MRC Biostatistics Unit, Institute of Public health, Cambridge, UK.

Tarone, R. E. (1982) The use of historical control information in testing for a trend in proportions. *Biometrics* **38**(1):215–220.

Examples

```
# Using 'kidiq' dataset
fit <- stan_lm(kid_score ~ mom_hs * mom_iq, data = kidiq,
               prior = R2(location = 0.30, what = "mean"),
               # the next line is only to make the example go fast enough
               chains = 1, iter = 500, seed = 12345)
pp_check(fit, nreps = 20)

bayesplot::color_scheme_set("brightblue")
pp_check(fit, plotfun = "stat_grouped", stat = "median",
         group = factor(kidiq$mom_hs, labels = c("No HS", "HS")))
```

`rstanarm-deprecated` *Deprecated functions*

Description

These functions are deprecated and will be removed in a future release. The **Arguments** section below provides details on how the functionliaty obtained via each of the arguments has been replaced.

Usage

```
prior_options(prior_scale_for_dispersion = 5, min_prior_scale = 1e-12,
  scaled = TRUE)
```

Arguments

`prior_scale_for_dispersion, min_prior_scale, scaled`
Arguments to deprecated `prior_options` function. The functionality provided by the now deprecated `prior_options` function has been replaced as follows:

`prior_scale_for_dispersion` Instead of using the `prior_scale_for_dispersion` argument to `prior_options`, priors for these parameters can now be specified directly when calling `stan_glm` (or `stan_glmer`, etc.) using the new `prior_aux` argument.

`scaled` Instead of setting `prior_options(scaled=FALSE)`, internal rescaling is now toggled using the new `autoscale` arguments to `normal`, `student_t`, and `cauchy` (the other prior distributions do not support 'autoscale').

min_prior_scale No replacement. min_prior_scale (the minimum possible scale parameter value that be used for priors) is now fixed to 1e-12.

shinystan *Using the ShinyStan GUI with rstanarm models*

Description

The ShinyStan interface provides visual and numerical summaries of model parameters and convergence diagnostics.

Details

The launch_shinystan function will accept a stanreg object as input. Currently, almost any model fit using one of **rstanarm**'s model-fitting functions can be used with ShinyStan. The only exception is that ShinyStan does not currently support **rstanarm** models fit using algorithm='optimizing'. See the **shinystan** package documentation for more information.

Faster launch times

For some **rstanarm** models ShinyStan may take a very long time to launch. If this is the case with one of your models you may be able to speed up launch_shinystan in one of several ways:

Prevent ShinyStan from preparing graphical posterior predictive checks: When used with a stanreg object (**rstanarm** model object) ShinyStan will draw from the posterior predictive distribution and prepare graphical posterior predictive checks before launching. That way when you go to the PPcheck page the plots are immediately available. This can be time consuming for models fit to very large datasets and you can prevent this behavior by creating a shinystan object before calling launch_shinystan. To do this use as.shinystan with optional argument ppd set to FALSE (see the Examples section below). When you then launch ShinyStan and go to the PPcheck page the plots will no longer be automatically generated and you will be presented with the standard interface requiring you to first specify the appropriate y and $yrep$, which can be done for many but not all **rstanarm** models.

Use a shinystan object: Even if you don't want to prevent ShinyStan from preparing graphical posterior predictive checks, first creating a shinystan object using as.shinystan can reduce *future* launch times. That is, launch_shinystan(sso) will be faster than launch_shinystan(fit), where sso is a shinystan object and fit is a stanreg object. It still may take some time for as.shinystan to create sso initially, but each time you subsequently call launch_shinystan(sso) it will reuse sso instead of internally creating a shinystan object every time. See the Examples section below.

Examples

```
## Not run:
if (!exists("example_model")) example(example_model)

# Launch the ShinyStan app without saving the resulting shinystan object
if (interactive()) launch_shinystan(example_model)
```

```
# Launch the ShinyStan app (saving resulting shinystan object as sso)
if (interactive()) sso <- launch_shinystan(example_model)

# First create shinystan object then call launch_shinystan
sso <- shinystan::as.shinystan(example_model)
if (interactive()) launch_shinystan(sso)

# Prevent ShinyStan from preparing graphical posterior predictive checks that
# can be time consuming. example_model is small enough that it won't matter
# much here but in general this can help speed up launch_shinystan
sso <- shinystan::as.shinystan(example_model, ppd = FALSE)
if (interactive()) launch_shinystan(sso)

## End(Not run)
```

stanreg-methods *Methods for stanreg objects*

Description

The methods documented on this page are actually some of the least important methods defined for stanreg objects. The most important methods are documented separately, each with its own page. Links to those pages are provided in the **See Also** section, below.

Usage

```
## S3 method for class 'stanreg'
coef(object, ...)

## S3 method for class 'stanreg'
confint(object, parm, level = 0.95, ...)

## S3 method for class 'stanreg'
fitted(object, ...)

## S3 method for class 'stanreg'
nobs(object, ...)

## S3 method for class 'stanreg'
residuals(object, ...)

## S3 method for class 'stanreg'
se(object, ...)

## S3 method for class 'stanreg'
```

```
update(object, formula., ..., evaluate = TRUE)

## S3 method for class 'stanreg'
vcov(object, correlation = FALSE, ...)

## S3 method for class 'stanreg'
fixef(object, ...)

## S3 method for class 'stanreg'
ngrps(object, ...)

## S3 method for class 'stanreg'
ranef(object, ...)

## S3 method for class 'stanreg'
sigma(object, ...)

## S3 method for class 'stanreg'
VarCorr(x, sigma = 1, ...)
```

Arguments

`object, x`	A fitted model object returned by one of the **rstanarm** modeling functions. See `stanreg-objects`.
`...`	Ignored, except by the `update` method. See `update`.
`parm`	For `confint`, an optional character vector of parameter names.
`level`	For `confint`, a scalar between 0 and 1 indicating the confidence level to use.
`formula., evaluate`	See `update`.
`correlation`	For `vcov`, if `FALSE` (the default) the covariance matrix is returned. If `TRUE`, the correlation matrix is returned instead.
`sigma`	Ignored (included for compatibility with `VarCorr`).

Details

The methods documented on this page are similar to the methods defined for objects of class 'lm', 'glm', 'glmer', etc. However there are a few key differences:

`residuals` Residuals are *always* of type `"response"` (not `"deviance"` residuals or any other type). However, in the case of `stan_polr` with more than two response categories, the residuals are the difference between the latent utility and its linear predictor.

`coef` Medians are used for point estimates. See the *Point estimates* section in `print.stanreg` for more details.

`se` The `se` function returns standard errors based on `mad`. See the *Uncertainty estimates* section in `print.stanreg` for more details.

`confint` For models fit using optimization, confidence intervals are returned via a call to `confint.default`. If `algorithm` is `"sampling"`, `"meanfield"`, or `"fullrank"`, the `confint` will throw an error because the `posterior_interval` function should be used to compute Bayesian uncertainty intervals.

See Also

- The `print`, `summary`, and `prior_summary` methods for stanreg objects for information on the fitted model.
- `launch_shinystan` to use the ShinyStan GUI to explore a fitted **rstanarm** model.
- The `plot` method to plot estimates and diagnostics.
- The `pp_check` method for graphical posterior predictive checking.
- The `posterior_predict` and `predictive_error` methods for predictions and predictive errors.
- The `posterior_interval` and `predictive_interval` methods for uncertainty intervals for model parameters and predictions.
- The `loo`, `kfold`, and `log_lik` methods for leave-one-out or K-fold cross-validation, model comparison, and computing the log-likelihood of (possibly new) data.
- The `as.matrix`, `as.data.frame`, and `as.array` methods to access posterior draws.

`stanreg-objects` *Fitted model objects*

Description

The **rstanarm** model-fitting functions return an object of class 'stanreg', which is a list containing at a minimum the components listed below. Each stanreg object will also have additional classes (e.g. 'aov', 'betareg', 'glm', 'polr', etc.) and several additional components depending on the model and estimation algorithm.

stanreg objects

`coefficients` Point estimates, as described in `print.stanreg`.

`ses` Standard errors based on `mad`, as described in `print.stanreg`.

`residuals` Residuals of type `'response'`.

`fitted.values` Fitted mean values. For GLMs the linear predictors are transformed by the inverse link function.

`linear.predictors` Linear fit on the link scale. For linear models this is the same as `fitted.values`.

`covmat` Variance-covariance matrix for the coefficients based on draws from the posterior distribution, the variational approximation, or the asymptotic sampling distribution, depending on the estimation algorithm.

`model,x,y` If requested, the the model frame, model matrix and response variable used, respectively.

`family` The family object used.

`call` The matched call.

`formula` The model formula.

`data,offset,weights` The `data`, `offset`, and `weights` arguments.

`algorithm` The estimation method used.

`prior.info` A list with information about the prior distributions used.

`stanfit,stan_summary` The object of `stanfit-class` returned by RStan and a matrix of various summary statistics from the stanfit object.

`rstan_version` The version of the **rstan** package that was used to fit the model.

Note

The `stan_biglm` function is an exception. It returns a stanfit object rather than a stanreg object.

See Also

`stanreg-methods`

`stan_aov` *Bayesian regularized linear models via Stan*

Description

Bayesian inference for linear modeling with regularizing priors on the model parameters that are driven by prior beliefs about R^2, the proportion of variance in the outcome attributable to the predictors. See `priors` for an explanation of this critical point. `stan_glm` with `family="gaussian"` also estimates a linear model with normally-distributed errors and allows for various other priors on the coefficients.

Usage

```
stan_aov(formula, data, projections = FALSE, contrasts = NULL, ...,
  prior = R2(stop("'location' must be specified")), prior_PD = FALSE,
  algorithm = c("sampling", "meanfield", "fullrank"), adapt_delta = NULL)

stan_lm(formula, data, subset, weights, na.action, model = TRUE, x = FALSE,
  y = FALSE, singular.ok = TRUE, contrasts = NULL, offset, ...,
  prior = R2(stop("'location' must be specified")), prior_intercept = NULL,
  prior_PD = FALSE, algorithm = c("sampling", "meanfield", "fullrank"),
  adapt_delta = NULL)

stan_lm.wfit(x, y, w, offset = NULL, singular.ok = TRUE, ...,
  prior = R2(stop("'location' must be specified")), prior_intercept = NULL,
  prior_PD = FALSE, algorithm = c("sampling", "meanfield", "fullrank"),
  adapt_delta = NULL)
```

```
stan_lm.fit(x, y, offset = NULL, singular.ok = TRUE, ...,
  prior = R2(stop("'location' must be specified")), prior_intercept = NULL,
  prior_PD = FALSE, algorithm = c("sampling", "meanfield", "fullrank"),
  adapt_delta = NULL)
```

Arguments

`formula, data, subset`	Same as `lm`, but *we strongly advise against omitting the* `data` *argument*. Unless `data` is specified (and is a data frame) many post-estimation functions (including `update`, `loo`, `kfold`) are not guaranteed to work properly.
`projections`	For `stan_aov`, a logical scalar (defaulting to `FALSE`) indicating whether `proj` should be called on the fit.
`...`	Further arguments passed to the function in the **rstan** package (`sampling`, `vb`, or `optimizing`), corresponding to the estimation method named by `algorithm`. For example, if `algorithm` is `"sampling"` it is possibly to specify `iter`, `chains`, `cores`, `refresh`, etc.
`prior`	Must be a call to `R2` with its `location` argument specified or `NULL`, which would indicate a standard uniform prior for the R^2.
`prior_PD`	A logical scalar (defaulting to `FALSE`) indicating whether to draw from the prior predictive distribution instead of conditioning on the outcome.
`algorithm`	A string (possibly abbreviated) indicating the estimation approach to use. Can be `"sampling"` for MCMC (the default), `"optimizing"` for optimization, `"meanfield"` for variational inference with independent normal distributions, or `"fullrank"` for variational inference with a multivariate normal distribution. See `rstanarm-package` for more details on the estimation algorithms. NOTE: not all fitting functions support all four algorithms.
`adapt_delta`	Only relevant if `algorithm="sampling"`. See `adapt_delta` for details.
`na.action, singular.ok, contrasts`	Same as `lm`, but rarely specified.
`model, offset, weights`	Same as `lm`, but rarely specified.
`x, y`	In `stan_lm, stan_aov`, logical scalars indicating whether to return the design matrix and response vector. In `stan_lm.fit or stan_lm.wfit`, a design matrix and response vector.
`prior_intercept`	Either `NULL` (the default) or a call to `normal`. If a `normal` prior is specified without a `scale`, then the standard deviation is taken to be the marginal standard deviation of the outcome divided by the square root of the sample size, which is legitimate because the marginal standard deviation of the outcome is a primitive parameter being estimated.
`w`	Same as in `lm.wfit` but rarely specified.

Details

The `stan_lm` function is similar in syntax to the `lm` function but rather than choosing the parameters to minimize the sum of squared residuals, samples from the posterior distribution are drawn using MCMC (if `algorithm` is `"sampling"`). The `stan_lm` function has a formula-based interface and would usually be called by users but the `stan_lm.fit` and `stan_lm.wfit` functions might be called by other functions that parse the data themselves and are analagous to `lm.fit` and `lm.wfit` respectively.

In addition to estimating `sigma` — the standard deviation of the normally-distributed errors — this model estimates a positive parameter called `log-fit_ratio`. If it is positive, the marginal posterior variance of the outcome will exceed the sample variance of the outcome by a multiplicative factor equal to the square of `fit_ratio`. Conversely if `log-fit_ratio` is negative, then the model underfits. Given the regularizing nature of the priors, a slight underfit is good.

Finally, the posterior predictive distribution is generated with the predictors fixed at their sample means. This quantity is useful for checking convergence because it is reasonably normally distributed and a function of all the parameters in the model.

The `stan_aov` function is similar to `aov` and has a somewhat customized `print` method but basically just calls `stan_lm` with dummy variables to do a Bayesian analysis of variance.

Value

A stanreg object is returned for `stan_lm, stan_aov`.

A stanfit object (or a slightly modified stanfit object) is returned if `stan_lm.fit or stan_lm.wfit` is called directly.

References

Lewandowski, D., Kurowicka D., and Joe, H. (2009). Generating random correlation matrices based on vines and extended onion method. *Journal of Multivariate Analysis*. **100**(9), 1989–2001.

See Also

The vignettes for `stan_lm` and `stan_aov`, which have more thorough descriptions and examples.

Also see `stan_glm`, which — if `family = gaussian(link="identity")` — also estimates a linear model with normally-distributed errors but specifies different priors.

Examples

```
op <- options(contrasts = c("contr.helmert", "contr.poly"))
stan_aov(yield ~ block + N*P*K, data = npk,
         prior = R2(0.5), seed = 12345)
options(op)


(fit <- stan_lm(mpg ~ wt + qsec + am, data = mtcars, prior = R2(0.75),
                # the next line is only to make the example go fast enough
                chains = 1, iter = 500, seed = 12345))
plot(fit, prob = 0.8)
plot(fit, "hist", pars = c("wt", "am", "qsec", "sigma"),
     transformations = list(sigma = "log"))
```

stan_betareg *Bayesian beta regression models via Stan*

Description

Beta regression modeling with optional prior distributions for the coefficients, intercept, and auxiliary parameter phi (if applicable).

Usage

```
stan_betareg(formula, data, subset, na.action, weights, offset,
  link = c("logit", "probit", "cloglog", "cauchit", "log", "loglog"),
  link.phi = NULL, model = TRUE, y = TRUE, x = FALSE, ...,
  prior = normal(), prior_intercept = normal(), prior_z = normal(),
  prior_intercept_z = normal(), prior_phi = cauchy(0, 5),
  prior_PD = FALSE, algorithm = c("sampling", "optimizing", "meanfield",
  "fullrank"), adapt_delta = NULL, QR = FALSE)

stan_betareg.fit(x, y, z = NULL, weights = rep(1, NROW(x)),
  offset = rep(0, NROW(x)), link = c("logit", "probit", "cloglog",
  "cauchit", "log", "loglog"), link.phi = NULL, ..., prior = normal(),
  prior_intercept = normal(), prior_z = normal(),
  prior_intercept_z = normal(), prior_phi = cauchy(0, 5),
  prior_PD = FALSE, algorithm = c("sampling", "optimizing", "meanfield",
  "fullrank"), adapt_delta = NULL, QR = FALSE)
```

Arguments

`formula, data, subset`	Same as betareg, but *we strongly advise against omitting the* `data` *argument*. Unless `data` is specified (and is a data frame) many post-estimation functions (including `update`, `loo`, `kfold`) are not guaranteed to work properly.
`na.action`	Same as betareg, but rarely specified.
`link`	Character specification of the link function used in the model for mu (specified through `x`). Currently, "logit", "probit", "cloglog", "cauchit", "log", and "loglog" are supported.
`link.phi`	If applicable, character specification of the link function used in the model for `phi` (specified through `z`). Currently, "identity", "log" (default), and "sqrt" are supported. Since the "sqrt" link function is known to be unstable, it is advisable to specify a different link function (or to model `phi` as a scalar parameter instead of via a linear predictor by excluding `z` from the `formula` and excluding `link.phi`).

`model, offset, weights`
: Same as `betareg`.

`x, y`
: In `stan_betareg`, logical scalars indicating whether to return the design matrix and response vector. In `stan_betareg.fit`, a design matrix and response vector.

`...`
: Further arguments passed to the function in the **rstan** package (`sampling`, `vb`, or `optimizing`), corresponding to the estimation method named by `algorithm`. For example, if `algorithm` is `"sampling"` it is possibly to specify `iter`, `chains`, `cores`, `refresh`, etc.

`prior`
: The prior distribution for the regression coefficients. `prior` should be a call to one of the various functions provided by **rstanarm** for specifying priors. The subset of these functions that can be used for the prior on the coefficients can be grouped into several "families":

Family	**Functions**
Student t family	`normal`, `student_t`, `cauchy`
Hierarchical shrinkage family	`hs`, `hs_plus`
Laplace family	`laplace`, `lasso`
Product normal family	`product_normal`

See the priors help page for details on the families and how to specify the arguments for all of the functions in the table above. To omit a prior —i.e., to use a flat (improper) uniform prior— `prior` can be set to `NULL`, although this is rarely a good idea.

Note: Unless `QR=TRUE`, if `prior` is from the Student t family or Laplace family, and if the `autoscale` argument to the function used to specify the prior (e.g. `normal`) is left at its default and recommended value of `TRUE`, then the default or user-specified prior scale(s) may be adjusted internally based on the scales of the predictors. See the priors help page for details on the rescaling and the `prior_summary` function for a summary of the priors used for a particular model.

`prior_intercept`
: The prior distribution for the intercept. `prior_intercept` can be a call to `normal`, `student_t` or `cauchy`. See the priors help page for details on these functions. To omit a prior on the intercept —i.e., to use a flat (improper) uniform prior— `prior_intercept` can be set to `NULL`.

Note: If using a dense representation of the design matrix —i.e., if the `sparse` argument is left at its default value of `FALSE`— then the prior distribution for the intercept is set so it applies to the value when all predictors are centered. If you prefer to specify a prior on the intercept without the predictors being auto-centered, then you have to omit the intercept from the `formula` and include a column of ones as a predictor, in which case some element of `prior` specifies the prior on it, rather than `prior_intercept`.

`prior_z`
: Prior distribution for the coefficients in the model for `phi` (if applicable). Same options as for `prior`.

`prior_intercept_z`	Prior distribution for the intercept in the model for `phi` (if applicable). Same options as for `prior_intercept`.
`prior_phi`	The prior distribution for `phi` if it is *not* modeled as a function of predictors. If `z` variables are specified then `prior_phi` is ignored and `prior_intercept_z` and `prior_z` are used to specify the priors on the intercept and coefficients in the model for `phi`. When applicable, `prior_phi` can be a call to `exponential` to use an exponential distribution, or one of `normal`, `student_t` or `cauchy` to use half-normal, half-t, or half-Cauchy prior. See `priors` for details on these functions. To omit a prior —i.e., to use a flat (improper) uniform prior— set `prior_phi` to `NULL`.
`prior_PD`	A logical scalar (defaulting to `FALSE`) indicating whether to draw from the prior predictive distribution instead of conditioning on the outcome.
`algorithm`	A string (possibly abbreviated) indicating the estimation approach to use. Can be `"sampling"` for MCMC (the default), `"optimizing"` for optimization, `"meanfield"` for variational inference with independent normal distributions, or `"fullrank"` for variational inference with a multivariate normal distribution. See `rstanarm-package` for more details on the estimation algorithms. NOTE: not all fitting functions support all four algorithms.
`adapt_delta`	Only relevant if `algorithm="sampling"`. See `adapt_delta` for details.
`QR`	A logical scalar defaulting to `FALSE`, but if `TRUE` applies a scaled `qr` decomposition to the design matrix, $X = Q^* R^*$, where $Q^* = Q \sqrt{n-1}$ and $R^* = \frac{1}{\sqrt{n-1}} R$. The coefficients relative to Q^* are obtained and then premultiplied by the inverse of R^* to obtain coefficients relative to the original predictors, X. These transformations do not change the likelihood of the data but are recommended for computational reasons when there are multiple predictors. Importantly, while the columns of X are almost always correlated, the columns of Q^* are uncorrelated by design, which often makes sampling from the posterior easier. However, because when `QR` is `TRUE` the `prior` argument applies to the coefficients relative to Q^* (and those are not very interpretable), setting `QR=TRUE` is only recommended if you do not have an informative prior for the regression coefficients.
`z`	For `stan_betareg.fit`, a regressor matrix for `phi`. Defaults to an intercept only.

Details

The `stan_betareg` function is similar in syntax to `betareg` but rather than performing maximum likelihood estimation, full Bayesian estimation is performed (if `algorithm` is `"sampling"`) via MCMC. The Bayesian model adds priors (independent by default) on the coefficients of the beta regression model. The `stan_betareg` function calls the workhorse `stan_betareg.fit` function, but it is also possible to call the latter directly.

Value

A stanreg object is returned for `stan_betareg`.

A stanfit object (or a slightly modified stanfit object) is returned if `stan_betareg.fit` is called directly.

References

Ferrari, SLP and Cribari-Neto, F (2004). Beta regression for modeling rates and proportions. *Journal of Applied Statistics*. 31(7), 799–815.

See Also

stanreg-methods and betareg.

The vignette for stan_betareg.

Examples

```
### Simulated data
N <- 200
x <- rnorm(N, 2, 1)
z <- rnorm(N, 2, 1)
mu <- binomial(link = "logit")$linkinv(1 + 0.2*x)
phi <- exp(1.5 + 0.4*z)
y <- rbeta(N, mu * phi, (1 - mu) * phi)
hist(y, col = "dark grey", border = FALSE, xlim = c(0,1))
fake_dat <- data.frame(y, x, z)

fit <- stan_betareg(y ~ x | z, data = fake_dat,
                    link = "logit", link.phi = "log",
                    algorithm = "optimizing")
print(fit, digits = 2)
```

stan_biglm *Bayesian regularized linear but big models via Stan*

Description

This is the same model as with stan_lm but it utilizes the output from biglm in the **biglm** package in order to proceed when the data is too large to fit in memory.

Usage

```
stan_biglm(biglm, xbar, ybar, s_y, ...,
  prior = R2(stop("'location' must be specified")), prior_intercept = NULL,
  prior_PD = FALSE, algorithm = c("sampling", "meanfield", "fullrank"),
  adapt_delta = NULL)

stan_biglm.fit(b, R, SSR, N, xbar, ybar, s_y, has_intercept = TRUE, ...,
  prior = R2(stop("'location' must be specified")), prior_intercept = NULL,
  prior_PD = FALSE, algorithm = c("sampling", "meanfield", "fullrank"),
  adapt_delta = NULL)
```

Arguments

`biglm`	The list output by `biglm` in the **biglm** package.
`xbar`	A numeric vector of column means in the implicit design matrix excluding the intercept for the observations included in the model.
`ybar`	A numeric scalar indicating the mean of the outcome for the observations included in the model.
`s_y`	A numeric scalar indicating the unbiased sample standard deviation of the outcome for the observations included in the model.
`...`	Further arguments passed to the function in the **rstan** package (`sampling`, `vb`, or `optimizing`), corresponding to the estimation method named by `algorithm`. For example, if `algorithm` is `"sampling"` it is possibly to specify `iter`, `chains`, `cores`, `refresh`, etc.
`prior`	Must be a call to `R2` with its `location` argument specified or `NULL`, which would indicate a standard uniform prior for the R^2.
`prior_intercept`	Either `NULL` (the default) or a call to `normal`. If a `normal` prior is specified without a `scale`, then the standard deviation is taken to be the marginal standard deviation of the outcome divided by the square root of the sample size, which is legitimate because the marginal standard deviation of the outcome is a primitive parameter being estimated.
`prior_PD`	A logical scalar (defaulting to `FALSE`) indicating whether to draw from the prior predictive distribution instead of conditioning on the outcome.
`algorithm`	A string (possibly abbreviated) indicating the estimation approach to use. Can be `"sampling"` for MCMC (the default), `"optimizing"` for optimization, `"meanfield"` for variational inference with independent normal distributions, or `"fullrank"` for variational inference with a multivariate normal distribution. See `rstanarm-package` for more details on the estimation algorithms. NOTE: not all fitting functions support all four algorithms.
`adapt_delta`	Only relevant if `algorithm="sampling"`. See `adapt_delta` for details.
`b`	A numeric vector of OLS coefficients, excluding the intercept
`R`	A square upper-triangular matrix from the QR decomposition of the design matrix, excluding the intercept
`SSR`	A numeric scalar indicating the sum-of-squared residuals for OLS
`N`	A integer scalar indicating the number of included observations
`has_intercept`	A logical scalar indicating whether to add an intercept to the model when estimating it.

Details

The `stan_biglm` function is intended to be used in the same circumstances as the `biglm` function in the **biglm** package but with an informative prior on the R^2 of the regression. Like `biglm`, the memory required to estimate the model depends largely on the number of predictors rather than the number of observations. However, `stan_biglm` and `stan_biglm.fit` have additional required arguments that are not necessary in `biglm`, namely `xbar`, `ybar`, and `s_y`. If any observations have any missing values on any of the predictors or the outcome, such observations do not contribute to these statistics.

Value

The output of both `stan_biglm` and `stan_biglm.fit` is an object of `stanfit-class` rather than `stanreg-objects`, which is more limited and less convenient but necessitated by the fact that `stan_biglm` does not bring the full design matrix into memory. Without the full design matrix,some of the elements of a `stanreg-objects` object cannot be calculated, such as residuals. Thus, the functions in the **rstanarm** package that input `stanreg-objects`, such as `posterior_predict` cannot be used.

Examples

```
# create inputs
ols <- lm(mpg ~ wt + qsec + am, data = mtcars, # all row are complete so ...
          na.action = na.exclude)              # not necessary in this case
b <- coef(ols)[-1]
R <- qr.R(ols$qr)[-1,-1]
SSR <- crossprod(ols$residuals)[1]
not_NA <- !is.na(fitted(ols))
N <- sum(not_NA)
xbar <- colMeans(mtcars[not_NA,c("wt", "qsec", "am")])
y <- mtcars$mpg[not_NA]
ybar <- mean(y)
s_y <- sd(y)
post <- stan_biglm.fit(b, R, SSR, N, xbar, ybar, s_y, prior = R2(.75),
                       # the next line is only to make the example go fast
                       chains = 1, iter = 500, seed = 12345)
cbind(lm = b, stan_lm = rstan::get_posterior_mean(post)[13:15,]) # shrunk
```

`stan_gamm4` *Bayesian generalized linear additive models with group-specific terms via Stan*

Description

Bayesian inference for GAMMs with flexible priors.

Usage

```
stan_gamm4(formula, random = NULL, family = gaussian(), data,
  weights = NULL, subset = NULL, na.action, knots = NULL,
  drop.unused.levels = TRUE, ..., prior = normal(),
  prior_intercept = normal(), prior_aux = cauchy(0, 5),
  prior_covariance = decov(), prior_PD = FALSE, algorithm = c("sampling",
  "meanfield", "fullrank"), adapt_delta = NULL, QR = FALSE,
  sparse = FALSE)

plot_nonlinear(x, smooths, ..., prob = 0.9, facet_args = list(),
  alpha = 1, size = 0.75)
```

Arguments

`formula, random, family, data, knots, drop.unused.levels`
: Same as for `gamm4`. *We strongly advise against omitting the* `data` *argument*. Unless `data` is specified (and is a data frame) many post-estimation functions (including `update`, `loo`, `kfold`) are not guaranteed to work properly.

`subset, weights, na.action`
: Same as `glm`, but rarely specified.

`...`
: Further arguments passed to `sampling` (e.g. `iter`, `chains`, `cores`, etc.) or to `vb` (if `algorithm` is `"meanfield"` or `"fullrank"`).

`prior`
: The prior distribution for the regression coefficients. `prior` should be a call to one of the various functions provided by **rstanarm** for specifying priors. The subset of these functions that can be used for the prior on the coefficients can be grouped into several "families":

Family	**Functions**
Student t family	`normal`, `student_t`, `cauchy`
Hierarchical shrinkage family	`hs`, `hs_plus`
Laplace family	`laplace`, `lasso`
Product normal family	`product_normal`

See the priors help page for details on the families and how to specify the arguments for all of the functions in the table above. To omit a prior —i.e., to use a flat (improper) uniform prior— `prior` can be set to `NULL`, although this is rarely a good idea.

Note: Unless `QR=TRUE`, if `prior` is from the Student t family or Laplace family, and if the `autoscale` argument to the function used to specify the prior (e.g. `normal`) is left at its default and recommended value of `TRUE`, then the default or user-specified prior scale(s) may be adjusted internally based on the scales of the predictors. See the priors help page for details on the rescaling and the `prior_summary` function for a summary of the priors used for a particular model.

`prior_intercept`
: The prior distribution for the intercept. `prior_intercept` can be a call to `normal`, `student_t` or `cauchy`. See the priors help page for details on these functions. To omit a prior on the intercept —i.e., to use a flat (improper) uniform prior— `prior_intercept` can be set to `NULL`.

Note: If using a dense representation of the design matrix —i.e., if the `sparse` argument is left at its default value of `FALSE`— then the prior distribution for the intercept is set so it applies to the value when all predictors are centered. If you prefer to specify a prior on the intercept without the predictors being auto-centered, then you have to omit the intercept from the `formula` and include a column of ones as a predictor, in which case some element of `prior` specifies the prior on it, rather than `prior_intercept`.

`prior_aux`
: The prior distribution for the "auxiliary" parameter (if applicable). The "auxiliary" parameter refers to a different parameter depending on the `family`. For Gaussian models `prior_aux` controls `"sigma"`, the error standard deviation.

For negative binomial models `prior_aux` controls `"reciprocal_dispersion"`, which is similar to the `"size"` parameter of `rnbinom`: smaller values of `"reciprocal_dispersion"` correspond to greater dispersion. For gamma models `prior_aux` sets the prior on to the `"shape"` parameter (see e.g., `rgamma`), and for inverse-Gaussian models it is the so-called `"lambda"` parameter (which is essentially the reciprocal of a scale parameter). Binomial and Poisson models do not have auxiliary parameters.

`prior_aux` can be a call to `exponential` to use an exponential distribution, or `normal`, `student_t` or `cauchy`, which results in a half-normal, half-t, or half-Cauchy prior. See `priors` for details on these functions. To omit a prior —i.e., to use a flat (improper) uniform prior— set `prior_aux` to `NULL`.

`prior_covariance`
: Cannot be `NULL`; see `decov` for more information about the default arguments.

`prior_PD`
: A logical scalar (defaulting to `FALSE`) indicating whether to draw from the prior predictive distribution instead of conditioning on the outcome.

`algorithm`
: A string (possibly abbreviated) indicating the estimation approach to use. Can be `"sampling"` for MCMC (the default), `"optimizing"` for optimization, `"meanfield"` for variational inference with independent normal distributions, or `"fullrank"` for variational inference with a multivariate normal distribution. See `rstanarm-package` for more details on the estimation algorithms. NOTE: not all fitting functions support all four algorithms.

`adapt_delta`
: Only relevant if `algorithm="sampling"`. See `adapt_delta` for details.

`QR`
: A logical scalar defaulting to `FALSE`, but if `TRUE` applies a scaled `qr` decomposition to the design matrix, $X = Q^* R^*$, where $Q^* = Q\sqrt{n-1}$ and $R^* = \frac{1}{\sqrt{n-1}}R$. The coefficients relative to Q^* are obtained and then premultiplied by the inverse of R^* to obtain coefficients relative to the original predictors, X. These transformations do not change the likelihood of the data but are recommended for computational reasons when there are multiple predictors. Importantly, while the columns of X are almost always correlated, the columns of Q^* are uncorrelated by design, which often makes sampling from the posterior easier. However, because when `QR` is `TRUE` the `prior` argument applies to the coefficients relative to Q^* (and those are not very interpretable), setting `QR=TRUE` is only recommended if you do not have an informative prior for the regression coefficients.

`sparse`
: A logical scalar (defaulting to `FALSE`) indicating whether to use a sparse representation of the design (X) matrix. Setting this to `TRUE` will likely be twice as slow, even if the design matrix has a considerable number of zeros, but it may allow the model to be estimated when the computer has too little RAM to utilize a dense design matrix. If `TRUE`, the the design matrix is not centered (since that would destroy the sparsity) and it is not possible to specify both `QR = TRUE` and `sparse = TRUE`.

`x`
: An object produced by `stan_gamm4`.

`smooths`
: An optional character vector specifying a subset of the smooth functions specified in the call to `stan_gamm4`. The default is include all smooth terms.

`prob`
: For univarite smooths, a scalar between 0 and 1 governing the width of the uncertainty interval.

`facet_args`	An optional named list of arguments passed to `facet_wrap` (other than the `facets` argument).
`alpha, size`	For univariate smooths, passed to `geom_ribbon`. For bivariate smooths, `size/2` is passed to `geom_contour`.

Details

The `stan_gamm4` function is similar in syntax to `gamm4` in the **gamm4** package, which accepts a syntax that is similar to (but not quite as extensive as) that for `gamm` in the **mgcv** package and converts it internally into the syntax accepted by `glmer` in the **lme4** package. But rather than performing (restricted) maximum likelihood estimation, the `stan_gamm4` function utilizes MCMC to perform Bayesian estimation. The Bayesian model adds priors on the common regression coefficients (in the same way as `stan_glm`) and priors on the terms of a decomposition of the covariance matrices of the group-specific parameters, including the smooths. Estimating these models via MCMC avoids the optimization issues that often crop up with GAMMs and provides better estimates for the uncertainty in the parameter estimates.

See `gamm4` for more information about the model specicification and `priors` for more information about the priors. If `random = NULL`, the output is a subset of that produced by `gam` in the sense that there are several estimated components for each smooth term. However, the parameterization used to estimate the model is different and corresponds to the parameterization in `gamm4` where is smooth term is decomposed into a linear and a non-linear part. If `prior` is not `NULL`, then the number of parameters to place priors on is equal to the number of linear terms in the `formula`. The prior on the non-linear part of each smooth term is handled by the `decov` function. If `random` is not `NULL`, then there are additional group-specific terms whose priors are also handled by the `decov` function and whose posterior medians can be extracted by calling `ranef`.

The `plot_nonlinear` function creates a ggplot object with one facet for each smooth function specified in the call to `stan_gamm4` in the case where all smooths are univariate. A subset of the smooth functions can be specified using the `smooths` argument, which is necessary to plot a bivariate smooth or to exclude the bivariate smooth and plot the univariate ones. In the bivariate case, a plot is produced using `geom_contour`. In the univariate case, the resulting plot is conceptually similar to `plot.gam` except the outer lines here demark the edges of posterior uncertainty intervals (credible intervals) rather than confidence intervals and the inner line is the posterior median of the function rather than the function implied by a point estimate. To change the colors used in the plot see `color_scheme_set`.

Value

A stanreg object is returned for `stan_gamm4`.

`plot_nonlinear` returns a ggplot object.

References

Crainiceanu, C., Ruppert D., and Wand, M. (2005). Bayesian analysis for penalized spline regression using WinBUGS. *Journal of Statistical Software*. **14**(14), 1–22. `https://www.jstatsoft.org/article/view/v014i14`

See Also

stanreg-methods and gamm4.

Examples

```
# from example(gamm4, package = "gamm4"), prefixing gamm4() call with stan_

dat <- mgcv::gamSim(1, n = 400, scale = 2) ## simulate 4 term additive truth
## Now add 20 level random effect `fac'...
dat$fac <- fac <- as.factor(sample(1:20, 400, replace = TRUE))
dat$y <- dat$y + model.matrix(~ fac - 1) %*% rnorm(20) * .5

br <- stan_gamm4(y ~ s(x0) + x1 + s(x2), data = dat, random = ~ (1 | fac),
                 QR = TRUE, chains = 1, iter = 200) # for example speed
print(br)
plot_nonlinear(br)
plot_nonlinear(br, smooths = "s(x0)", alpha = 2/3)
```

---

stan_glm *Bayesian generalized linear models via Stan*

Description

Generalized linear modeling with optional prior distributions for the coefficients, intercept, and auxiliary parameters.

Usage

```
stan_glm(formula, family = gaussian(), data, weights, subset,
  na.action = NULL, offset = NULL, model = TRUE, x = FALSE, y = TRUE,
  contrasts = NULL, ..., prior = normal(), prior_intercept = normal(),
  prior_aux = cauchy(0, 5), prior_PD = FALSE, algorithm = c("sampling",
  "optimizing", "meanfield", "fullrank"), adapt_delta = NULL, QR = FALSE,
  sparse = FALSE)

stan_glm.nb(formula, data, weights, subset, na.action = NULL, offset = NULL,
  model = TRUE, x = FALSE, y = TRUE, contrasts = NULL, link = "log",
  ..., prior = normal(), prior_intercept = normal(), prior_aux = cauchy(0,
  5), prior_PD = FALSE, algorithm = c("sampling", "optimizing", "meanfield",
  "fullrank"), adapt_delta = NULL, QR = FALSE)

stan_glm.fit(x, y, weights = rep(1, NROW(x)), offset = rep(0, NROW(x)),
  family = gaussian(), ..., prior = normal(), prior_intercept = normal(),
  prior_aux = cauchy(0, 5), prior_ops = NULL, group = list(),
  prior_PD = FALSE, algorithm = c("sampling", "optimizing", "meanfield",
  "fullrank"), adapt_delta = NULL, QR = FALSE, sparse = FALSE)
```

Arguments

`formula, data, subset`
: Same as `glm`, but *we strongly advise against omitting the* `data` *argument*. Unless `data` is specified (and is a data frame) many post-estimation functions (including `update`, `loo`, `kfold`) are not guaranteed to work properly.

`family`
: Same as `glm`, except negative binomial GLMs are also possible using the `neg_binomial_2` family object.

`na.action, contrasts`
: Same as `glm`, but rarely specified.

`model, offset, weights`
: Same as `glm`.

`x, y`
: In `stan_glm, stan_glm.nb`, logical scalars indicating whether to return the design matrix and response vector. In `stan_glm.fit`, a design matrix and response vector.

`...`
: Further arguments passed to the function in the **rstan** package (`sampling`, `vb`, or `optimizing`), corresponding to the estimation method named by `algorithm`. For example, if `algorithm` is `"sampling"` it is possibly to specify `iter`, `chains`, `cores`, `refresh`, etc.

`prior`
: The prior distribution for the regression coefficients. `prior` should be a call to one of the various functions provided by **rstanarm** for specifying priors. The subset of these functions that can be used for the prior on the coefficients can be grouped into several "families":

Family	**Functions**
Student t family	`normal`, `student_t`, `cauchy`
Hierarchical shrinkage family	`hs`, `hs_plus`
Laplace family	`laplace`, `lasso`
Product normal family	`product_normal`

See the priors help page for details on the families and how to specify the arguments for all of the functions in the table above. To omit a prior —i.e., to use a flat (improper) uniform prior— `prior` can be set to `NULL`, although this is rarely a good idea.

Note: Unless `QR=TRUE`, if `prior` is from the Student t family or Laplace family, and if the `autoscale` argument to the function used to specify the prior (e.g. `normal`) is left at its default and recommended value of `TRUE`, then the default or user-specified prior scale(s) may be adjusted internally based on the scales of the predictors. See the priors help page for details on the rescaling and the `prior_summary` function for a summary of the priors used for a particular model.

`prior_intercept`
: The prior distribution for the intercept. `prior_intercept` can be a call to `normal`, `student_t` or `cauchy`. See the priors help page for details on these functions. To omit a prior on the intercept —i.e., to use a flat (improper) uniform prior— `prior_intercept` can be set to `NULL`.

	Note: If using a dense representation of the design matrix —i.e., if the `sparse` argument is left at its default value of `FALSE`— then the prior distribution for the intercept is set so it applies to the value when all predictors are centered. If you prefer to specify a prior on the intercept without the predictors being auto-centered, then you have to omit the intercept from the `formula` and include a column of ones as a predictor, in which case some element of `prior` specifies the prior on it, rather than `prior_intercept`.
`prior_aux`	The prior distribution for the "auxiliary" parameter (if applicable). The "auxiliary" parameter refers to a different parameter depending on the `family`. For Gaussian models `prior_aux` controls `"sigma"`, the error standard deviation. For negative binomial models `prior_aux` controls `"reciprocal_dispersion"`, which is similar to the `"size"` parameter of `rnbinom`: smaller values of `"reciprocal_dispersion"` correspond to greater dispersion. For gamma models `prior_aux` sets the prior on to the `"shape"` parameter (see e.g., `rgamma`), and for inverse-Gaussian models it is the so-called `"lambda"` parameter (which is essentially the reciprocal of a scale parameter). Binomial and Poisson models do not have auxiliary parameters. `prior_aux` can be a call to `exponential` to use an exponential distribution, or `normal`, `student_t` or `cauchy`, which results in a half-normal, half-t, or half-Cauchy prior. See `priors` for details on these functions. To omit a prior —i.e., to use a flat (improper) uniform prior— set `prior_aux` to `NULL`.
`prior_PD`	A logical scalar (defaulting to `FALSE`) indicating whether to draw from the prior predictive distribution instead of conditioning on the outcome.
`algorithm`	A string (possibly abbreviated) indicating the estimation approach to use. Can be `"sampling"` for MCMC (the default), `"optimizing"` for optimization, `"meanfield"` for variational inference with independent normal distributions, or `"fullrank"` for variational inference with a multivariate normal distribution. See `rstanarm-package` for more details on the estimation algorithms. NOTE: not all fitting functions support all four algorithms.
`adapt_delta`	Only relevant if `algorithm="sampling"`. See `adapt_delta` for details.
`QR`	A logical scalar defaulting to `FALSE`, but if `TRUE` applies a scaled `qr` decomposition to the design matrix, $X = Q^* R^*$, where $Q^* = Q\sqrt{n-1}$ and $R^* = \frac{1}{\sqrt{n-1}}R$. The coefficients relative to Q^* are obtained and then premultiplied by the inverse of R^* to obtain coefficients relative to the original predictors, X. These transformations do not change the likelihood of the data but are recommended for computational reasons when there are multiple predictors. Importantly, while the columns of X are almost always correlated, the columns of Q^* are uncorrelated by design, which often makes sampling from the posterior easier. However, because when `QR` is `TRUE` the `prior` argument applies to the coefficients relative to Q^* (and those are not very interpretable), setting `QR=TRUE` is only recommended if you do not have an informative prior for the regression coefficients.
`sparse`	A logical scalar (defaulting to `FALSE`) indicating whether to use a sparse representation of the design (X) matrix. Setting this to `TRUE` will likely be twice as slow, even if the design matrix has a considerable number of zeros, but it may allow the model to be estimated when the computer has too little RAM to utilize a dense design matrix. If `TRUE`, the the design matrix is not centered (since that

	would destroy the sparsity) and it is not possible to specify both `QR = TRUE` and `sparse = TRUE`.
`link`	For `stan_glm.nb` only, the link function to use. See `neg_binomial_2`.
`prior_ops`	Deprecated. See rstanarm-deprecated for details.
`group`	A list, possibly of length zero (the default), but otherwise having the structure of that produced by `mkReTrms` to indicate the group-specific part of the model. In addition, this list must have elements for the `regularization`, `concentration` `shape`, and `scale` components of a `decov` prior for the covariance matrices among the group-specific coefficients.

Details

The `stan_glm` function is similar in syntax to `glm` but rather than performing maximum likelihood estimation of generalized linear models, full Bayesian estimation is performed (if `algorithm` is `"sampling"`) via MCMC. The Bayesian model adds priors (independent by default) on the coefficients of the GLM. The `stan_glm` function calls the workhorse `stan_glm.fit` function, but it is also possible to call the latter directly.

The `stan_glm.nb` function, which takes the extra argument `link`, is a wrapper for `stan_glm` with `family = neg_binomial_2(link)`.

Value

A stanreg object is returned for `stan_glm, stan_glm.nb`.

A stanfit object (or a slightly modified stanfit object) is returned if `stan_glm.fit` is called directly.

References

Gelman, A. and Hill, J. (2007). *Data Analysis Using Regression and Multilevel/Hierarchical Models.* Cambridge University Press, Cambridge, UK. (Ch. 3-6)

See Also

`stanreg-methods` and `glm`.

The various vignettes for `stan_glm`.

Examples

```
if (!grepl("^sparc",  R.version$platform)) {
### Linear regression
fit <- stan_glm(mpg / 10 ~ ., data = mtcars, QR = TRUE,
                algorithm = "fullrank") # for speed of example only
plot(fit, prob = 0.5)
plot(fit, prob = 0.5, pars = "beta")
}

### Logistic regression
head(wells)
wells$dist100 <- wells$dist / 100
fit2 <- stan_glm(
  switch ~ dist100 + arsenic,
  data = wells,
  family = binomial(link = "logit"),
  prior_intercept = normal(0, 10),
  QR = TRUE,
  chains = 2, iter = 200 # for speed of example only
)
print(fit2)
prior_summary(fit2)

plot(fit2, plotfun = "areas", prob = 0.9, # ?bayesplot::mcmc_areas
     pars = c("(Intercept)", "arsenic"))
pp_check(fit2, plotfun = "error_binned")  # ?bayesplot::ppc_error_binned


### Poisson regression (example from help("glm"))
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
fit3 <- stan_glm(counts ~ outcome + treatment, family = poisson(link="log"),
                 prior = normal(0, 1), prior_intercept = normal(0, 5),
                 chains = 2, iter = 250) # for speed of example only
print(fit3)

bayesplot::color_scheme_set("green")
plot(fit3)
plot(fit3, regex_pars = c("outcome", "treatment"))
plot(fit3, plotfun = "combo", regex_pars = "treatment") # ?bayesplot::mcmc_combo

### Gamma regression (example from help("glm"))
clotting <- data.frame(log_u = log(c(5,10,15,20,30,40,60,80,100)),
                       lot1 = c(118,58,42,35,27,25,21,19,18),
                       lot2 = c(69,35,26,21,18,16,13,12,12))
fit4 <- stan_glm(lot1 ~ log_u, data = clotting, family = Gamma(link="log"),
                 chains = 2, iter = 300) # for speed of example only
print(fit4, digits = 2)
fit5 <- update(fit4, formula = lot2 ~ log_u)

### Negative binomial regression
fit6 <- stan_glm.nb(Days ~ Sex/(Age + Eth*Lrn), data = MASS::quine,
                    link = "log", prior_aux = exponential(1),
                    chains = 2, iter = 200) # for speed of example only

prior_summary(fit6)
bayesplot::color_scheme_set("brightblue")
plot(fit6)
pp_check(fit6, plotfun = "hist", nreps = 5)

# 80% interval of estimated reciprocal_dispersion parameter
posterior_interval(fit6, pars = "reciprocal_dispersion", prob = 0.8)
plot(fit6, "areas", pars = "reciprocal_dispersion", prob = 0.8)
```

stan_glmer *Bayesian generalized linear models with group-specific terms via Stan*

Description

Bayesian inference for GLMs with group-specific coefficients that have unknown covariance matrices with flexible priors.

Usage

```
stan_glmer(formula, data = NULL, family = gaussian, subset, weights,
  na.action = getOption("na.action", "na.omit"), offset, contrasts = NULL,
  ..., prior = normal(), prior_intercept = normal(), prior_aux = cauchy(0,
  5), prior_covariance = decov(), prior_PD = FALSE,
  algorithm = c("sampling", "meanfield", "fullrank"), adapt_delta = NULL,
  QR = FALSE, sparse = FALSE)

stan_lmer(formula, data = NULL, subset, weights,
  na.action = getOption("na.action", "na.omit"), offset, contrasts = NULL,
  ..., prior = normal(), prior_intercept = normal(), prior_aux = cauchy(0,
  5), prior_covariance = decov(), prior_PD = FALSE,
  algorithm = c("sampling", "meanfield", "fullrank"), adapt_delta = NULL,
  QR = FALSE)

stan_glmer.nb(formula, data = NULL, subset, weights,
  na.action = getOption("na.action", "na.omit"), offset, contrasts = NULL,
  link = "log", ..., prior = normal(), prior_intercept = normal(),
  prior_aux = cauchy(0, 5), prior_covariance = decov(), prior_PD = FALSE,
  algorithm = c("sampling", "meanfield", "fullrank"), adapt_delta = NULL,
  QR = FALSE)
```

Arguments

`formula, data, family`
: Same as for `glmer`. *We strongly advise against omitting the* `data` *argument*. Unless `data` is specified (and is a data frame) many post-estimation functions (including `update`, `loo`, `kfold`) are not guaranteed to work properly.

`subset, weights, offset`
: Same as `glm`.

`na.action, contrasts`
: Same as `glm`, but rarely specified.

`...`
: For `stan_glmer`, further arguments passed to `sampling` (e.g. `iter`, `chains`, `cores`, etc.) or to `vb` (if `algorithm` is `"meanfield"` or `"fullrank"`). For `stan_lmer` and `stan_glmer.nb`, `...` should also contain all relevant arguments to pass to `stan_glmer` (except `family`).

`prior` The prior distribution for the regression coefficients. `prior` should be a call to one of the various functions provided by **rstanarm** for specifying priors. The subset of these functions that can be used for the prior on the coefficients can be grouped into several "families":

Family	**Functions**
Student t family	`normal`, `student_t`, `cauchy`
Hierarchical shrinkage family	`hs`, `hs_plus`
Laplace family	`laplace`, `lasso`
Product normal family	`product_normal`

See the priors help page for details on the families and how to specify the arguments for all of the functions in the table above. To omit a prior —i.e., to use a flat (improper) uniform prior— `prior` can be set to `NULL`, although this is rarely a good idea.

Note: Unless `QR=TRUE`, if `prior` is from the Student t family or Laplace family, and if the `autoscale` argument to the function used to specify the prior (e.g. `normal`) is left at its default and recommended value of `TRUE`, then the default or user-specified prior scale(s) may be adjusted internally based on the scales of the predictors. See the priors help page for details on the rescaling and the `prior_summary` function for a summary of the priors used for a particular model.

`prior_intercept`

The prior distribution for the intercept. `prior_intercept` can be a call to `normal`, `student_t` or `cauchy`. See the priors help page for details on these functions. To omit a prior on the intercept —i.e., to use a flat (improper) uniform prior— `prior_intercept` can be set to `NULL`.

Note: If using a dense representation of the design matrix —i.e., if the `sparse` argument is left at its default value of `FALSE`— then the prior distribution for the intercept is set so it applies to the value when all predictors are centered. If you prefer to specify a prior on the intercept without the predictors being auto-centered, then you have to omit the intercept from the `formula` and include a column of ones as a predictor, in which case some element of `prior` specifies the prior on it, rather than `prior_intercept`.

`prior_aux` The prior distribution for the "auxiliary" parameter (if applicable). The "auxiliary" parameter refers to a different parameter depending on the `family`. For Gaussian models `prior_aux` controls `"sigma"`, the error standard deviation. For negative binomial models `prior_aux` controls `"reciprocal_dispersion"`, which is similar to the `"size"` parameter of `rnbinom`: smaller values of `"reciprocal_dispersion"` correspond to greater dispersion. For gamma models `prior_aux` sets the prior on to the `"shape"` parameter (see e.g., `rgamma`), and for inverse-Gaussian models it is the so-called `"lambda"` parameter (which is essentially the reciprocal of a scale parameter). Binomial and Poisson models do not have auxiliary parameters.

`prior_aux` can be a call to `exponential` to use an exponential distribution, or `normal`, `student_t` or `cauchy`, which results in a half-normal, half-t, or half-

Cauchy prior. See `priors` for details on these functions. To omit a prior —i.e., to use a flat (improper) uniform prior— set `prior_aux` to `NULL`.

`prior_covariance`
: Cannot be `NULL`; see `decov` for more information about the default arguments.

`prior_PD`
: A logical scalar (defaulting to `FALSE`) indicating whether to draw from the prior predictive distribution instead of conditioning on the outcome.

`algorithm`
: A string (possibly abbreviated) indicating the estimation approach to use. Can be `"sampling"` for MCMC (the default), `"optimizing"` for optimization, `"meanfield"` for variational inference with independent normal distributions, or `"fullrank"` for variational inference with a multivariate normal distribution. See `rstanarm-package` for more details on the estimation algorithms. NOTE: not all fitting functions support all four algorithms.

`adapt_delta`
: Only relevant if `algorithm="sampling"`. See `adapt_delta` for details.

`QR`
: A logical scalar defaulting to `FALSE`, but if `TRUE` applies a scaled `qr` decomposition to the design matrix, $X = Q^* R^*$, where $Q^* = Q\sqrt{n-1}$ and $R^* = \frac{1}{\sqrt{n-1}} R$. The coefficients relative to Q^* are obtained and then premultiplied by the inverse of R^* to obtain coefficients relative to the original predictors, X. These transformations do not change the likelihood of the data but are recommended for computational reasons when there are multiple predictors. Importantly, while the columns of X are almost always correlated, the columns of Q^* are uncorrelated by design, which often makes sampling from the posterior easier. However, because when `QR` is `TRUE` the `prior` argument applies to the coefficients relative to Q^* (and those are not very interpretable), setting `QR=TRUE` is only recommended if you do not have an informative prior for the regression coefficients.

`sparse`
: A logical scalar (defaulting to `FALSE`) indicating whether to use a sparse representation of the design (X) matrix. Setting this to `TRUE` will likely be twice as slow, even if the design matrix has a considerable number of zeros, but it may allow the model to be estimated when the computer has too little RAM to utilize a dense design matrix. If `TRUE`, the the design matrix is not centered (since that would destroy the sparsity) and it is not possible to specify both `QR = TRUE` and `sparse = TRUE`.

`link`
: For `stan_glmer.nb` only, the link function to use. See `neg_binomial_2`.

Details

The `stan_glmer` function is similar in syntax to `glmer` but rather than performing (restricted) maximum likelihood estimation of generalized linear models, Bayesian estimation is performed via MCMC. The Bayesian model adds priors on the regression coefficients (in the same way as `stan_glm`) and priors on the terms of a decomposition of the covariance matrices of the group-specific parameters. See `priors` for more information about the priors.

The `stan_lmer` function is equivalent to `stan_glmer` with `family = gaussian(link = "identity")`.

The `stan_glmer.nb` function, which takes the extra argument `link`, is a wrapper for `stan_glmer` with `family =    neg_binomial_2(link)`.

Value

A stanreg object is returned for `stan_glmer, stan_lmer, stan_glmer.nb`.

References

Gelman, A. and Hill, J. (2007). *Data Analysis Using Regression and Multilevel/Hierarchical Models.* Cambridge University Press, Cambridge, UK. (Ch. 11-15)

See Also

`stanreg-methods` and `glmer`.

The vignette for `stan_glmer` and the *Hierarchical Partial Pooling* vignette.

Examples

```
# see help(example_model) for details on the model below
if (!exists("example_model")) example(example_model)
print(example_model, digits = 1)
```

`stan_polr` *Bayesian ordinal regression models via Stan*

Description

Bayesian inference for ordinal (or binary) regression models under a proportional odds assumption.

Usage

```
stan_polr(formula, data, weights, ..., subset,
  na.action = getOption("na.action", "na.omit"), contrasts = NULL,
  model = TRUE, method = c("logistic", "probit", "loglog", "cloglog",
  "cauchit"), prior = R2(stop("'location' must be specified")),
  prior_counts = dirichlet(1), shape = NULL, rate = NULL,
  prior_PD = FALSE, algorithm = c("sampling", "meanfield", "fullrank"),
  adapt_delta = NULL, do_residuals = NULL)

stan_polr.fit(x, y, wt = NULL, offset = NULL, method = c("logistic",
  "probit", "loglog", "cloglog", "cauchit"), ...,
  prior = R2(stop("'location' must be specified")),
  prior_counts = dirichlet(1), shape = NULL, rate = NULL,
  prior_PD = FALSE, algorithm = c("sampling", "meanfield", "fullrank"),
  adapt_delta = NULL, do_residuals = algorithm == "sampling")
```

Arguments

`formula, data, subset`	Same as `polr`, but *we strongly advise against omitting the* `data` *argument*. Unless `data` is specified (and is a data frame) many post-estimation functions (including `update`, `loo`, `kfold`) are not guaranteed to work properly.
`weights, na.action, contrasts, model`	Same as `polr`, but rarely specified.
`...`	Further arguments passed to the function in the **rstan** package (`sampling`, `vb`, or `optimizing`), corresponding to the estimation method named by `algorithm`. For example, if `algorithm` is `"sampling"` it is possibly to specify `iter`, `chains`, `cores`, `refresh`, etc.
`method`	One of 'logistic', 'probit', 'loglog', 'cloglog' or 'cauchit', but can be abbreviated. See `polr` for more details.
`prior`	Prior for coefficients. Should be a call to `R2` to specify the prior location of the R^2 but can be `NULL` to indicate a standard uniform prior. See `priors`.
`prior_counts`	A call to `dirichlet` to specify the prior counts of the outcome when the predictors are at their sample means.
`shape`	Either `NULL` or a positive scalar that is interpreted as the shape parameter for a `GammaDist`ribution on the exponent applied to the probability of success when there are only two outcome categories. If `NULL`, which is the default, then the exponent is taken to be fixed at 1.
`rate`	Either `NULL` or a positive scalar that is interpreted as the rate parameter for a `GammaDist`ribution on the exponent applied to the probability of success when there are only two outcome categories. If `NULL`, which is the default, then the exponent is taken to be fixed at 1.
`prior_PD`	A logical scalar (defaulting to `FALSE`) indicating whether to draw from the prior predictive distribution instead of conditioning on the outcome.
`algorithm`	A string (possibly abbreviated) indicating the estimation approach to use. Can be `"sampling"` for MCMC (the default), `"optimizing"` for optimization, `"meanfield"` for variational inference with independent normal distributions, or `"fullrank"` for variational inference with a multivariate normal distribution. See `rstanarm-package` for more details on the estimation algorithms. NOTE: not all fitting functions support all four algorithms.
`adapt_delta`	Only relevant if `algorithm="sampling"`. See `adapt_delta` for details.
`do_residuals`	A logical scalar indicating whether or not to automatically calculate fit residuals after sampling completes. Defaults to `TRUE` if and only if `algorithm="sampling"`. Setting `do_residuals=FALSE` is only useful in the somewhat rare case that `stan_polr` appears to finish sampling but hangs instead of returning the fitted model object.
`x`	A design matrix.
`y`	A response variable, which must be a (preferably ordered) factor.
`wt`	A numeric vector (possibly `NULL`) of observation weights.
`offset`	A numeric vector (possibly `NULL`) of offsets.

Details

The `stan_polr` function is similar in syntax to `polr` but rather than performing maximum likelihood estimation of a proportional odds model, Bayesian estimation is performed (if `algorithm = "sampling"`) via MCMC. The `stan_polr` function calls the workhorse `stan_polr.fit` function, but it is possible to call the latter directly.

As for `stan_lm`, it is necessary to specify the prior location of R^2. In this case, the R^2 pertains to the proportion of variance in the latent variable (which is discretized by the cutpoints) attributable to the predictors in the model.

Prior beliefs about the cutpoints are governed by prior beliefs about the outcome when the predictors are at their sample means. Both of these are explained in the help page on `priors` and in the **rstanarm** vignettes.

Unlike `polr`, `stan_polr` also allows the "ordinal" outcome to contain only two levels, in which case the likelihood is the same by default as for `stan_glm` with `family = binomial` but the prior on the coefficients is different. However, `stan_polr` allows the user to specify the `shape` and `rate` hyperparameters, in which case the probability of success is defined as the logistic CDF of the linear predictor, raised to the power of `alpha` where `alpha` has a gamma prior with the specified `shape` and `rate`. This likelihood is called "scobit" by Nagler (1994) because if `alpha` is not equal to 1, then the relationship between the linear predictor and the probability of success is skewed. If `shape` or `rate` is `NULL`, then `alpha` is assumed to be fixed to 1.

Otherwise, it is usually advisible to set `shape` and `rate` to the same number so that the expected value of `alpha` is 1 while leaving open the possibility that `alpha` may depart from 1 a little bit. It is often necessary to have a lot of data in order to estimate `alpha` with much precision and always necessary to inspect the Pareto shape parameters calculated by `loo` to see if the results are particularly sensitive to individual observations.

Users should think carefully about how the outcome is coded when using a scobit-type model. When `alpha` is not 1, the asymmetry implies that the probability of success is most sensitive to the predictors when the probability of success is less than 0.63. Reversing the coding of the successes and failures allows the predictors to have the greatest impact when the probability of failure is less than 0.63. Also, the gamma prior on `alpha` is positively skewed, but you can reverse the coding of the successes and failures to circumvent this property.

Value

A stanreg object is returned for `stan_polr`.

A stanfit object (or a slightly modified stanfit object) is returned if `stan_polr.fit` is called directly.

References

Nagler, J., (1994). Scobit: An Alternative Estimator to Logit and Probit. *American Journal of Political Science*. 230 – 255.

See Also

`stanreg-methods` and `polr`.

The vignette for `stan_polr`.

Examples

```
if (!grepl("^sparc",  R.version$platform)) {
 fit <- stan_polr(tobgp ~ agegp, data = esoph, method = "probit",
          prior = R2(0.2, "mean"), init_r = 0.1, seed = 12345,
          algorithm = "fullrank") # for speed only
 print(fit)
 plot(fit)
}
```

`summary.stanreg` *Summary method for stanreg objects*

Description

Summaries of parameter estimates and MCMC convergence diagnostics (Monte Carlo error, effective sample size, Rhat).

Usage

```
## S3 method for class 'stanreg'
summary(object, pars = NULL, regex_pars = NULL,
  probs = NULL, ..., digits = 1)

## S3 method for class 'summary.stanreg'
print(x, digits = max(1, attr(x, "print.digits")),
  ...)

## S3 method for class 'summary.stanreg'
as.data.frame(x, ...)
```

Arguments

`object`	A fitted model object returned by one of the **rstanarm** modeling functions. See `stanreg-objects`.
`pars`	An optional character vector specifying a subset of parameters to display. Parameters can be specified by name or several shortcuts can be used. Using `pars="beta"` will restrict the displayed parameters to only the regression coefficients (without the intercept). `"alpha"` can also be used as a shortcut for `"(Intercept)"`. If the model has varying intercepts and/or slopes they can be selected using `pars = "varying"`. If `pars` is `NULL` all parameters are selected. See Examples.
`regex_pars`	An optional character vector of regular expressions to use for parameter selection. `regex_pars` can be used in place of `pars` or in addition to `pars`. Currently, all functions that accept a `regex_pars` argument ignore it for models fit using optimization.

probs	For models fit using MCMC or one of the variational algorithms, an optional numeric vector of probabilities passed to quantile.
...	Currently ignored.
digits	Number of digits to use for formatting numbers when printing. When calling summary, the value of digits is stored as the "print.digits" attribute of the returned object.
x	An object of class "summary.stanreg".

Value

The summary method returns an object of class "summary.stanreg", which is a matrix of summary statistics and diagnostics, with attributes storing information for use by the print method. The print method for summary.stanreg objects is called for its side effect and just returns its input. The as.data.frame method for summary.stanreg objects converts the matrix to a data.frame, preserving row and column names but dropping the print-related attributes.

See Also

prior_summary to extract or print a summary of the priors used for a particular model.

Examples

```
if (!exists("example_model")) example(example_model)
summary(example_model, probs = c(0.1, 0.9))

# These produce the same output for this example,
# but the second method can be used for any model
summary(example_model, pars = c("(Intercept)", "size",
                                paste0("period", 2:4)))
summary(example_model, pars = c("alpha", "beta"))

# Only show parameters varying by group
summary(example_model, pars = "varying")
as.data.frame(summary(example_model, pars = "varying"))
```

Index