

Package ‘brms’

May 14, 2018

Encoding UTF-8

Type Package

Title Bayesian Regression Models using 'Stan'

Version 2.3.0

Date 2018-05-14

Depends R (>= 3.2.0), Rcpp (>= 0.12.0), ggplot2 (>= 2.0.0), methods

Imports rstan (>= 2.17.2), loo (>= 2.0.0), Matrix (>= 1.1.1), mgcv (>= 1.8-13), rstantools (>= 1.3.0), bayesplot (>= 1.5.0), shinystan (>= 2.4.0), bridgesampling (>= 0.3-0), matrixStats, nleqslv, nlme, coda, abind, stats, utils, parallel, grDevices,

Suggests testthat (>= 0.9.1), RWiener, future, mice, spdep, mnormt, lme4, MCMCglmm, ape, arm, digest, R.rsp, knitr, rmarkdown

Description Fit Bayesian generalized (non-)linear multivariate multilevel models using 'Stan' for full Bayesian inference. A wide range of distributions and link functions are supported, allowing users to fit -- among others -- linear, robust linear, count data, survival, response times, ordinal, zero-inflated, hurdle, and even self-defined mixture models all in a multilevel context. Further modeling options include non-linear and smooth terms, auto-correlation structures, censored data, meta-analytic standard errors, and quite a few more. In addition, all parameters of the response distribution can be predicted in order to perform distributional regression. Prior specifications are flexible and explicitly encourage users to apply prior distributions that actually reflect their beliefs. Model fit can easily be assessed and compared with posterior predictive checks and leave-one-out cross-validation. References: Bürkner (2017) <doi:10.18637/jss.v080.i01>; Carpenter et al. (2017) <doi:10.18637/jss.v076.i01>.

LazyData true

NeedsCompilation no

License GPL (>= 3)

URL <https://github.com/paul-buerkner/brms>,
<https://groups.google.com/forum/#!forum/brms-users>

BugReports <https://github.com/paul-buerkner/brms/issues>

VignetteBuilder knitr, R.rsp

RoxygenNote 6.0.1

Author Paul-Christian Bürkner [aut, cre]

Maintainer Paul-Christian Bürkner <paul.buerkner@gmail.com>

Repository CRAN

Date/Publication 2018-05-14 15:46:34 UTC

R topics documented:

brms-package	5
addition-terms	6
add_ic	8
as.mcmc.brmsfit	9
AsymLaplace	10
autocor	11
bayes_factor.brmsfit	11
bayes_R2.brmsfit	12
bridge_sampler.brmsfit	14
brm	15
brmsfamily	21
brmsfit-class	25
brmsformula	26
brmsformula-helpers	34
brmshypothesis	36
brm_multiple	37
coef.brmsfit	38
combine_models	39
compare_ic	40
control_params.brmsfit	41
cor_ar	42
cor_arma	43
cor_arr	44
cor_brms	45
cor_bsts	46
cor_car	47
cor_fixed	48
cor_ma	49
cor_sar	50
cs	51
custom_family	52
epilepsy	53
ExGaussian	55
expose_functions.brmsfit	55
expp1	56
fitted.brmsfit	56

fixef.brmsfit	59
Frechet	60
GenExtremeValue	61
get_prior	62
gp	63
gr	65
horseshoe	66
hypothesis.brmsfit	68
inhaler	70
InvGaussian	71
inv_logit_scaled	72
is.brmsfit	73
is.brmsformula	73
is.brmsprior	73
is.brmsterms	74
is.cor_brms	74
is.mvbrmsformula	75
is.mvbrmsterms	75
kfold.brmsfit	76
kidney	78
lasso	79
launch_shinystan.brmsfit	80
logit_scaled	81
logm1	81
log_lik.brmsfit	82
log_posterior.brmsfit	83
loo.brmsfit	84
loo_model_weights.brmsfit	86
loo_predict.brmsfit	87
make_conditions	88
make_stancode	89
make_standata	91
marginal_effects.brmsfit	93
marginal_smooths.brmsfit	97
me	99
mi	100
mixture	101
mm	102
mmc	104
mo	105
model_weights.brmsfit	106
MultiNormal	107
MultiStudentT	108
mvbrmsformula	108
ngrps.brmsfit	109
nsamples.brmsfit	110
pairs.brmsfit	110
parnames	111

parse_bf	112
plot.brmsfit	113
posterior_average.brmsfit	114
posterior_interval.brmsfit	116
posterior_samples.brmsfit	117
posterior_summary.brmsfit	118
posterior_table	119
post_prob.brmsfit	120
pp_average.brmsfit	121
pp_check.brmsfit	123
pp_mixture.brmsfit	124
predict.brmsfit	127
print.brmsfit	130
print.brmsprior	130
prior_samples.brmsfit	131
prior_summary.brmsfit	132
ranef.brmsfit	133
reloo	134
residuals.brmsfit	135
restructure	137
rows2labels	138
s	138
set_prior	139
Shifted_Lognormal	144
SkewNormal	145
stancode.brmsfit	146
standata.brmsfit	147
stanplot.brmsfit	148
stanvar	149
StudentT	150
summary.brmsfit	151
theme_black	152
theme_default	153
update.brmsfit	153
VarCorr.brmsfit	154
vcov.brmsfit	155
VonMises	156
waic.brmsfit	157
Wiener	158

Description

The **brms** package provides an interface to fit Bayesian generalized multivariate (non-)linear multilevel models using **Stan**, which is a C++ package for obtaining full Bayesian inference (see <http://mc-stan.org/>). The formula syntax is an extended version of the syntax applied in the **lme4** package to provide a familiar and simple interface for performing regression analyses.

Details

The main function of **brms** is `brm`, which uses formula syntax to specify a wide range of complex Bayesian models (see `brmsformula` for details). Based on the supplied formulas, data, and additional information, it writes the Stan code on the fly via `make_stancode`, prepares the data via `make_standata`, and fits the model using **Stan**.

Subsequently, a large number of post-processing methods can be applied: To get an overview on the estimated parameters, `summary` or `marginal_effects` are perfectly suited. Detailed visual analyses can be performed by applying the `pp_check` and `stanplot` methods, which both rely on the `bayesplot` package. Model comparisons can be done via `loo` and `waic`, which make use of the `loo` package as well as via `bayes_factor` which relies on the `bridgesampling` package. For a full list of methods to apply, type `methods(class = "brmsfit")`.

Because **brms** is based on **Stan**, a C++ compiler is required. The program Rtools (available on <https://cran.r-project.org/bin/windows/Rtools/>) comes with a C++ compiler for Windows. On Mac, you should use Xcode. For further instructions on how to get the compilers running, see the prerequisites section at the [RStan-Getting-Started](#) page.

When comparing other packages fitting multilevel models to **brms**, keep in mind that the latter needs to compile models before actually fitting them, which will require between 20 and 40 seconds depending on your machine, operating system and overall model complexity.

Thus, fitting smaller models may be relatively slow as compilation time makes up the majority of the whole running time. For larger / more complex models however, fitting may take several minutes or even hours, so that the compilation time won't make much of a difference for these models.

See `vignette("brms_overview")` and `vignette("brms_multilevel")` for a general introduction and overview of **brms**. For a full list of available vignettes, type `vignette(package = "brms")`.

References

Paul-Christian Buerkner (2017). *brms: An R Package for Bayesian Multilevel Models Using Stan*. *Journal of Statistical Software*, 80(1), 1-28. doi:10.18637/jss.v080.i01

Paul-Christian Buerkner (in press). *Advanced Bayesian Multilevel Modeling with the R Package brms*. *The R Journal*.

The Stan Development Team. *Stan Modeling Language User's Guide and Reference Manual*. <http://mc-stan.org/users/documentation>.

Stan Development Team (2018). RStan: the R interface to Stan. R package version 2.18.1. <http://mc-stan.org>

See Also

[brm](#), [brmsformula](#), [brmsfamily](#), [brmsfit](#)

addition-terms

Additional Response Information

Description

Provide additional information on the response variable in **brms** models, such as censoring, truncation, or known measurement error.

Usage

```
resp_se(x, sigma = FALSE)
```

```
resp_weights(x, scale = FALSE)
```

```
resp_trials(x)
```

```
resp_cat(x)
```

```
resp_dec(x)
```

```
resp_cens(x, y2 = NULL)
```

```
resp_trunc(lb = -Inf, ub = Inf)
```

```
resp_mi(sdy = NULL)
```

Arguments

x	A vector; usually a variable defined in the data. Allowed values depend on the function: <code>resp_se</code> and <code>resp_weights</code> require positive numeric values. <code>resp_trials</code> and <code>resp_cat</code> require positive integers. <code>resp_dec</code> requires 0 and 1, or alternatively 'lower' and 'upper'; <code>resp_cens</code> requires 'left', 'none', 'right', and 'interval' (or equivalently -1, 0, 1, and 2) to indicate left, no, right, or interval censoring.
sigma	Logical; Indicates whether the residual standard deviation parameter <code>sigma</code> should be included in addition to the known measurement error. Defaults to <code>FALSE</code> for backwards compatibility, but setting it to <code>TRUE</code> is usually the better choice.
scale	Logical; Indicates whether weights should be scaled so that the average weight equals one. Defaults to <code>FALSE</code> .
y2	A vector specifying the upper bounds in interval censoring.
lb	A numeric vector or single numeric value specifying the lower truncation bound.
ub	A numeric vector or single numeric value specifying the upper truncation bound.

`sdv` Optional known measurement error of the response treated as standard deviation. If specified, handles measurement error and (completely) missing values at the same time using the plausible-values-technique.

Details

These functions are almost solely useful when called in formulas passed to the **brms** package. Within formulas, the `resp_` prefix may be omitted. More information is given in the 'Details' section of [brmsformula](#).

Value

A vector containing additional information on the response variable in an appropriate format.

See Also

[brm](#), [brmsformula](#)

Examples

```
## Not run:
## Random effects meta-analysis
nstudies <- 20
true_effects <- rnorm(nstudies, 0.5, 0.2)
sei <- runif(nstudies, 0.05, 0.3)
outcomes <- rnorm(nstudies, true_effects, sei)
data1 <- data.frame(outcomes, sei)
fit1 <- brm(outcomes | se(sei, sigma = TRUE) ~ 1,
            data = data1)
summary(fit1)

## Probit regression using the binomial family
n <- sample(1:10, 100, TRUE) # number of trials
success <- rbinom(100, size = n, prob = 0.4)
x <- rnorm(100)
data2 <- data.frame(n, success, x)
fit2 <- brm(success | trials(n) ~ x, data = data2,
            family = binomial("probit"))
summary(fit2)

## Survival regression modeling the time between the first
## and second recurrence of an infection in kidney patients.
fit3 <- brm(time | cens(censored) ~ age * sex + disease + (1|patient),
            data = kidney, family = lognormal())
summary(fit3)

## Poisson model with truncated counts
fit4 <- brm(count | trunc(ub = 104) ~ log_Base4_c * Trt_c,
            data = epilepsy, family = poisson())
summary(fit4)

## End(Not run)
```

add_ic	<i>Add information criteria and fit indices to fitted model objects</i>
--------	---

Description

Add information criteria and fit indices to fitted model objects

Usage

```
add_ic(x, ...)

## S3 method for class 'brmsfit'
add_ic(x, ic = "loo", model_name = NULL, ...)

add_ic(x, ...) <- value

add_loo(x, ...)

add_waic(x, ...)
```

Arguments

x	An R object typically of class <code>brmsfit</code> .
...	Further arguments passed to the underlying functions computing the information criteria or fit indices.
ic, value	Names of the information criteria / fit indices to compute. Currently supported are "loo", "waic", "kfold", "R2" (R-squared), and "marglik" (log marginal likelihood).
model_name	Optional name of the model. If NULL (the default) the name is taken from the call to x.

Details

The methods `add_loo` and `add_waic` are just convenient wrappers around `add_ic`.

Value

An object of the same class as x, but with information criteria added for later usage.

Examples

```
## Not run:
fit <- brm(count ~ Trt, epilepsy, poisson())
# add both LOO and WAIC at once
fit <- add_ic(fit, ic = c("loo", "waic"))
print(fit$loo)
print(fit$waic)

## End(Not run)
```

as.mcmc.brmsfit

*Extract posterior samples for use with the **coda** package***Description**

Extract posterior samples for use with the **coda** package

Usage

```
## S3 method for class 'brmsfit'
as.mcmc(x, pars = NA, exact_match = FALSE,
        combine_chains = FALSE, inc_warmup = FALSE, ...)
```

Arguments

<code>x</code>	An R object typically of class <code>brmsfit</code>
<code>pars</code>	Names of parameters for which posterior samples should be returned, as given by a character vector or regular expressions. By default, all posterior samples of all parameters are extracted.
<code>exact_match</code>	Indicates whether parameter names should be matched exactly or treated as regular expression. Default is <code>FALSE</code> .
<code>combine_chains</code>	Indicates whether chains should be combined.
<code>inc_warmup</code>	Indicates if the warmup samples should be included. Default is <code>FALSE</code> . Warmup samples are used to tune the parameters of the sampling algorithm and should not be analyzed.
<code>...</code>	currently unused

Value

If `combine_chains = TRUE` an `mcmc` object is returned. If `combine_chains = FALSE` an `mcmc.list` object is returned.

Description

Density, distribution function, quantile function and random generation for the asymmetric Laplace distribution with location μ , scale σ and asymmetry parameter q .

Usage

```
dasym_laplace(x, mu = 0, sigma = 1, quantile = 0.5, log = FALSE)
```

```
pasym_laplace(q, mu = 0, sigma = 1, quantile = 0.5, lower.tail = TRUE,
log.p = FALSE)
```

```
qasym_laplace(p, mu = 0, sigma = 1, quantile = 0.5, lower.tail = TRUE,
log.p = FALSE)
```

```
rasym_laplace(n, mu = 0, sigma = 1, quantile = 0.5)
```

Arguments

<code>x, q</code>	Vector of quantiles.
<code>mu</code>	Vector of locations.
<code>sigma</code>	Vector of scales.
<code>quantile</code>	Asymmetry parameter corresponding to quantiles in quantile regression (hence the name).
<code>log</code>	Logical; If TRUE, values are returned on the log scale.
<code>lower.tail</code>	Logical; If TRUE (default), return $P(X \leq x)$. Else, return $P(X > x)$.
<code>log.p</code>	Logical; If TRUE, values are returned on the log scale.
<code>p</code>	Vector of probabilities.
<code>n</code>	Number of samples to draw from the distribution.

Details

See `vignette("brms_families")` for details on the parameterization.

autocor *Extract Autocorrelation Structures*

Description

Extract Autocorrelation Structures

Usage

```
autocor(object, ...)
```

Arguments

object	An R object.
...	Further arguments passed to or from other methods.

Value

What exactly is returned depends on the specific method.

bayes_factor.brmsfit *Bayes Factors from Marginal Likelihoods*

Description

Compute Bayes factors from marginal likelihoods.

Usage

```
## S3 method for class 'brmsfit'
bayes_factor(x1, x2, log = FALSE, ...)
```

Arguments

x1	A brmsfit object
x2	Another brmsfit object based on the same responses.
log	Report Bayes factors on the log-scale?
...	Additional arguments passed to bridge_sampler .

Details

Computing the marginal likelihood requires samples of all variables defined in Stan's parameters block to be saved. Otherwise bayes_factor cannot be computed. Thus, please set `save_all_pars = TRUE` in the call to `brm`, if you are planning to apply `bayes_factor` to your models.

More details are provided under [bayes_factor](#).

See Also

[bridge_sampler](#), [post_prob](#)

Examples

```
## Not run:
# model with the treatment effect
fit1 <- brm(
  count ~ log_Age_c + log_Base4_c + Trt_c,
  data = epilepsy, family = negbinomial(),
  prior = prior(normal(0, 1), class = b),
  save_all_pars = TRUE
)
summary(fit1)

# model without the treatment effect
fit2 <- brm(
  count ~ log_Age_c + log_Base4_c,
  data = epilepsy, family = negbinomial(),
  prior = prior(normal(0, 1), class = b),
  save_all_pars = TRUE
)
summary(fit2)

# compute the bayes factor
bayes_factor(fit1, fit2)

## End(Not run)
```

bayes_R2.brmsfit

Compute a Bayesian version of R-squared for regression models

Description

Compute a Bayesian version of R-squared for regression models

Usage

```
## S3 method for class 'brmsfit'
bayes_R2(object, newdata = NULL, re_formula = NULL,
  allow_new_levels = FALSE, sample_new_levels = "uncertainty",
  new_objects = list(), incl_autocor = TRUE, subset = NULL,
  nsamples = NULL, resp = NULL, nug = NULL, summary = TRUE,
  robust = FALSE, probs = c(0.025, 0.975), ...)
```

Arguments

<code>object</code>	An object of class <code>brmsfit</code>
<code>newdata</code>	An optional <code>data.frame</code> for which to evaluate predictions. If <code>NULL</code> (default), the original data of the model is used.
<code>re_formula</code>	formula containing group-level effects to be considered in the prediction. If <code>NULL</code> (default), include all group-level effects; if <code>NA</code> , include no group-level effects.
<code>allow_new_levels</code>	A flag indicating if new levels of group-level effects are allowed (defaults to <code>FALSE</code>). Only relevant if <code>newdata</code> is provided.
<code>sample_new_levels</code>	Indicates how to sample new levels for grouping factors specified in <code>re_formula</code> . This argument is only relevant if <code>newdata</code> is provided and <code>allow_new_levels</code> is set to <code>TRUE</code> . If <code>"uncertainty"</code> (default), include group-level uncertainty in the predictions based on the variation of the existing levels. If <code>"gaussian"</code> , sample new levels from the (multivariate) normal distribution implied by the group-level standard deviations and correlations. This options may be useful for conducting Bayesian power analysis. If <code>"old_levels"</code> , directly sample new levels from the existing levels.
<code>new_objects</code>	A named list of objects containing new data, which cannot be passed via argument <code>newdata</code> . Currently, only required for objects passed to <code>cor_sar</code> and <code>cor_fixed</code> .
<code>incl_autocor</code>	A flag indicating if ARMA autocorrelation parameters should be included in the predictions. Defaults to <code>TRUE</code> . Setting it to <code>FALSE</code> will not affect other correlation structures such as <code>cor_bsts</code> , or <code>cor_fixed</code> .
<code>subset</code>	A numeric vector specifying the posterior samples to be used. If <code>NULL</code> (the default), all samples are used.
<code>nsamples</code>	Positive integer indicating how many posterior samples should be used. If <code>NULL</code> (the default) all samples are used. Ignored if <code>subset</code> is not <code>NULL</code> .
<code>resp</code>	Optional names of response variables. If specified, fitted values of these response variables are returned.
<code>nug</code>	Small positive number for Gaussian process terms only. For numerical reasons, the covariance matrix of a Gaussian process might not be positive definite. Adding a very small number to the matrix's diagonal often solves this problem. If <code>NULL</code> (the default), <code>nug</code> is chosen internally.
<code>summary</code>	Should summary statistics (i.e. means, sds, and 95% intervals) be returned instead of the raw values? Default is <code>TRUE</code> .
<code>robust</code>	If <code>FALSE</code> (the default) the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If <code>TRUE</code> , the median and the median absolute deviation (MAD) are applied instead. Only used if <code>summary</code> is <code>TRUE</code> .
<code>probs</code>	The percentiles to be computed by the <code>quantile</code> function. Only used if <code>summary</code> is <code>TRUE</code> .
<code>...</code>	Currently ignored.

Details

For an introduction to the approach, see https://github.com/jgabry/bayes_R2/blob/master/bayes_R2.pdf.

Value

If `summary = TRUE` a 1 x C matrix is returned (C = length(probs) + 2) containing summary statistics of Bayesian R-squared values. If `summary = FALSE` the posterior samples of the R-squared values are returned in a S x 1 matrix (S is the number of samples).

Examples

```
## Not run:
fit <- brm(mpg ~ wt + cyl, data = mtcars)
summary(fit)
bayes_R2(fit)

# compute R2 with new data
nd <- data.frame(mpg = c(10, 20, 30), wt = c(4, 3, 2), cyl = c(8, 6, 4))
bayes_R2(fit, newdata = nd)

## End(Not run)
```

bridge_sampler.brmsfit

Log Marginal Likelihood via Bridge Sampling

Description

Computes log marginal likelihood via bridge sampling, which can be used in the computation of bayes factors and posterior model probabilities. The `brmsfit` method is just a thin wrapper around the corresponding method for `stanfit` objects.

Usage

```
## S3 method for class 'brmsfit'
bridge_sampler(samples, ...)
```

Arguments

<code>samples</code>	A <code>brmsfit</code> object.
<code>...</code>	Additional arguments passed to <code>bridge_sampler.stanfit</code> .

Details

Computing the marginal likelihood requires samples of all variables defined in Stan's parameters block to be saved. Otherwise `bridge_sampler` cannot be computed. Thus, please set `save_all_pars = TRUE` in the call to `brm`, if you are planning to apply `bridge_sampler` to your models.

More details are provided under [bridge_sampler](#).

See Also

[bayes_factor](#), [post_prob](#)

Examples

```
## Not run:
# model with the treatment effect
fit1 <- brm(
  count ~ log_Age_c + log_Base4_c + Trt_c,
  data = epilepsy, family = negbinomial(),
  prior = prior(normal(0, 1), class = b),
  save_all_pars = TRUE
)
summary(fit1)
bridge_sampler(fit1)

# model without the treatment effect
fit2 <- brm(
  count ~ log_Age_c + log_Base4_c,
  data = epilepsy, family = negbinomial(),
  prior = prior(normal(0, 1), class = b),
  save_all_pars = TRUE
)
summary(fit2)
bridge_sampler(fit2)

## End(Not run)
```

brm

Fit Bayesian Generalized (Non-)Linear Multivariate Multilevel Models

Description

Fit Bayesian generalized (non-)linear multivariate multilevel models using Stan for full Bayesian inference. A wide range of distributions and link functions are supported, allowing users to fit – among others – linear, robust linear, count data, survival, response times, ordinal, zero-inflated, hurdle, and even self-defined mixture models all in a multilevel context. Further modeling options include non-linear and smooth terms, auto-correlation structures, censored data, meta-analytic standard errors, and quite a few more. In addition, all parameters of the response distributions can

be predicted in order to perform distributional regression. Prior specifications are flexible and explicitly encourage users to apply prior distributions that actually reflect their beliefs. In addition, model fit can easily be assessed and compared with posterior predictive checks and leave-one-out cross-validation.

Usage

```
brm(formula, data, family = gaussian(), prior = NULL, autocor = NULL,
    cov_ranef = NULL, sample_prior = c("no", "yes", "only"), sparse = FALSE,
    knots = NULL, stanvars = NULL, stan_funs = NULL, fit = NA,
    save_ranef = TRUE, save_mevars = FALSE, save_all_pars = FALSE,
    inits = "random", chains = 4, iter = 2000, warmup = floor(iter/2),
    thin = 1, cores = getOption("mc.cores", 1L), control = NULL,
    algorithm = c("sampling", "meanfield", "fullrank"),
    future = getOption("future", FALSE), silent = TRUE, seed = NA,
    save_model = NULL, save_dso = TRUE, ...)
```

Arguments

formula	An object of class formula , brmsformula , or mvbrmsformula (or one that can be coerced to that classes): A symbolic description of the model to be fitted. The details of model specification are explained in brmsformula .
data	An object of class <code>data.frame</code> (or one that can be coerced to that class) containing data of all variables used in the model.
family	A description of the response distribution and link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a <code>link</code> argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details of supported families see brmsfamily . By default, a linear gaussian model is applied. In multivariate models, family might also be a list of families.
prior	One or more <code>brmsprior</code> objects created by set_prior or related functions and combined using the <code>c</code> method or the <code>+</code> operator. See also get_prior for more help.
autocor	An optional cor_brms object describing the correlation structure within the response variable (i.e., the 'autocorrelation'). See the documentation of cor_brms for a description of the available correlation structures. Defaults to <code>NULL</code> , corresponding to no correlations. In multivariate models, <code>autocor</code> might also be a list of autocorrelation structures.
cov_ranef	A list of matrices that are proportional to the (within) covariance structure of the group-level effects. The names of the matrices should correspond to columns in <code>data</code> that are used as grouping factors. All levels of the grouping factor should appear as rownames of the corresponding matrix. This argument can be used, among others to model pedigrees and phylogenetic effects. See <code>vignette("brms_phylogenetics")</code> for more details.
sample_prior	Indicate if samples from all specified proper priors should be drawn additionally to the posterior samples (defaults to "no"). Among others, these samples

	can be used to calculate Bayes factors for point hypotheses. If set to "only", samples are drawn solely from the priors ignoring the likelihood. In this case, all parameters must have proper priors.
sparse	Logical; indicates whether the population-level design matrices should be treated as sparse (defaults to FALSE). For design matrices with many zeros, this can considerably reduce required memory. Sampling speed is currently not improved or even slightly decreased.
knots	Optional list containing user specified knot values to be used for basis construction of smoothing terms. See gamm for more details.
stanvars	An optional stanvars object generated by function stanvar to define additional variables in the data block of Stan .
stan_funs	An optional character string containing self-defined Stan functions, which will be included in the functions block of the generated Stan code.
fit	An instance of S3 class brmsfit derived from a previous fit; defaults to NA. If fit is of class brmsfit, the compiled model associated with the fitted result is re-used and all arguments modifying the model code or data are ignored. It is not recommended to use this argument directly, but to call the update method, instead.
save_ranef	A flag to indicate if group-level effects for each level of the grouping factor(s) should be saved (default is TRUE). Set to FALSE to save memory. The argument has no impact on the model fitting itself.
save_mevars	A flag to indicate if samples of latent noise-free variables obtained by using me and mi terms should be saved (default is FALSE). Saving these samples allows to better use methods such as predict with the latent variables but leads to very large R objects even for models of moderate size and complexity.
save_all_pars	A flag to indicate if samples from all variables defined in Stan's parameters block should be saved (default is FALSE). Saving these samples is required in order to apply the methods bridge_sampler, bayes_factor, and post_prob.
inits	Either "random" or "0". If inits is "random" (the default), Stan will randomly generate initial values for parameters. If it is "0", all parameters are initialized to zero. This option is sometimes useful for certain families, as it happens that default ("random") inits cause samples to be essentially constant. Generally, setting inits = "0" is worth a try, if chains do not behave well. Alternatively, inits can be a list of lists containing the initial values, or a function (or function name) generating initial values. The latter options are mainly implemented for internal testing.
chains	Number of Markov chains (defaults to 4).
iter	Number of total iterations per chain (including warmup; defaults to 2000).
warmup	A positive integer specifying number of warmup (aka burnin) iterations. This also specifies the number of iterations used for stepsize adaptation, so warmup samples should not be used for inference. The number of warmup should not be larger than iter and the default is iter/2.
thin	Thinning rate. Must be a positive integer. Set thin > 1 to save memory and computation time if iter is large.

cores	Number of cores to use when executing the chains in parallel, which defaults to 1 but we recommend setting the <code>mc.cores</code> option to be as many processors as the hardware and RAM allow (up to the number of chains). For non-Windows OS in non-interactive R sessions, forking is used instead of PSOCK clusters.
control	A named list of parameters to control the sampler's behavior. It defaults to NULL so all the default values are used. The most important control parameters are discussed in the 'Details' section below. For a comprehensive overview see stan .
algorithm	Character string indicating the estimation approach to use. Can be "sampling" for MCMC (the default), "meanfield" for variational inference with independent normal distributions, or "fullrank" for variational inference with a multivariate normal distribution.
future	Logical; If TRUE, the future package is used for parallel execution of the chains and argument <code>cores</code> will be ignored. Can be set globally for the current R session via the <code>future</code> option. The execution type is controlled via plan (see the examples section below).
silent	logical; If TRUE (the default), most of the informational messages of compiler and sampler are suppressed. The actual sampling progress is still printed. Set <code>refresh = 0</code> to turn this off as well.
seed	The seed for random number generation to make results reproducible. If NA (the default), Stan will set the seed randomly.
save_model	Either NULL or a character string. In the latter case, the model code is saved in a file named after the string supplied in <code>save_model</code> , which may also contain the full path where to save the file. If only a name is given, the file is saved in the current working directory.
save_dso	Logical, defaulting to TRUE, indicating whether the dynamic shared object (DSO) compiled from the C++ code for the model will be saved or not. If TRUE, we can draw samples from the same model in another R session using the saved DSO (i.e., without compiling the C++ code again).
...	Further arguments to be passed to Stan.

Details

Fit a generalized (non-)linear multivariate multilevel model via full Bayesian inference using Stan. A general overview is provided in the vignettes `vignette("brms_overview")` and `vignette("brms_multilevel")`. For a full list of available vignettes see `vignette(package = "brms")`.

Formula syntax of brms models

Details of the formula syntax applied in **brms** can be found in [brmsformula](#).

Families and link functions

Details of families supported by **brms** can be found in [brmsfamily](#).

Prior distributions

Priors should be specified using the [set_prior](#) function. Its documentation contains detailed information on how to correctly specify priors. To find out on which parameters or parameter classes priors can be defined, use [get_prior](#). Default priors are chosen to be non or very weakly informative so that their influence on the results will be negligible and you usually don't have to worry

about them. However, after getting more familiar with Bayesian statistics, I recommend you to start thinking about reasonable informative priors for your model parameters: Nearly always, there is at least some prior information available that can be used to improve your inference.

Adjusting the sampling behavior of Stan

In addition to choosing the number of iterations, warmup samples, and chains, users can control the behavior of the NUTS sampler, by using the `control` argument. The most important reason to use `control` is to decrease (or eliminate at best) the number of divergent transitions that cause a bias in the obtained posterior samples. Whenever you see the warning "There were x divergent transitions after warmup." you should really think about increasing `adapt_delta`. To do this, write `control = list(adapt_delta = <x>)`, where `<x>` should usually be value between 0.8 (current default) and 1. Increasing `adapt_delta` will slow down the sampler but will decrease the number of divergent transitions threatening the validity of your posterior samples.

Another problem arises when the depth of the tree being evaluated in each iteration is exceeded. This is less common than having divergent transitions, but may also bias the posterior samples. When it happens, **Stan** will throw out a warning suggesting to increase `max_treedepth`, which can be accomplished by writing `control = list(max_treedepth = <x>)` with a positive integer `<x>` that should usually be larger than the current default of 10. For more details on the `control` argument see [stan](#).

Value

An object of class `brmsfit`, which contains the posterior samples along with many other useful information about the model. Use `methods(class = "brmsfit")` for an overview on available methods.

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

References

Paul-Christian Buerkner (2017). *brms: An R Package for Bayesian Multilevel Models Using Stan*. *Journal of Statistical Software*, 80(1), 1-28. doi:10.18637/jss.v080.i01

Paul-Christian Buerkner (in review). *Advanced Bayesian Multilevel Modeling with the R Package brms*. arXiv preprint.

See Also

[brms](#), [brmsformula](#), [brmsfamily](#), [brmsfit](#)

Examples

```
## Not run:
# Poisson regression for the number of seizures in epileptic patients
# using student_t priors for population-level effects
# and half cauchy priors for standard deviations of group-level effects
bprior1 <- prior(student_t(5,0,10), class = b) +
  prior(cauchy(0,2), class = sd)
fit1 <- brm(count ~ log_Age_c + log_Base4_c * Trt + (1|patient),
```

```

      data = epilepsy, family = poisson(), prior = bprior1)

# generate a summary of the results
summary(fit1)

# plot the MCMC chains as well as the posterior distributions
plot(fit1, ask = FALSE)

# predict responses based on the fitted model
head(predict(fit1))

# plot marginal effects for each predictor
plot(marginal_effects(fit1), ask = FALSE)

# investigate model fit
loo(fit1)
pp_check(fit1)

# Ordinal regression modeling patient's rating of inhaler instructions
# category specific effects are estimated for variable 'treat'
fit2 <- brm(rating ~ period + carry + cs(treat),
            data = inhaler, family = sratio("logit"),
            prior = set_prior("normal(0,5)"), chains = 2)
summary(fit2)
plot(fit2, ask = FALSE)
WAIC(fit2)

# Survival regression modeling the time between the first
# and second recurrence of an infection in kidney patients.
fit3 <- brm(time | cens(censored) ~ age * sex + disease + (1|patient),
            data = kidney, family = lognormal())
summary(fit3)
plot(fit3, ask = FALSE)
plot(marginal_effects(fit3), ask = FALSE)

# Probit regression using the binomial family
ntrials <- sample(1:10, 100, TRUE)
success <- rbinom(100, size = ntrials, prob = 0.4)
x <- rnorm(100)
data4 <- data.frame(ntrials, success, x)
fit4 <- brm(success | trials(ntrials) ~ x, data = data4,
            family = binomial("probit"))
summary(fit4)

# Simple non-linear gaussian model
x <- rnorm(100)
y <- rnorm(100, mean = 2 - 1.5*x, sd = 1)
data5 <- data.frame(x, y)
bprior5 <- prior(normal(0, 2), nlpar = a1) +

```

```

  prior(normal(0, 2), nlpar = a2)
fit5 <- brm(bf(y ~ a1 - a2^x, a1 + a2 ~ 1, nl = TRUE),
           data = data5, prior = bprior5)
summary(fit5)
plot(marginal_effects(fit5), ask = FALSE)

# Normal model with heterogeneous variances
data_het <- data.frame(
  y = c(rnorm(50), rnorm(50, 1, 2)),
  x = factor(rep(c("a", "b"), each = 50))
)
fit6 <- brm(bf(y ~ x, sigma ~ 0 + x), data = data_het)
summary(fit6)
plot(fit6)
marginal_effects(fit6)

# extract estimated residual SDs of both groups
sigmas <- exp(posterior_samples(fit6, "^b_sigma_"))
ggplot(stack(sigmas), aes(values)) +
  geom_density(aes(fill = ind))

# Quantile regression predicting the 25%-quantile
fit7 <- brm(bf(y ~ x, quantile = 0.25), data = data_het,
           family = asym_laplace())
summary(fit7)
marginal_effects(fit7)

# use the future package for more flexible parallelization
library(future)
plan(multiprocess)
fit7 <- update(fit7, future = TRUE)

## End(Not run)

```

Description

Family objects provide a convenient way to specify the details of the models used by many model fitting functions. The family functions presented here are for use with **brms** only and will **not** work with other model fitting functions such as `glm` or `glmer`. However, the standard family functions as described in [family](#) will work with **brms**. You can also specify custom families for use in **brms** with the `custom_family` function.

Usage

```
brmsfamily(family, link = NULL, link_sigma = "log", link_shape = "log",
  link_nu = "logm1", link_phi = "log", link_kappa = "log",
  link_beta = "log", link_zi = "logit", link_hu = "logit",
  link_zoi = "logit", link_coi = "logit", link_disc = "log",
  link_bs = "log", link_ndt = "log", link_bias = "logit",
  link_xi = "log1p", link_alpha = "identity", link_quantile = "logit",
  threshold = c("flexible", "equidistant"))

student(link = "identity", link_sigma = "log", link_nu = "logm1")

bernoulli(link = "logit")

negbinomial(link = "log", link_shape = "log")

geometric(link = "log")

lognormal(link = "identity", link_sigma = "log")

shifted_lognormal(link = "identity", link_sigma = "log", link_ndt = "log")

skew_normal(link = "identity", link_sigma = "log",
  link_alpha = "identity")

exponential(link = "log")

weibull(link = "log", link_shape = "log")

frechet(link = "log", link_nu = "logm1")

gen_extreme_value(link = "identity", link_sigma = "log",
  link_xi = "log1p")

exgaussian(link = "identity", link_sigma = "log", link_beta = "log")

wiener(link = "identity", link_bs = "log", link_ndt = "log",
  link_bias = "logit")

Beta(link = "logit", link_phi = "log")

von_mises(link = "tan_half", link_kappa = "log")

asym_laplace(link = "identity", link_sigma = "log",
  link_quantile = "logit")

hurdle_poisson(link = "log")

hurdle_negbinomial(link = "log", link_shape = "log", link_hu = "logit")
```

```

hurdle_gamma(link = "log", link_shape = "log", link_hu = "logit")
hurdle_lognormal(link = "identity", link_sigma = "log", link_hu = "logit")
zero_inflated_beta(link = "logit", link_phi = "log", link_zi = "logit")
zero_one_inflated_beta(link = "logit", link_phi = "log",
  link_zoi = "logit", link_coi = "logit")
zero_inflated_poisson(link = "log", link_zi = "logit")
zero_inflated_negbinomial(link = "log", link_shape = "log",
  link_zi = "logit")
zero_inflated_binomial(link = "logit", link_zi = "logit")
categorical(link = "logit")
cumulative(link = "logit", link_disc = "log", threshold = c("flexible",
  "equidistant"))
sratio(link = "logit", link_disc = "log", threshold = c("flexible",
  "equidistant"))
cratio(link = "logit", link_disc = "log", threshold = c("flexible",
  "equidistant"))
acat(link = "logit", link_disc = "log", threshold = c("flexible",
  "equidistant"))

```

Arguments

family	A character string naming the distribution of the response variable be used in the model. Currently, the following families are supported: gaussian, student, binomial, bernoulli, poisson, negbinomial, geometric, Gamma, skew_normal, lognormal, shifted_lognormal, exgaussian, wiener, inverse.gaussian, exponential, weibull, frechet, Beta, von_mises, asym_laplace, gen_extreme_value, categorical, cumulative, cratio, sratio, acat, hurdle_poisson, hurdle_negbinomial, hurdle_gamma, hurdle_lognormal, zero_inflated_binomial, zero_inflated_beta, zero_inflated_negbinomial, zero_inflated_poisson, and zero_one_inflated_beta.
link	A specification for the model link function. This can be a name/expression or character string. See the 'Details' section for more information on link functions supported by each family.
link_sigma	Link of auxiliary parameter sigma if being predicted.
link_shape	Link of auxiliary parameter shape if being predicted.
link_nu	Link of auxiliary parameter nu if being predicted.

link_phi	Link of auxiliary parameter phi if being predicted.
link_kappa	Link of auxiliary parameter kappa if being predicted.
link_beta	Link of auxiliary parameter beta if being predicted.
link_zi	Link of auxiliary parameter zi if being predicted.
link_hu	Link of auxiliary parameter hu if being predicted.
link_zoi	Link of auxiliary parameter zoi if being predicted.
link_coi	Link of auxiliary parameter coi if being predicted.
link_disc	Link of auxiliary parameter disc if being predicted.
link_bs	Link of auxiliary parameter bs if being predicted.
link_ndt	Link of auxiliary parameter ndt if being predicted.
link_bias	Link of auxiliary parameter bias if being predicted.
link_xi	Link of auxiliary parameter xi if being predicted.
link_alpha	Link of auxiliary parameter alpha if being predicted.
link_quantile	Link of auxiliary parameter quantile if being predicted.
threshold	A character string indicating the type of thresholds (i.e. intercepts) used in an ordinal model. "flexible" provides the standard unstructured thresholds and "equidistant" restricts the distance between consecutive thresholds to the same value.

Details

Family gaussian with identity link leads to linear regression. Family student with identity link leads to robust linear regression that is less influenced by outliers. Family skew_normal can handle skewed responses in linear regression. Families poisson, negbinomial, and geometric with log link lead to regression models for count data. Families binomial and bernoulli with logit link leads to logistic regression and family categorical to multi-logistic regression when there are more than two possible outcomes. Families cumulative, cratio ('continuation ratio'), sratio ('stopping ratio'), and acat ('adjacent category') leads to ordinal regression. Families Gamma, weibull, exponential, lognormal, frechet, and inverse.gaussian can be used (among others) for survival regression. Families weibull, frechet, and gen_extreme_value ('generalized extreme value') allow for modeling extremes. Family asym_laplace allows for quantile regression when fixing the auxiliary quantile parameter to the quantile of interest. Family exgaussian ('exponentially modified Gaussian') and shifted_lognormal are especially suited to model reaction times. The wiener family provides an implementation of the Wiener diffusion model. For this family, the main formula predicts the drift parameter 'delta' and all other parameters are modeled as auxiliary parameters (see [brmsformula](#) for details). Families hurdle_poisson, hurdle_negbinomial, hurdle_gamma, hurdle_lognormal, zero_inflated_poisson, zero_inflated_negbinomial, zero_inflated_binomial, zero_inflated_beta, and zero_one_inflated_beta allow to estimate zero-inflated and hurdle models. These models can be very helpful when there are many zeros in the data (or ones in case of one-inflated models) that cannot be explained by the primary distribution of the response. Families hurdle_lognormal and hurdle_gamma are especially useful, as traditional lognormal or Gamma models cannot be reasonably fitted for data containing zeros in the response.

In the following, we list all possible links for each family. The families gaussian, student, skew_normal, exgaussian, asym_laplace, and gen_extreme_value accept the links (as names)

identity, log, and inverse; families poisson, negbinomial, geometric, zero_inflated_poisson, zero_inflated_negbinomial, hurdle_poisson, and hurdle_negbinomial the links log, identity, and sqrt; families binomial, bernoulli, Beta, zero_inflated_binomial, zero_inflated_beta, and zero_one_inflated_beta the links logit, probit, probit_approx, cloglog, cauchit, and identity; families cumulative, cratio, sratio, and acat the links logit, probit, probit_approx, cloglog, and cauchit; family categorical the link logit; families Gamma, weibull, exponential, frechet, and hurdle_gamma the links log, identity, and inverse; families lognormal and hurdle_lognormal the links identity and inverse; family inverse.gaussian the links $1/\mu^2$, inverse, identity and log; family von_mises the link tan_half; family wiener the link identity. The first link mentioned for each family is the default.

Please note that when calling the [Gamma](#) family function, the default link will be inverse not log. Also, the probit_approx link cannot be used when calling the [binomial](#) family function.

The current implementation of inverse.gaussian models has some convergence problems and requires carefully chosen prior distributions to work efficiently. For this reason, we currently do not recommend to use the inverse.gaussian family, unless you really feel that your data requires exactly this type of model.

See Also

[brm](#), [family](#), [customfamily](#)

Examples

```
# create a family object
(fam1 <- student("log"))
# alternatively use the brmsfamily function
(fam2 <- brmsfamily("student", "log"))
# both leads to the same object
identical(fam1, fam2)
```

brmsfit-class

*Class brmsfit of models fitted with the **brms** package*

Description

Models fitted with the [brms](#) package are represented as a brmsfit object, which contains the posterior samples, model formula, Stan code, relevant data, and other information.

Details

See `methods(class = "brmsfit")` for an overview of available methods.

Slots

formula A [brmsformula](#) object
family A [brmsfamily](#) object
data A data.frame containing all variables used in the model
data.name The name of data as specified by the user
model The model code in **Stan** language
prior A [brmsprior](#) object containing information on the priors used in the model
autocor An [cor_brms](#) object containing the autocorrelation structure if specified
ranef A data.frame containing the group-level structure
cov_ranef A list of customized group-level covariance matrices
stanvars A [stanvars](#) object or NULL
stan_funs A character string of length one or NULL
loo An empty slot for adding the [loo](#) information criterion after model fitting
waic An empty slot for adding the [waic](#) information criterion after model fitting
R2 An empty slot for adding the [bayes_R2](#) (Bayesian R-squared) value after model fitting
marglik An empty slot for adding a bridge object after model fitting containing the log marginal likelihood (see [bridge_sampler](#) for details)
fit An object of class [stanfit](#) among others containing the posterior samples
exclude The names of the parameters for which samples are not saved
algorithm The name of the algorithm used to fit the model
version The versions of **brms** and **rstan** with which the model was fitted

See Also

[brms](#), [brm](#), [brmsformula](#), [brmsfamily](#)

brmsformula

*Set up a model formula for use in **brms***

Description

Set up a model formula for use in the **brms** package allowing to define (potentially non-linear) additive multilevel models for all parameters of the assumed response distribution.

Usage

```
brmsformula(formula, ..., flist = NULL, family = NULL, autocor = NULL,
            nl = NULL)
```

Arguments

formula	An object of class formula (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given in 'Details'.
...	Additional formula objects to specify predictors of non-linear and distributional parameters. Formulas can either be named directly or contain names on their left-hand side. The following are distributional parameters of specific families (all other parameters are treated as non-linear parameters): sigma (residual standard deviation or scale of the gaussian, student, skew_normal, lognormal, exgaussian, and asym_laplace families); shape (shape parameter of the Gamma, weibull, negbinomial, and related zero-inflated / hurdle families); nu (degrees of freedom parameter of the student and frechet families); phi (precision parameter of the beta and zero_inflated_beta families); kappa (precision parameter of the von_mises family); beta (mean parameter of the exponential component of the exgaussian family); quantile (quantile parameter of the asym_laplace family); zi (zero-inflation probability); hu (hurdle probability); zoi (zero-one-inflation probability); coi (conditional one-inflation probability); disc (discrimination) for ordinal models; bs, ndt, and bias (boundary separation, non-decision time, and initial bias of the wiener diffusion model). By default, distributional parameters are modeled on the log scale if they can be positive only or on the logit scale if they can only be within the unit interval. See 'Details' for more explanation.
flist	Optional list of formulas, which are treated in the same way as formulas passed via the ... argument.
family	Same argument as in brm . If family is specified in brmsformula, it will overwrite the value specified in brm .
autocor	Same argument as in brm . If autocor is specified in brmsformula, it will overwrite the value specified in brm .
nl	Logical; Indicates whether formula should be treated as specifying a non-linear model. By default, formula is treated as an ordinary linear model formula.

Details**General formula structure**

The formula argument accepts formulas of the following syntax:

```
response | aterms ~ pterms + (gterms | group)
```

The pterms part contains effects that are assumed to be the same across observations. We call them 'population-level' effects or (adopting frequentist vocabulary) 'fixed' effects. The optional gterms part may contain effects that are assumed to vary across grouping variables specified in group. We call them 'group-level' effects or (adopting frequentist vocabulary) 'random' effects, although the latter name is misleading in a Bayesian context. For more details type `vignette("brms_overview")` and `vignette("brms_multilevel")`.

Group-level terms

Multiple grouping factors each with multiple group-level effects are possible. (Of course we can also run models without any group-level effects.) Instead of | you may use || in grouping terms to

prevent correlations from being modeled. Alternatively, it is possible to model different group-level terms of the same grouping factor as correlated (even across different formulas, e.g., in non-linear models) by using `|<ID>|` instead of `|`. All group-level terms sharing the same ID will be modeled as correlated. If, for instance, one specifies the terms $(1+x|2|g)$ and $(1+z|2|g)$ somewhere in the formulas passed to `brmsformula`, correlations between the corresponding group-level effects will be estimated.

If levels of the grouping factor belong to different sub-populations, it may be reasonable to assume a different covariance matrix for each of the sub-populations. For instance, the variation within the treatment group and within the control group in a randomized control trial might differ. Suppose that y is the outcome, and x is the factor indicating the treatment and control group. Then, we could estimate different hyper-parameters of the varying effects (in this case a varying intercept) for treatment and control group via $y \sim x + (1 | gr(\text{subject}, by = x))$.

You can specify multi-membership terms using the `mm` function. For instance, a multi-membership term with two members could be $(1 | mm(g1, g2))$, where $g1$ and $g2$ specify the first and second member, respectively. Moreover, if a covariate x varies across the levels of the grouping-factors $g1$ and $g2$, we can save the respective covariate values in the variables $x1$ and $x2$ and then model the varying effect as $(1 + mmc(x1, x2) | mm(g1, g2))$.

Special predictor terms

Smoothing terms can be modeled using the `s` and `t2` functions in the `pterms` part of the model formula. This allows to fit generalized additive mixed models (GAMMs) with **brms**. The implementation is similar to that used in the **gamm4** package. For more details on this model class see [gam](#) and [gamm](#).

Gaussian process terms can be fitted using the `gp` function in the `pterms` part of the model formula. Similar to smooth terms, Gaussian processes can be used to model complex non-linear relationships, for instance temporal or spatial autocorrelation. However, they are computationally demanding and are thus not recommended for very large datasets.

The `pterms` and `gterms` parts may contain four non-standard effect types namely monotonic, measurement error, missing value, and category specific effects, which can be specified using terms of the form `mo(predictor)`, `me(predictor, sd_predictor)`, `mi(predictor)`, and `cs(<predictors>)`, respectively. Category specific effects can only be estimated in ordinal models and are explained in more detail in the package's main vignette (`type vignette("brms_overview")`). The other three effect types are explained in the following.

A monotonic predictor must either be integer valued or an ordered factor, which is the first difference to an ordinary continuous predictor. More importantly, predictor categories (or integers) are not assumed to be equidistant with respect to their effect on the response variable. Instead, the distance between adjacent predictor categories (or integers) is estimated from the data and may vary across categories. This is realized by parameterizing as follows: One parameter takes care of the direction and size of the effect similar to an ordinary regression parameter, while an additional parameter vector estimates the normalized distances between consecutive predictor categories. A main application of monotonic effects are ordinal predictors that can this way be modeled without (falsely) treating them as continuous or as unordered categorical predictors. For more details and examples see `vignette("brms_monotonic")`.

Quite often, predictors are measured and as such naturally contain measurement error. Although most researchers are well aware of this problem, measurement error in predictors is ignored in most regression analyses, possibly because only few packages allow for modeling it. Notably, measurement error can be handled in structural equation models, but many more general regression models (such as those featured by **brms**) cannot be transferred to the SEM framework. In **brms**,

effects of noise-free predictors can be modeled using the `me` (for 'measurement error') function. If, say, y is the response variable and x is a measured predictor with known measurement error sd_x , we can simply include it on the right-hand side of the model formula via $y \sim me(x, sd_x)$. This can easily be extended to more general formulas. If x_2 is another measured predictor with corresponding error sd_{x_2} and z is a predictor without error (e.g., an experimental setting), we can model all main effects and interactions of the three predictors in the well known manner: $y \sim me(x, sd_x) * me(x_2, sd_{x_2}) * z$. In future version of **brms**, a vignette will be added to explain more details about these so called 'error-in-variables' models and provide real world examples.

When a variable contains missing values, the corresponding rows will be excluded from the data by default (row-wise exclusion). However, quite often we want to keep these rows and instead estimate the missing values. There are two approaches for this: (a) Impute missing values before the model fitting for instance via multiple imputation (see `brm_multiple` for a way to handle multiple imputed datasets). (b) Impute missing values on the fly during model fitting. The latter approach is explained in the following. Using a variable with missing values as predictors requires two things, First, we need to specify that the predictor contains missings that should to be imputed. If, say, y is the primary response, x is a predictor with missings and z is a predictor without missings, we go for $y \sim mi(x) + z$. Second, we need to model x as an additional response with corresponding predictors and the addition term `mi()`. In our example, we could write $x | mi() \sim z$. See `mi` for examples with real data.

Additional response information

Another special of the **brms** formula syntax is the optional `aterms` part, which may contain multiple terms of the form `fun(<variable>)` separated by `+` each providing special information on the response variable. `fun` can be replaced with either `se`, `weights`, `cens`, `trunc`, `trials`, `cat`, or `dec`. Their meanings are explained below. (see also [addition-terms](#)).

For families `gaussian`, `student` and `skew_normal`, it is possible to specify standard errors of the observations, thus allowing to perform meta-analysis. Suppose that the variable y_i contains the effect sizes from the studies and sei the corresponding standard errors. Then, fixed and random effects meta-analyses can be conducted using the formulas $y_i | se(sei) \sim 1$ and $y_i | se(sei) \sim 1 + (1|study)$, respectively, where `study` is a variable uniquely identifying every study. If desired, meta-regression can be performed via $y_i | se(sei) \sim 1 + mod1 + mod2 + (1|study)$ or $y_i | se(sei) \sim 1 + mod1 + mod2 + (1 + mod1 + mod2|study)$, where `mod1` and `mod2` represent moderator variables. By default, the standard errors replace the parameter `sigma`. To model `sigma` in addition to the known standard errors, set argument `sigma` in function `se` to `TRUE`, for instance, $y_i | se(sei, sigma = TRUE) \sim 1$.

For all families, weighted regression may be performed using `weights` in the `aterms` part. Internally, this is implemented by multiplying the log-posterior values of each observation by their corresponding weights. Suppose that variable w_{ei} contains the weights and that y_i is the response variable. Then, formula $y_i | weights(w_{ei}) \sim predictors$ implements a weighted regression.

With the exception of categorical, ordinal, and mixture families, left, right, and interval censoring can be modeled through $y | cens(censored) \sim predictors$. The censoring variable (named `censored` in this example) should contain the values 'left', 'none', 'right', and 'interval' (or equivalently -1, 0, 1, and 2) to indicate that the corresponding observation is left censored, not censored, right censored, or interval censored. For interval censored data, a second variable (let's call it y_2) has to be passed to `cens`. In this case, the formula has the structure $y | cens(censored, y_2) \sim predictors$.

While the lower bounds are given in y , the upper bounds are given in $y2$ for interval censored data. Intervals are assumed to be open on the left and closed on the right: $(y, y2]$.

With the exception of categorical, ordinal, and mixture families, the response distribution can be truncated using the `trunc` function in the addition part. If the response variable is truncated between, say, 0 and 100, we can specify this via `yi | trunc(lb = 0, ub = 100) ~ predictors`. Instead of numbers, variables in the data set can also be passed allowing for varying truncation points across observations. Defining only one of the two arguments in `trunc` leads to one-sided truncation.

For all continuous families, missing values in the responses can be imputed within Stan by using the addition term `mi`. This is mostly useful in combination with `mi` predictor terms as explained above under 'Special predictor terms'.

For families `binomial` and `zero_inflated_binomial`, addition should contain a variable indicating the number of trials underlying each observation. In `lme4` syntax, we may write for instance `cbind(success, n - success)`, which is equivalent to `success | trials(n)` in `brms` syntax. If the number of trials is constant across all observations, say 10, we may also write `success | trials(10)`.

For all ordinal families, `aterms` may contain a term `cat(number)` to specify the number categories (e.g, `cat(7)`). If not given, the number of categories is calculated from the data.

In Wiener diffusion models (family `wiener`) the addition term `dec` is mandatory to specify the (vector of) binary decisions corresponding to the reaction times. Non-zero values will be treated as a response on the upper boundary of the diffusion process and zeros will be treated as a response on the lower boundary. Alternatively, the variable passed to `dec` might also be a character vector consisting of 'lower' and 'upper'.

Multiple addition terms may be specified at the same time using the `+` operator, for instance `formula = yi | se(sei) + cens(censored) ~ 1` for a censored meta-analytic model.

The addition argument `disp` (short for dispersion) has been removed in version 2.0. You may instead use the distributional regression approach by specifying `sigma ~ 1 + offset(log(xdisp))` or `shape ~ 1 + offset(log(xdisp))`, where `xdisp` is the variable being previously passed to `disp`.

Parameterization of the population-level intercept

The population-level intercept (if incorporated) is estimated separately and not as part of population-level parameter vector `b`. As a result, priors on the intercept also have to be specified separately. Furthermore, to increase sampling efficiency, the population-level design matrix `X` is centered around its column means `X_means` if the intercept is incorporated. This leads to a temporary bias in the intercept equal to $\langle X_means, b \rangle$, where \langle, \rangle is the scalar product. The bias is corrected after fitting the model, but be aware that you are effectively defining a prior on the intercept of the centered design matrix not on the real intercept. For more details on setting priors on population-level intercepts, see [set_prior](#).

This behavior can be avoided by using the reserved (and internally generated) variable `intercept`. Instead of `y ~ x`, you may write `y ~ 0 + intercept + x`. This way, priors can be defined on the real intercept, directly. In addition, the intercept is just treated as an ordinary population-level effect and thus priors defined on `b` will also apply to it. Note that this parameterization may be less efficient than the default parameterization discussed above.

Formula syntax for non-linear models

In **brms**, it is possible to specify non-linear models of arbitrary complexity. The non-linear model can just be specified within the formula argument. Suppose, that we want to predict the response y through the predictor x , where x is linked to y through $y = \alpha - \beta * \lambda^x$, with parameters α , β , and λ . This is certainly a non-linear model being defined via formula $= y \sim \alpha - \beta * \lambda^x$ (addition arguments can be added in the same way as for ordinary formulas). To tell **brms** that this is a non-linear model, we set argument `n1` to `TRUE`. Now we have to specify a model for each of the non-linear parameters. Let's say we just want to estimate those three parameters with no further covariates or random effects. Then we can pass $\alpha + \beta + \lambda \sim 1$ or equivalently (and more flexible) $\alpha \sim 1$, $\beta \sim 1$, $\lambda \sim 1$ to the `...` argument. This can, of course, be extended. If we have another predictor z and observations nested within the grouping factor g , we may write for instance $\alpha \sim 1$, $\beta \sim 1 + z + (1|g)$, $\lambda \sim 1$. The formula syntax described above applies here as well. In this example, we are using z and g only for the prediction of β , but we might also use them for the other non-linear parameters (provided that the resulting model is still scientifically reasonable).

Non-linear models may not be uniquely identified and / or show bad convergence. For this reason it is mandatory to specify priors on the non-linear parameters. For instructions on how to do that, see [set_prior](#). For some examples of non-linear models, see `vignette("brms_nonlinear")`.

Formula syntax for predicting distributional parameters

It is also possible to predict parameters of the response distribution such as the residual standard deviation σ in gaussian models or the hurdle probability hu in hurdle models. The syntax closely resembles that of a non-linear parameter, for instance $\sigma \sim x + s(z) + (1+x|g)$. For some examples of distributional models, see `vignette("brms_distreg")`.

Alternatively, one may fix distributional parameters to certain values. However, this is mainly useful when models become too complicated and otherwise have convergence issues. We thus suggest to be generally careful when making use of this option. The `quantile` parameter of the `asym_laplace` distribution is a good example where it is useful. By fixing `quantile`, one can perform quantile regression for the specified quantile. For instance, `quantile = 0.25` allows predicting the 25%-quantile. Furthermore, the `bias` parameter in drift-diffusion models, is assumed to be `0.5` (i.e. no bias) in many applications. To achieve this, simply write `bias = 0.5`. Other possible applications are the Cauchy distribution as a special case of the Student-t distribution with `nu = 1`, or the geometric distribution as a special case of the negative binomial distribution with `shape = 1`. Furthermore, the parameter `disc` ('discrimination') in ordinal models is fixed to `1` by default and not estimated, but may be modeled as any other distributional parameter if desired (see examples). For reasons of identification, 'disc' can only be positive, which is achieved by applying the log-link.

In categorical models, distributional parameters do not have fixed names. Instead, they are named after the response categories (excluding the first one, which serves as the reference category), with the prefix 'mu'. If, for instance, categories are named `cat1`, `cat2`, and `cat3`, the distributional parameters will be named `mucat2` and `mucat3`.

Some distributional parameters currently supported by **brmsformula** have to be positive (a negative standard deviation or precision parameter does not make any sense) or are bounded between 0 and 1 (for zero-inflated / hurdle probabilities, quantiles, or the initial bias parameter of drift-diffusion models). However, linear predictors can be positive or negative, and thus the log link (for positive parameters) or logit link (for probability parameters) are used by default to ensure that distributional parameters are within their valid intervals. This implies that, by default, effects for such distributional parameters are estimated on the log / logit scale and one has to apply the inverse link function

to get to the effects on the original scale. Alternatively, it is possible to use the identity link to predict parameters on their original scale, directly. However, this is much more likely to lead to problems in the model fitting, if the parameter actually has a restricted range.

See also [brmsfamily](#) for an overview of valid link functions.

Formula syntax for mixture models

The specification of mixture models closely resembles that of non-mixture models. If not specified otherwise (see below), all mean parameters of the mixture components are predicted using the right-hand side of formula. All types of predictor terms allowed in non-mixture models are allowed in mixture models as well.

distributional parameters of mixture distributions have the same name as those of the corresponding ordinary distributions, but with a number at the end to indicate the mixture component. For instance, if you use family `mixture(gaussian, gaussian)`, the distributional parameters are `sigma1` and `sigma2`. distributional parameters of the same class can be fixed to the same value. For the above example, we could write `sigma2 = "sigma1"` to make sure that both components have the same residual standard deviation, which is in turn estimated from the data.

In addition, there are two types of special distributional parameters. The first are named `mu<ID>`, that allow for modeling different predictors for the mean parameters of different mixture components. For instance, if you want to predict the mean of the first component using predictor `x` and the mean of the second component using predictor `z`, you can write `mu1 ~ x` as well as `mu2 ~ z`. The second are named `theta<ID>`, which constitute the mixing proportions. If the mixing proportions are fixed to certain values, they are internally normalized to form a probability vector. If one seeks to predict the mixing proportions, all but one of the them has to be predicted, while the remaining one is used as the reference category to identify the model. The `softmax` function is applied on the linear predictor terms to form a probability vector.

For more information on mixture models, see the documentation of [mixture](#).

Formula syntax for multivariate models

Multivariate models may be specified using `cbind` notation or with help of the `mvbf` function. Suppose that `y1` and `y2` are response variables and `x` is a predictor. Then `cbind(y1, y2) ~ x` specifies a multivariate model. The effects of all terms specified at the RHS of the formula are assumed to vary across response variables. For instance, two parameters will be estimated for `x`, one for the effect on `y1` and another for the effect on `y2`. This is also true for group-level effects. When writing, for instance, `cbind(y1, y2) ~ x + (1+x|g)`, group-level effects will be estimated separately for each response. To model these effects as correlated across responses, use the ID syntax (see above). For the present example, this would look as follows: `cbind(y1, y2) ~ x + (1+x|2|g)`. Of course, you could also use any value other than 2 as ID.

It is also possible to specify different formulas for different responses. If, for instance, `y1` should be predicted by `x` and `y2` should be predicted by `z`, we could write `mvbf(y1 ~ x, y2 ~ z)`. Alternatively, multiple `brmsformula` objects can be added to specify a joint multivariate model (see 'Examples').

Value

An object of class `brmsformula`, which is essentially a list containing all model formulas as well as some additional information.

See Also

[mvbrmsformula](#), [brmsformula-helpers](#)

Examples

```
# multilevel model with smoothing terms
brmsformula(y ~ x1*x2 + s(z) + (1+x1|1) + (1|g2))

# additionally predict 'sigma'
brmsformula(y ~ x1*x2 + s(z) + (1+x1|1) + (1|g2),
            sigma ~ x1 + (1|g2))

# use the shorter alias 'bf'
(formula1 <- brmsformula(y ~ x + (x|g)))
(formula2 <- bf(y ~ x + (x|g)))
# will be TRUE
identical(formula1, formula2)

# incorporate censoring
bf(y | cens(censor_variable) ~ predictors)

# define a simple non-linear model
bf(y ~ a1 - a2^x, a1 + a2 ~ 1, nl = TRUE)

# predict a1 and a2 differently
bf(y ~ a1 - a2^x, a1 ~ 1, a2 ~ x + (x|g), nl = TRUE)

# correlated group-level effects across parameters
bf(y ~ a1 - a2^x, a1 ~ 1 + (1|2|g), a2 ~ x + (x|2|g), nl = TRUE)

# define a multivariate model
bf(cbind(y1, y2) ~ x * z + (1|g))

# define a zero-inflated model
# also predicting the zero-inflation part
bf(y ~ x * z + (1+x|ID1|g), zi ~ x + (1|ID1|g))

# specify a predictor as monotonic
bf(y ~ mo(x) + more_predictors)

# for ordinal models only
# specify a predictor as category specific
bf(y ~ cs(x) + more_predictors)
# add a category specific group-level intercept
bf(y ~ cs(x) + (cs(1)|g))
# specify parameter 'disc'
bf(y ~ person + item, disc ~ item)

# specify variables containing measurement error
bf(y ~ me(x, sdX))

# specify predictors on all parameters of the wiener diffusion model
```

```

# the main formula models the drift rate 'delta'
bf(rt | dec(decision) ~ x, bs ~ x, ndt ~ x, bias ~ x)

# fix the bias parameter to 0.5
bf(rt | dec(decision) ~ x, bias = 0.5)

# specify different predictors for different mixture components
mix <- mixture(gaussian, gaussian)
bf(y ~ 1, mu1 ~ x, mu2 ~ z, family = mix)

# fix both residual standard deviations to the same value
bf(y ~ x, sigma2 = "sigma1", family = mix)

# use the '+' operator to specify models
bf(y ~ 1) +
  nlf(sigma ~ a * exp(b * x), a ~ x) +
  lf(b ~ z + (1|g), dpar = "sigma") +
  gaussian()

# specify a multivariate model using the '+' operator
bf(y1 ~ x + (1|g)) +
  gaussian() + cor_ar(~1|g) +
  bf(y2 ~ z) + poisson()

# model missing values in predictors
bf(bmi ~ age * mi(chl)) +
  bf(chl | mi() ~ age) +
  set_rescor(FALSE)

```

brmsformula-helpers *Linear and Non-linear formulas in **brms***

Description

Helper functions to specify linear and non-linear formulas for use with [brmsformula](#).

Usage

```
nlf(formula, ..., flist = NULL, dpar = NULL, resp = NULL)
```

```
lf(..., flist = NULL, dpar = NULL, resp = NULL)
```

```
set_nl(nl = TRUE, dpar = NULL, resp = NULL)
```

```
set_rescor(rescor = TRUE)
```

```
set_mecor(mecor = TRUE)
```

Arguments

formula	Non-linear formula for a distributional parameter. The name of the distributional parameter can either be specified on the left-hand side of formula or via argument dpar.
...	Additional formula objects to specify predictors of non-linear and distributional parameters. Formulas can either be named directly or contain names on their left-hand side. The following are distributional parameters of specific families (all other parameters are treated as non-linear parameters): sigma (residual standard deviation or scale of the gaussian, student, skew_normal, lognormal, exgaussian, and asym_laplace families); shape (shape parameter of the Gamma, weibull, negbinomial, and related zero-inflated / hurdle families); nu (degrees of freedom parameter of the student and frechet families); phi (precision parameter of the beta and zero_inflated_beta families); kappa (precision parameter of the von_mises family); beta (mean parameter of the exponential component of the exgaussian family); quantile (quantile parameter of the asym_laplace family); zi (zero-inflation probability); hu (hurdle probability); zoi (zero-one-inflation probability); coi (conditional one-inflation probability); disc (discrimination) for ordinal models; bs, ndt, and bias (boundary separation, non-decision time, and initial bias of the wiener diffusion model). By default, distributional parameters are modeled on the log scale if they can be positive only or on the logit scale if they can only be within the unit interval. See 'Details' for more explanation.
flist	Optional list of formulas, which are treated in the same way as formulas passed via the ... argument.
dpar	Optional character string specifying the distributional parameter to which the formulas passed via ... and flist belong.
resp	Optional character string specifying the response variable to which the formulas passed via ... and flist belong. Only relevant in multivariate models.
nl	Logical; Indicates whether formula should be treated as specifying a non-linear model. By default, formula is treated as an ordinary linear model formula.
rescor	Logical; Indicates if residual correlation between the response variables should be modeled. Currently this is only possible in multivariate gaussian and student models. Only relevant in multivariate models.
mecor	Logical; Indicates if correlations between latent variables defined by <i>me</i> terms should be modeled. Defaults to TRUE.

Value

For *lf* and *nlf* a list that can be passed to [brmsformula](#) or added to an existing *brmsformula* or *mvbrmsformula* object. For *set_nl* and *set_rescor* a logical value that can be added to an existing *brmsformula* or *mvbrmsformula* object.

See Also

[brmsformula](#), [mvbrmsformula](#)

Examples

```
# add more formulas to the model
bf(y ~ 1) +
  nlf(sigma ~ a * exp(b * x), a ~ x) +
  lf(b ~ z + (1|g), dpar = "sigma") +
  gaussian()

# specify 'nl' later on
bf(y ~ a * inv_logit(x * b)) +
  lf(a + b ~ z) +
  set_nl(TRUE)

# specify a multivariate model
bf(y1 ~ x + (1|g)) +
  bf(y2 ~ z) +
  set_rescor(TRUE)
```

brmshypothesis

*Descriptions of brmshypothesis Objects***Description**

A brmshypothesis object contains posterior samples as well as summary statistics of non-linear hypotheses as returned by [hypothesis](#).

Usage

```
## S3 method for class 'brmshypothesis'
print(x, digits = 2, chars = 20, ...)

## S3 method for class 'brmshypothesis'
plot(x, N = 5, ignore_prior = FALSE, chars = 40,
     colors = NULL, theme = NULL, ask = TRUE, plot = TRUE, ...)
```

Arguments

x	An object of class brmsfit.
digits	Minimal number of significant digits, see print.default .
chars	Maximum number of characters of each hypothesis to print or plot. If NULL, print the full hypotheses. Defaults to 20.
...	Currently ignored.
N	The number of parameters plotted per page.
ignore_prior	A flag indicating if prior distributions should also be plotted. Only used if priors were specified on the relevant parameters.
colors	Two values specifying the colors of the posterior and prior density respectively. If NULL (the default) colors are taken from the current color scheme of the bayesplot package.

theme	A theme object modifying the appearance of the plots. For some basic themes see ggtheme and theme_default .
ask	logical; indicates if the user is prompted before a new page is plotted. Only used if <code>plot</code> is TRUE.
plot	logical; indicates if plots should be plotted directly in the active graphic device. Defaults to TRUE.

Details

The two most important elements of a `brmshypothesis` object are `hypothesis`, which is a `data.frame` containing the summary estimates of the hypotheses, and `samples`, which is a `data.frame` containing the corresponding posterior samples.

See Also

[hypothesis](#)

brm_multiple	<i>Run the same brms model on multiple datasets</i>
--------------	--

Description

Run the same **brms** model on multiple datasets and then combine the results into one fitted model object. This is useful in particular for multiple missing value imputation, where the same model is fitted on multiple imputed data sets.

Usage

```
brm_multiple(formula, data, combine = TRUE, ...)
```

Arguments

formula	An object of class formula , brmsformula , or mvbrmsformula (or one that can be coerced to that classes): A symbolic description of the model to be fitted. The details of model specification are explained in brmsformula .
data	A list of <code>data.frames</code> each of which will be used to fit a separate model. Alternatively, a <code>mids</code> object from the mice package.
combine	Logical; Indicates if the fitted models should be combined into a single fitted model object via combine_models . Defaults to TRUE.
...	Further arguments passed to brm .

Details

The combined model may issue false positive convergence warnings, as the MCMC chains corresponding to different datasets may not necessarily overlap, even if each of the original models did converge. To find out whether each of the original models converged, investigate `fit$rhats`, where `fit` denotes the output of `brm_multiple`.

Value

If `combine = TRUE` a `brmsfit_multiple` object, which inherits from class `brmsfit` and behaves essentially the same. If `combine = FALSE` a list of `brmsfit` objects.

Examples

```
## Not run:
library(mice)
imp <- mice(nhanes2)

# fit the model using mice and lm
fit_imp1 <- with(lm(bmi~age+hyp+chl), data = imp)
summary(pool(fit_imp1))

# fit the model using brms
fit_imp2 <- brm_multiple(bmi~age+hyp+chl, data = imp, chains = 1)
summary(fit_imp2)
plot(fit_imp2, pars = "^b_")
# investigate convergence of the original models
fit_imp2$rhats

## End(Not run)
```

coef.brmsfit

Extract Model Coefficients

Description

Extract model coefficients, which are the sum of population-level effects and corresponding group-level effects

Usage

```
## S3 method for class 'brmsfit'
coef(object, summary = TRUE, robust = FALSE,
      probs = c(0.025, 0.975), ...)
```

Arguments

<code>object</code>	An object of class <code>brmsfit</code>
<code>summary</code>	Should summary statistics (i.e. means, sds, and 95% intervals) be returned instead of the raw values? Default is <code>TRUE</code> .
<code>robust</code>	If <code>FALSE</code> (the default) the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If <code>TRUE</code> , the median and the median absolute deviation (MAD) are applied instead. Only used if <code>summary</code> is <code>TRUE</code> .

probs	The percentiles to be computed by the <code>quantile</code> function. Only used if <code>summary</code> is <code>TRUE</code> .
...	Currently ignored.

Value

If `old` is `FALSE`: A list of arrays (one per grouping factor). If `summary` is `TRUE`, names of the first dimension are the factor levels and names of the third dimension are the group-level effects. If `summary` is `FALSE`, names of the second dimension are the factor levels and names of the third dimension are the group-level effects.

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

Examples

```
## Not run:
fit <- brm(count ~ log_Age_c + log_Base4_c * Trt_c + (1+Trt_c|visit),
           data = epilepsy, family = gaussian(), chains = 2)
## extract population and group-level coefficients separately
fixef(fit)
ranef(fit)
## extract combined coefficients
coef(fit)

## End(Not run)
```

combine_models

*Combine Models fitted with **brms***

Description

Combine multiple `brmsfit` objects, which fitted the same model. This is useful for instance when having manually run models in parallel.

Usage

```
combine_models(..., mlist = NULL, check_data = TRUE)
```

Arguments

...	One or more <code>brmsfit</code> objects.
<code>mlist</code>	Optional list of one or more <code>brmsfit</code> objects.
<code>check_data</code>	Logical; indicates if the data should be checked for being the same across models (defaults to <code>TRUE</code>). Setting it to <code>FALSE</code> may be useful for instance when combining models fitted on multiple imputed data sets.

Details

This function just takes the first model and replaces its `stanfit` object (slot `fit`) by the combined `stanfit` objects of all models.

Value

A `brmsfit` object.

compare_ic

Compare Information Criteria of Different Models

Description

Compare information criteria of different models fitted with [WAIC](#) or [loo](#).

Usage

```
compare_ic(..., x = NULL, ic = c("loo", "waic", "kfold"))
```

Arguments

<code>...</code>	At least two objects returned by WAIC or loo . Alternatively, <code>brmsfit</code> objects with information criteria precomputed via add_ic may be passed, as well.
<code>x</code>	A list containing the same types of objects as can be passed via <code>...</code>
<code>ic</code>	The name of the information criterion to be extracted from <code>brmsfit</code> objects. Ignored if information criterion objects are only passed directly.

Details

For more details see [compare](#).

Value

An object of class `iclist`.

See Also

[WAIC](#), [loo](#), [add_ic](#), [compare](#)

Examples

```
## Not run:
# model with population-level effects only
fit1 <- brm(rating ~ treat + period + carry,
            data = inhaler, family = "gaussian")
waic1 <- WAIC(fit1)

# model with an additional varying intercept for subjects
fit2 <- brm(rating ~ treat + period + carry + (1|subject),
            data = inhaler, family = "gaussian")
waic2 <- WAIC(fit2)

# compare both models
compare_ic(waic1, waic2)

## End(Not run)
```

control_params.brmsfit

Extract Control Parameters of the NUTS Sampler

Description

Extract control parameters of the NUTS sampler such as `adapt_delta` or `max_treedepth`.

Usage

```
## S3 method for class 'brmsfit'
control_params(x, pars = NULL, ...)

control_params(x, ...)
```

Arguments

<code>x</code>	An R object
<code>pars</code>	Optional names of the control parameters to be returned. If <code>NULL</code> (the default) all control parameters are returned. See stan for more details.
<code>...</code>	Currently ignored.

Value

A named list with control parameter values.

cor_ar *AR(p) correlation structure*

Description

This function is a constructor for the `cor_arma` class, allowing for autoregression terms only.

Usage

```
cor_ar(formula = ~1, p = 1, cov = FALSE)
```

Arguments

formula	A one sided formula of the form $\sim t$, or $\sim t \mid g$, specifying a time covariate t and, optionally, a grouping factor g . A covariate for this correlation structure must be integer valued. When a grouping factor is present in <code>formula</code> , the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1 , which corresponds to using the order of the observations in the data as a covariate, and no groups.
p	A non-negative integer specifying the autoregressive (AR) order of the ARMA structure. Default is 1.
cov	A flag indicating whether ARMA effects should be estimated by means of residual covariance matrices (currently only possible for stationary ARMA effects of order 1). If <code>FALSE</code> (the default) a regression formulation is used that is considerably faster and allows for ARMA effects of order higher than 1 but cannot handle user defined standard errors.

Details

AR refers to autoregressive effects of residuals, which is what is typically understood as autoregressive effects. However, one may also model autoregressive effects of the response variable, which is called ARR in **brms**.

Value

An object of class `cor_arma` containing solely autoregression terms.

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

See Also

[cor_arma](#)

Examples

```
cor_ar(~visit|patient, p = 2)
```

cor_arma	<i>ARMA(p,q) correlation structure</i>
----------	--

Description

This function is a constructor for the `cor_arma` class, representing an autoregression-moving average correlation structure of order (p, q).

Usage

```
cor_arma(formula = ~1, p = 0, q = 0, r = 0, cov = FALSE)
```

Arguments

formula	A one sided formula of the form $\sim t$, or $\sim t g$, specifying a time covariate t and, optionally, a grouping factor g . A covariate for this correlation structure must be integer valued. When a grouping factor is present in <code>formula</code> , the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1 , which corresponds to using the order of the observations in the data as a covariate, and no groups.
p	A non-negative integer specifying the autoregressive (AR) order of the ARMA structure. Default is 0.
q	A non-negative integer specifying the moving average (MA) order of the ARMA structure. Default is 0.
r	A non-negative integer specifying the autoregressive response (ARR) order. See 'Details' for differences of AR and ARR effects. Default is 0.
cov	A flag indicating whether ARMA effects should be estimated by means of residual covariance matrices (currently only possible for stationary ARMA effects of order 1). If FALSE (the default) a regression formulation is used that is considerably faster and allows for ARMA effects of order higher than 1 but cannot handle user defined standard errors.

Details

AR refers to autoregressive effects of residuals, which is what is typically understood as autoregressive effects. However, one may also model autoregressive effects of the response variable, which is called ARR in **brms**.

Value

An object of class `cor_arma`, representing an autoregression-moving-average correlation structure.

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

See Also

[cor_ar](#), [cor_ma](#), [cor_arr](#)

Examples

```
cor_arma(~visit|patient, p = 2, q = 2)
```

<code>cor_arr</code>	<i>ARR(r) correlation structure</i>
----------------------	-------------------------------------

Description

This function is a constructor for the `cor_arma` class allowing for autoregressive effects of the response only.

Usage

```
cor_arr(formula = ~1, r = 1)
```

Arguments

<code>formula</code>	A one sided formula of the form $\sim t$, or $\sim t g$, specifying a time covariate t and, optionally, a grouping factor g . A covariate for this correlation structure must be integer valued. When a grouping factor is present in <code>formula</code> , the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1 , which corresponds to using the order of the observations in the data as a covariate, and no groups.
<code>r</code>	A non-negative integer specifying the autoregressive response (ARR) order of the ARMA structure. Default is 1.

Details

AR refers to autoregressive effects of residuals, which is what is typically understood as autoregressive effects. However, one may also model autoregressive effects of the response variable, which is called ARR in **brms**.

Value

An object of class `cor_arma` containing solely autoregressive response terms.

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

See Also

[cor_arma](#)

Examples

```
cor_arr(~visit|patient, r = 2)
```

cor_brms

*Correlation structure classes for the **brms** package*

Description

Classes of correlation structures available in the **brms** package. `cor_brms` is not a correlation structure itself, but the class common to all correlation structures implemented in **brms**.

Available correlation structures

cor_arma autoregressive-moving average (ARMA) structure, with arbitrary orders for the autoregressive and moving average components

cor_ar autoregressive (AR) structure of arbitrary order

cor_ma moving average (MA) structure of arbitrary order

cor_arr response autoregressive (ARR) structure

cor_car Spatial conditional autoregressive (CAR) structure

cor_sar Spatial simultaneous autoregressive (SAR) structure

cor_bsts Bayesian structural time series (BSTS) structure

cor_fixed fixed user-defined covariance structure

See Also

[cor_arma](#), [cor_ar](#), [cor_ma](#), [cor_arr](#), [cor_car](#), [cor_sar](#), [cor_bsts](#), [cor_fixed](#)

cor_bsts

*Basic Bayesian Structural Time Series***Description**

Add a basic Bayesian structural time series component to a brms model

Usage

```
cor_bsts(formula = ~1)
```

Arguments

formula	A one sided formula of the form $\sim t$, or $\sim t \mid g$, specifying a time covariate t and, optionally, a grouping factor g . A covariate for this correlation structure must be integer valued. When a grouping factor is present in formula, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1 , which corresponds to using the order of the observations in the data as a covariate, and no groups.
---------	---

Details

Bayesian structural time series models offer an alternative to classical AR(I)MA models (they are in fact a generalization). The basic version currently implemented in **brms** introduces local level terms for each observation, whereas each local level term depends on the former local level term:

$$LL_t \sim N(LL_{t-1}, \text{sigma}LL)$$

A simple introduction can be found in this blogpost: <http://multithreaded.stitchfix.com/blog/2016/04/21/forget-arma/>. More complicated Bayesian structural time series models may follow in the future.

Value

An object of class cor_bsts.

Examples

```
## Not run:
dat <- data.frame(y = rnorm(100), x = rnorm(100))
fit <- brm(y~x, data = dat, autocor = cor_bsts())

## End(Not run)
```

cor_car *Spatial conditional autoregressive (CAR) structures*

Description

These functions are constructors for the `cor_car` class implementing spatial conditional autoregressive structures.

Usage

```
cor_car(W, formula = ~1, type = c("escar", "esicar", "icar"))
cor_icar(W, formula = ~1)
```

Arguments

<code>W</code>	Adjacency matrix of locations. All non-zero entries are treated as if the two locations are adjacent. If <code>formula</code> contains a grouping factor, the row names of <code>W</code> have to match the levels of the grouping factor.
<code>formula</code>	An optional one-sided formula of the form $\sim 1 \mid g$, where <code>g</code> is a grouping factor mapping observations to spatial locations. If not specified, each observation is treated as a separate location. It is recommended to always specify a grouping factor to allow for handling of new data in post-processing methods.
<code>type</code>	Type of the CAR structure. Currently implemented are "escar" (exact sparse CAR) and "esicar" (exact sparse intrinsic CAR) and "icar" (intrinsic CAR). More information is provided in the 'Details' section.

Details

The `escar` and `esicar` types are implemented based on the case study of Max Joseph (<https://github.com/mbjoseph/CARstan>). The `icar` type is implemented based on the case study of Mitzi Morris (http://mc-stan.org/users/documentation/case-studies/icar_stan.html).

Examples

```
## Not run:
# generate some spatial data
east <- north <- 1:10
Grid <- expand.grid(east, north)
K <- nrow(Grid)

# set up distance and neighbourhood matrices
distance <- as.matrix(dist(Grid))
W <- array(0, c(K, K))
W[distance == 1] <- 1

# generate the covariates and response data
x1 <- rnorm(K)
```

```

x2 <- rnorm(K)
theta <- rnorm(K, sd = 0.05)
phi <- rmulti_normal(
  1, mu = rep(0, K), Sigma = 0.4 * exp(-0.1 * distance)
)
eta <- x1 + x2 + phi
prob <- exp(eta) / (1 + exp(eta))
size <- rep(50, K)
y <- rbinom(n = K, size = size, prob = prob)
dat <- data.frame(y, size, x1, x2)

# fit a CAR model
fit <- brm(y | trials(size) ~ x1 + x2, data = dat,
  family = binomial(), autocor = cor_car(W))
summary(fit)

## End(Not run)

```

cor_fixed

Fixed user-defined covariance matrices

Description

Define a fixed covariance matrix of the response variable for instance to model multivariate effect sizes in meta-analysis.

Usage

```
cor_fixed(V)
```

Arguments

V Known covariance matrix of the response variable. If a vector is passed, it will be used as diagonal entries (variances) and covariances will be set to zero.

Value

An object of class cor_fixed.

Examples

```

## Not run:
dat <- data.frame(y = rnorm(3))
V <- cbind(c(0.5, 0.3, 0.2), c(0.3, 1, 0.1), c(0.2, 0.1, 0.2))
fit <- brm(y~1, data = dat, autocor = cor_fixed(V))

## End(Not run)

```

cor_ma	<i>MA(q) correlation structure</i>
--------	------------------------------------

Description

This function is a constructor for the `cor_arma` class, allowing for moving average terms only.

Usage

```
cor_ma(formula = ~1, q = 1, cov = FALSE)
```

Arguments

formula	A one sided formula of the form $\sim t$, or $\sim t g$, specifying a time covariate t and, optionally, a grouping factor g . A covariate for this correlation structure must be integer valued. When a grouping factor is present in <code>formula</code> , the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1 , which corresponds to using the order of the observations in the data as a covariate, and no groups.
q	A non-negative integer specifying the moving average (MA) order of the ARMA structure. Default is 1.
cov	A flag indicating whether ARMA effects should be estimated by means of residual covariance matrices (currently only possible for stationary ARMA effects of order 1). If FALSE (the default) a regression formulation is used that is considerably faster and allows for ARMA effects of order higher than 1 but cannot handle user defined standard errors.

Value

An object of class `cor_arma` containing solely moving average terms.

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

See Also

[cor_arma](#)

Examples

```
cor_ma(~visit|patient, q = 2)
```

cor_sar

*Spatial simultaneous autoregressive (SAR) structures***Description**

These functions are constructors for the cor_sar class implementing spatial simultaneous autoregressive structures. The lagsar structure implements SAR of the response values:

$$y = \rho W y + \eta + e$$

The errorsar structure implements SAR of the residuals:

$$y = \eta + u, u = \rho W u + e$$

In the above equations, η is the predictor term and e are independent normally or t-distributed residuals.

Usage

```
cor_sar(W, type = c("lag", "error"))
```

```
cor_lagsar(W)
```

```
cor_errorsar(W)
```

Arguments

W	An object specifying the spatial weighting matrix. Can be either the spatial weight matrix itself or an object of class listw or nb, from which the spatial weighting matrix can be computed.
type	Type of the SAR structure. Either "lag" (for SAR of the response values) or "error" (for SAR of the residuals).

Details

Currently, only families gaussian and student support SAR structures.

Value

An object of class cor_sar to be used in calls to [brm](#).

Examples

```
## Not run:
data(oldcol, package = "spdep")
fit1 <- brm(CRIME ~ INC + HOVAL, data = COL.OLD,
           autocor = cor_lagsar(COL.nb),
           chains = 2, cores = 2)
summary(fit1)
```

```

plot(fit1)

fit2 <- brm(CRIME ~ INC + HOVAL, data = COL.OLD,
            autocor = cor_errorsar(COL.nb),
            chains = 2, cores = 2)
summary(fit2)
plot(fit2)

## End(Not run)

```

Description

Category Specific Predictors in **brms** Models

Usage

```
cs(expr)
```

Arguments

`expr` Expression containing predictors, for which category specific effects should be estimated. For evaluation, R formula syntax is applied.

Details

For detailed documentation see `help(brmsformula)` as well as `vignette("brms_overview")`.

This function is almost solely useful when called in formulas passed to the **brms** package.

See Also

[brmsformula](#)

Examples

```

## Not run:
fit <- brm(rating ~ period + carry + cs(treat),
          data = inhaler, family = sratio("cloglog"),
          prior = set_prior("normal(0,5)"), chains = 2)
summary(fit)
plot(fit, ask = FALSE)

## End(Not run)

```

Description

Define custom families for use in **brms** models. It allows users to benefit from the modeling flexibility of **brms**, while applying their self-defined likelihood functions. All of the post-processing methods for `brmsfit` objects can be made compatible with custom families. See `vignette("brms_customfamilies")` for more details. For a list of built-in families see [brmsfamily](#).

Usage

```
custom_family(name, dpars = "mu", links = "identity", type = c("real",
  "int"), lb = NA, ub = NA, vars = NULL, env = parent.frame())
```

Arguments

name	Name of the custom family.
dpars	Names of the distributional parameters of the family. One parameter must be named "mu" and the main formula of the model will correspond to that parameter.
links	Names of the link functions of the distributional parameters.
type	Indicates if the response distribution is continuous ("real") or discrete ("int").
lb	Vector of lower bounds of the distributional parameters. Defaults to NA that is no lower bound.
ub	Vector of upper bounds of the distributional parameters. Defaults to NA that is no upper bound.
vars	Names of variables, which are part of the likelihood function without being distributional parameters. That is, <code>vars</code> can be used to pass data to the likelihood. See stanvar for details about adding self-defined data to the generated Stan model.
env	An environment in which certain post-processing functions related to the custom family can be found. This is only relevant if one wants to ensure compatibility with the methods predict , fitted , or log_lik .

Details

The corresponding probability density or mass Stan functions need to have the same name as the custom family. That is if a family is called `myfamily`, then the **Stan** functions should be called `myfamily_lpdf` or `myfamily_lpmf` depending on whether it defines a continuous or discrete distribution.

Value

An object of class `customfamily` inheriting for class [brmsfamily](#).

See Also

[brmsfamily](#), [stanvar](#)

Examples

```
## Not run:
## demonstrate how to fit a beta-binomial model
## generate some fake data
phi <- 0.7
n <- 300
z <- rnorm(n, sd = 0.2)
ntrials <- sample(1:10, n, replace = TRUE)
eta <- 1 + z
mu <- exp(eta) / (1 + exp(eta))
a <- mu * phi
b <- (1 - mu) * phi
p <- rbeta(n, a, b)
y <- rbinom(n, ntrials, p)
dat <- data.frame(y, z, ntrials)

# define a custom family
beta_binomial2 <- custom_family(
  "beta_binomial2", dpars = c("mu", "phi"),
  links = c("logit", "log"), lb = c(NA, 0),
  type = "int", vars = "trials[n]"
)

# define the corresponding Stan density function
stan_funs <- "
  real beta_binomial2_lpmf(int y, real mu, real phi, int N) {
    return beta_binomial_lpmf(y | N, mu * phi, (1 - mu) * phi);
  }
"

# fit the model
fit <- brm(y | trials(ntrials) ~ z, data = dat,
  family = beta_binomial2, stan_funs = stan_funs)
summary(fit)

## End(Not run)
```

epilepsy

Epileptic seizure counts

Description

Breslow and Clayton (1993) analyze data initially provided by Thall and Vail (1990) concerning seizure counts in a randomized trial of anti-convulsant therapy in epilepsy. Covariates are treatment, 8-week baseline seizure counts, and age of the patients in years.

Usage

epilepsy

Format

A dataframe of 236 observations containing information on the following 9 variables.

Age The age of the patients in years

Base The seizure count at 8-weeks baseline

Trt Either 0 or 1 indicating if the patient received anti-convulsant therapy

log_Age_c The logarithm of Age centered around its mean

log_Base4_c The logarithm of Base divided by 4 (i.e. $\log(\text{Base}/4)$) centered around its mean

Trt_c Trt centered around its mean

visit The session number from 1 (first visit) to 4 (last visit)

count The seizure count between two visits

patient The patient number

obs The observation number, i.e. a unique identifier for each observation

Source

Thall, P. F., & Vail, S. C. (1990). Some covariance models for longitudinal count data with overdispersion. *Biometrics*, 46(2), 657-671.

Breslow, N. E., & Clayton, D. G. (1993). Approximate inference in generalized linear mixed models. *Journal of the American Statistical Association*, 88(421), 9-25.

Examples

```
## Not run:
## poisson regression without random effects.
fit1 <- brm(count ~ log_Age_c + log_Base4_c * Trt_c,
            data = epilepsy, family = poisson())
summary(fit1)
plot(fit1)

## poisson regression with random intercepts of patients and visits
## as well as normal priors for fixed effects parameters.
fit2 <- brm(count ~ log_Age_c + log_Base4_c * Trt_c
            + (1|patient) + (1|visit),
            data = epilepsy, family = poisson(),
            prior = set_prior("normal(0,5)")
            )
summary(fit2)
plot(fit2)

## End(Not run)
```

 ExGaussian

The Exponentially Modified Gaussian Distribution

Description

Density, distribution function, and random generation for the exponentially modified Gaussian distribution with mean μ and standard deviation σ of the gaussian component, as well as scale β of the exponential component.

Usage

```
dexgaussian(x, mu, sigma, beta, log = FALSE)
```

```
pexgaussian(q, mu, sigma, beta, lower.tail = TRUE, log.p = FALSE)
```

```
rexgaussian(n, mu, sigma, beta)
```

Arguments

<code>x, q</code>	Vector of quantiles.
<code>mu</code>	Vector of means of the combined distribution.
<code>sigma</code>	Vector of standard deviations of the gaussian component.
<code>beta</code>	Vector of scales of the exponential component.
<code>log</code>	Logical; If TRUE, values are returned on the log scale.
<code>lower.tail</code>	Logical; If TRUE (default), return $P(X \leq x)$. Else, return $P(X > x)$.
<code>log.p</code>	Logical; If TRUE, values are returned on the log scale.
<code>n</code>	Number of samples to draw from the distribution.

Details

See `vignette("brms_families")` for details on the parameterization.

 expose_functions.brmsfit

Expose user-defined Stan functions

Description

Export user-defined **Stan** function and optionally vectorize them. For more details see [expose_stan_functions](#).

Usage

```
## S3 method for class 'brmsfit'
expose_functions(x, vectorize = FALSE, env = globalenv(),
  ...)

expose_functions(x, ...)
```

Arguments

x	An R object
vectorize	Logical; Indicates if the exposed functions should be vectorized via Vectorize . Defaults to FALSE.
env	Environment where the functions should be made available. Defaults to the global environment.
...	Further arguments passed to expose_stan_functions .

exp1	<i>Exponential function plus one.</i>
------	---------------------------------------

Description

Computes $\exp(x) + 1$.

Usage

```
exp1(x)
```

Arguments

x	A numeric or complex vector.
---	------------------------------

fitted.brmsfit	<i>Extract Model Fitted Values of brmsfit Objects</i>
----------------	---

Description

Predict fitted values (i.e., the 'regression line') of a fitted model. Can be performed for the data used to fit the model (posterior predictive checks) or for new data. By definition, these predictions have smaller variance than the response predictions performed by the [predict](#) method. This is because the measurement error is not incorporated. The estimated means of both methods should, however, be very similar.

Usage

```
## S3 method for class 'brmsfit'
fitted(object, newdata = NULL, re_formula = NULL,
  scale = c("response", "linear"), allow_new_levels = FALSE,
  sample_new_levels = "uncertainty", new_objects = list(),
  incl_autocor = TRUE, dpar = NULL, resp = NULL, subset = NULL,
  nsamples = NULL, sort = FALSE, nug = NULL, summary = TRUE,
  robust = FALSE, probs = c(0.025, 0.975), ...)

## S3 method for class 'brmsfit'
posterior_linpred(object, transform = FALSE,
  newdata = NULL, re_formula = NULL, re.form = NULL,
  allow_new_levels = FALSE, sample_new_levels = "uncertainty",
  new_objects = list(), incl_autocor = TRUE, dpar = NULL, resp = NULL,
  subset = NULL, nsamples = NULL, sort = FALSE, nug = NULL,
  robust = FALSE, probs = c(0.025, 0.975), ...)
```

Arguments

object	An object of class <code>brmsfit</code>
newdata	An optional <code>data.frame</code> for which to evaluate predictions. If <code>NULL</code> (default), the original data of the model is used.
re_formula	formula containing group-level effects to be considered in the prediction. If <code>NULL</code> (default), include all group-level effects; if <code>NA</code> , include no group-level effects.
scale	Either <code>"response"</code> or <code>"linear"</code> . If <code>"response"</code> results are returned on the scale of the response variable. If <code>"linear"</code> fitted values are returned on the scale of the linear predictor.
allow_new_levels	A flag indicating if new levels of group-level effects are allowed (defaults to <code>FALSE</code>). Only relevant if <code>newdata</code> is provided.
sample_new_levels	Indicates how to sample new levels for grouping factors specified in <code>re_formula</code> . This argument is only relevant if <code>newdata</code> is provided and <code>allow_new_levels</code> is set to <code>TRUE</code> . If <code>"uncertainty"</code> (default), include group-level uncertainty in the predictions based on the variation of the existing levels. If <code>"gaussian"</code> , sample new levels from the (multivariate) normal distribution implied by the group-level standard deviations and correlations. This options may be useful for conducting Bayesian power analysis. If <code>"old_levels"</code> , directly sample new levels from the existing levels.
new_objects	A named list of objects containing new data, which cannot be passed via argument <code>newdata</code> . Currently, only required for objects passed to <code>cor_sar</code> and <code>cor_fixed</code> .
incl_autocor	A flag indicating if ARMA autocorrelation parameters should be included in the predictions. Defaults to <code>TRUE</code> . Setting it to <code>FALSE</code> will not affect other correlation structures such as <code>cor_bsts</code> , or <code>cor_fixed</code> .

dpar	Optional name of a predicted distributional parameter. If specified, fitted values of this parameters are returned.
resp	Optional names of response variables. If specified, fitted values of these response variables are returned.
subset	A numeric vector specifying the posterior samples to be used. If NULL (the default), all samples are used.
nsamples	Positive integer indicating how many posterior samples should be used. If NULL (the default) all samples are used. Ignored if subset is not NULL.
sort	Logical. Only relevant for time series models. Indicating whether to return predicted values in the original order (FALSE; default) or in the order of the time series (TRUE).
nug	Small positive number for Gaussian process terms only. For numerical reasons, the covariance matrix of a Gaussian process might not be positive definite. Adding a very small number to the matrix's diagonal often solves this problem. If NULL (the default), nug is chosen internally.
summary	Should summary statistics (i.e. means, sds, and 95% intervals) be returned instead of the raw values? Default is TRUE.
robust	If FALSE (the default) the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If TRUE, the median and the median absolute deviation (MAD) are applied instead. Only used if summary is TRUE.
probs	The percentiles to be computed by the quantile function. Only used if summary is TRUE.
...	Currently ignored.
transform	Logical; alias of scale. If TRUE, scale is set to "response". If FALSE, scale is set to "linear". Only implemented for compatibility with the posterior_linpred generic.
re.form	Alias of re_formula.

Details

NA values within factors in newdata, are interpreted as if all dummy variables of this factor are zero. This allows, for instance, to make predictions of the grand mean when using sum coding.

Method `posterior_linpred.brmsfit` is an alias of `fitted.brmsfit` with `scale = "linear"` and `summary = FALSE`.

Value

Fitted values extracted from object. The output depends on the family: If `summary = TRUE` it is a $N \times E \times C$ array for categorical and ordinal models and a $N \times E$ matrix else. If `summary = FALSE` it is a $S \times N \times C$ array for categorical and ordinal models and a $S \times N$ matrix else. N is the number of observations, S is the number of samples, C is the number of categories, and E is equal to `length(probs) + 2`. In multivariate models, the output is an array of 3 dimensions, where the third dimension indicates the response variables.

Examples

```
## Not run:
## fit a model
fit <- brm(rating ~ treat + period + carry + (1|subject),
           data = inhaler)

## extract fitted values
fitted_values <- fitted(fit)
head(fitted_values)

## plot fitted means against actual response
dat <- as.data.frame(cbind(Y = standata(fit)$Y, fitted_values))
ggplot(dat) + geom_point(aes(x = Estimate, y = Y))

## End(Not run)
```

fixef.brmsfit

Extract Population-Level Estimates

Description

Extract the population-level ('fixed') effects from a `brmsfit` object.

Usage

```
## S3 method for class 'brmsfit'
fixef(object, summary = TRUE, robust = FALSE,
       probs = c(0.025, 0.975), ...)
```

Arguments

<code>object</code>	An object of class <code>brmsfit</code> .
<code>summary</code>	Should summary statistics (i.e. means, sds, and 95% intervals) be returned instead of the raw values? Default is <code>TRUE</code> .
<code>robust</code>	If <code>FALSE</code> (the default) the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If <code>TRUE</code> , the median and the median absolute deviation (MAD) are applied instead. Only used if <code>summary</code> is <code>TRUE</code> .
<code>probs</code>	The percentiles to be computed by the <code>quantile</code> function. Only used if <code>summary</code> is <code>TRUE</code> .
<code>...</code>	Currently ignored.

Value

If `summary` is `TRUE`, a matrix with one row per population-level effect and one column per calculated estimate. If `summary` is `FALSE`, a matrix with one row per posterior sample and one column per population-level effect.

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

Examples

```
## Not run:
fit <- brm(time | cens(censored) ~ age + sex + disease,
           data = kidney, family = "exponential")
fixef(fit)

## End(Not run)
```

Frechet

The Frechet Distribution

Description

Density, distribution function, quantile function and random generation for the Frechet distribution with location `loc`, scale `scale`, and shape `shape`.

Usage

```
dfrechet(x, loc = 0, scale = 1, shape = 1, log = FALSE)

pfrechet(q, loc = 0, scale = 1, shape = 1, lower.tail = TRUE,
         log.p = FALSE)

qfrechet(p, loc = 0, scale = 1, shape = 1, lower.tail = TRUE,
         log.p = FALSE)

rfrechet(n, loc = 0, scale = 1, shape = 1)
```

Arguments

<code>x, q</code>	Vector of quantiles.
<code>loc</code>	Vector of locations.
<code>scale</code>	Vector of scales.
<code>shape</code>	Vector of shapes.
<code>log</code>	Logical; If TRUE, values are returned on the log scale.
<code>lower.tail</code>	Logical; If TRUE (default), return $P(X \leq x)$. Else, return $P(X > x)$.
<code>log.p</code>	Logical; If TRUE, values are returned on the log scale.
<code>p</code>	Vector of probabilities.
<code>n</code>	Number of samples to draw from the distribution.

Details

See `vignette("brms_families")` for details on the parameterization.

GenExtremeValue *The Generalized Extreme Value Distribution*

Description

Density, distribution function, and random generation for the generalized extreme value distribution with location μ , scale σ and shape ξ .

Usage

```
dgen_extreme_value(x, mu = 0, sigma = 1, xi = 0, log = FALSE)

pgen_extreme_value(q, mu = 0, sigma = 1, xi = 0, lower.tail = TRUE,
  log.p = FALSE)

rgen_extreme_value(n, mu = 0, sigma = 1, xi = 0)
```

Arguments

<code>x</code> , <code>q</code>	Vector of quantiles.
<code>mu</code>	Vector of locations.
<code>sigma</code>	Vector of scales.
<code>xi</code>	Vector of shapes.
<code>log</code>	Logical; If TRUE, values are returned on the log scale.
<code>lower.tail</code>	Logical; If TRUE (default), return $P(X \leq x)$. Else, return $P(X > x)$.
<code>log.p</code>	Logical; If TRUE, values are returned on the log scale.
<code>n</code>	Number of samples to draw from the distribution.

Details

See `vignette("brms_families")` for details on the parameterization.

 get_prior

 Overview on Priors for **brms** Models

Description

Get information on all parameters (and parameter classes) for which priors may be specified including default priors.

Usage

```
get_prior(formula, data, family = gaussian(), autocor = NULL,
          internal = FALSE)
```

Arguments

formula	An object of class formula , brmsformula , or mvbrmsformula (or one that can be coerced to that classes): A symbolic description of the model to be fitted. The details of model specification are explained in brmsformula .
data	An object of class <code>data.frame</code> (or one that can be coerced to that class) containing data of all variables used in the model.
family	A description of the response distribution and link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a <code>link</code> argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details of supported families see brmsfamily . By default, a linear gaussian model is applied. In multivariate models, family might also be a list of families.
autocor	An optional cor_brms object describing the correlation structure within the response variable (i.e., the 'autocorrelation'). See the documentation of cor_brms for a description of the available correlation structures. Defaults to <code>NULL</code> , corresponding to no correlations. In multivariate models, autocor might also be a list of autocorrelation structures.
internal	A flag indicating if the names of additional internal parameters should be displayed. Setting priors on these parameters is not recommended

Value

A `data.frame` with columns `prior`, `class`, `coef`, and `group` and several rows, each providing information on a parameter (or parameter class) on which priors can be specified. The `prior` column is empty except for internal default priors.

See Also

[set_prior](#)

Examples

```
## get all parameters and parameters classes to define priors on
(prior <- get_prior(count ~ log_Age_c + log_Base4_c * Trt_c
  + (1|patient) + (1|visit),
  data = epilepsy, family = poisson()))

## define a prior on all population-level effects a once
prior$prior[1] <- "normal(0,10)"

## define a specific prior on the population-level effect of Trt_c
prior$prior[5] <- "student_t(10, 0, 5)"

## verify that the priors indeed found their way into Stan's model code
make_stancode(count ~ log_Age_c + log_Base4_c * Trt_c
  + (1|patient) + (1|visit),
  data = epilepsy, family = poisson(),
  prior = prior)
```

gp

*Set up Gaussian process terms in **brms***

Description

Function used to set up a Gaussian process term in **brms**. The function does not evaluate its arguments – it exists purely to help set up a model with Gaussian process terms.

Usage

```
gp(..., by = NA, cov = "exp_quad", scale = TRUE)
```

Arguments

...	One or more predictors for the Gaussian process.
by	A numeric or factor variable of the same length as each predictor. In the numeric vector case, the elements multiply the values returned by the Gaussian process. In the factor variable case, a separate Gaussian process is fitted for each factor level.
cov	Name of the covariance kernel. By default, the exponentiated-quadratic kernel "exp_quad" is used.
scale	Logical; If TRUE (the default), predictors are scaled so that the maximum Euclidean distance between two points is 1. Since the default prior on <code>lscale</code> expects scaled predictors, it is recommended to manually specify priors on <code>lscale</code> , if <code>scale</code> is set to FALSE.

Details

A Gaussian process is a stochastic process, which describes the relation between one or more predictors $x = (x_1, \dots, x_d)$ and a response $f(x)$, where d is the number of predictors. A Gaussian process is the generalization of the multivariate normal distribution to an infinite number of dimensions. Thus, it can be interpreted as a prior over functions. Any finite sample realized from this stochastic process is jointly multivariate normal, with a covariance matrix defined by the covariance kernel $k_p(x)$, where p is the vector of parameters of the Gaussian process:

$$f(x) \text{ MVN}(0, k_p(x))$$

The smoothness and general behavior of the function f depends only on the choice of covariance kernel. For a more detailed introduction to Gaussian processes, see https://en.wikipedia.org/wiki/Gaussian_process.

Below, we describe the currently supported covariance kernels:

- "exp_quad": The exponentiated-quadratic kernel is defined as $k(x_i, x_j) = sdgp^2 \exp(-\|x_i - x_j\|/(2lscale^2))$, where $\|\cdot\|$ is the Euclidean norm, $sdgp$ is a standard deviation parameter, and $lscale$ is characteristic length-scale parameter. The latter practically measures how close two points x_i and x_j have to be to influence each other substantially.

In the current implementation, "exp_quad" is the only supported covariance kernel. More options will follow in the future.

Value

An object of class 'gpterm', which is a list of arguments to be interpreted by the formula parsing functions of brms.

See Also

[brmsformula](#)

Examples

```
## Not run:
# simulate data using the mgcv package
dat <- mgcv::gamSim(1, n = 30, scale = 2)

# fit a simple gaussian process model
fit1 <- brm(y ~ gp(x2), dat, chains = 2)
summary(fit1)
me1 <- marginal_effects(fit1, nsamples = 200, spaghetti = TRUE)
plot(me1, ask = FALSE, points = TRUE)

# fit a more complicated gaussian process model
fit2 <- brm(y ~ gp(x0) + x1 + gp(x2) + x3, dat, chains = 2)
summary(fit2)
me2 <- marginal_effects(fit2, nsamples = 200, spaghetti = TRUE)
plot(me2, ask = FALSE, points = TRUE)

# fit a multivariate gaussian process model
```



```

fit3 <- brm(y ~ gp(x1, x2), dat, chains = 2)
summary(fit3)
me3 <- marginal_effects(fit3, nsamples = 200, spaghetti = TRUE)
plot(me3, ask = FALSE, points = TRUE)

# compare model fit
LOO(fit1, fit2, fit3)

# simulate data with a factor covariate
dat2 <- mgcv::gamSim(4, n = 90, scale = 2)

# fit separate gaussian processes for different levels of 'fac'
fit4 <- brm(y ~ gp(x2, by = fac), dat2, chains = 2)
summary(fit4)
plot(marginal_effects(fit4), points = TRUE)

## End(Not run)

```

gr

*Set up basic grouping terms in **brms***

Description

Function used to set up a basic grouping term in **brms**. The function does not evaluate its arguments – it exists purely to help set up a model with grouping terms. `gr` is called implicitly inside the package and there is usually no need to call it directly.

Usage

```
gr(..., by = NULL, dist = "gaussian")
```

Arguments

<code>...</code>	One or more terms containing grouping factors.
<code>by</code>	An optional factor variable, specifying sub-populations of the groups. For each level of the <code>by</code> variable, a separate variance-covariance matrix will be fitted. Levels of the grouping factor must be nested in levels of the <code>by</code> variable.
<code>dist</code>	Name of the distribution of the group-level effects. Currently "gaussian" is the only option.

See Also

[brmsformula](#)

Examples

```
## Not run:
# model using basic lme4-style formula
fit1 <- brm(count ~ Trt + (1|patient), data = epilepsy)
summary(fit1)

# equivalent model using 'gr' which is called anyway internally
fit2 <- brm(count ~ Trt + (1|gr(patient)), data = epilepsy)
summary(fit2)

# include Trt as a by variable
fit3 <- brm(count ~ Trt + (1|gr(patient, by = Trt)), data = epilepsy)
summary(fit3)

## End(Not run)
```

horseshoe

*Set up a horseshoe prior in **brms***

Description

Function used to set up a horseshoe prior for population-level effects in **brms**. The function does not evaluate its arguments – it exists purely to help set up the model.

Usage

```
horseshoe(df = 1, scale_global = 1, df_global = 1, scale_slab = 2,
          df_slab = 4, par_ratio = NULL, autoscale = TRUE)
```

Arguments

<code>df</code>	Degrees of freedom of student-t prior of the local shrinkage parameters. Defaults to 1.
<code>scale_global</code>	Scale of the student-t prior of the global shrinkage parameter. Defaults to 1. In linear models, <code>scale_global</code> will internally be multiplied by the residual standard deviation parameter <code>sigma</code> .
<code>df_global</code>	Degrees of freedom of student-t prior of the global shrinkage parameter. Defaults to 1.
<code>scale_slab</code>	Scale of the student-t prior of the regularization parameter. Defaults to 2.
<code>df_slab</code>	Degrees of freedom of the student-t prior of the regularization parameter. Defaults to 4.
<code>par_ratio</code>	Ratio of the expected number of non-zero coefficients to the expected number of zero coefficients. If specified, <code>scale_global</code> is ignored and internally computed as $\text{par_ratio} / \sqrt{N}$, where N is the total number of observations in the data.

`autoscale` Logical; indicating whether the horseshoe prior should be scaled using the residual standard deviation `sigma` if possible and sensible (defaults to TRUE). Autoscaling is not applied for distributional parameters or when the model does not contain the parameter `sigma`.

Details

The horseshoe prior is a special shrinkage prior initially proposed by Carvalho et al. (2009). It is symmetric around zero with fat tails and an infinitely large spike at zero. This makes it ideal for sparse models that have many regression coefficients, although only a minority of them is non-zero. The horseshoe prior can be applied on all population-level effects at once (excluding the intercept) by using `set_prior("horseshoe(1)")`. The 1 implies that the student-t prior of the local shrinkage parameters has 1 degrees of freedom. This may, however, lead to an increased number of divergent transition in **Stan**. Accordingly, increasing the degrees of freedom to slightly higher values (e.g., 3) may often be a better option, although the prior no longer resembles a horseshoe in this case. Further, the scale of the global shrinkage parameter plays an important role in amount of shrinkage applied. It defaults to 1, but this may result in too few shrinkage (Piironen & Vehtari, 2016). It is thus possible to change the scale using argument `scale_global` of the horseshoe prior, for instance `horseshoe(1, scale_global = 0.5)`. In linear models, `scale_global` will internally be multiplied by the residual standard deviation parameter `sigma`. See Piironen and Vehtari (2016) for recommendations how to properly set the global scale. The degrees of freedom of the global shrinkage prior may also be adjusted via argument `df_global`. Piironen and Vehtari (2017) recommend to specifying the ratio of the expected number of non-zero coefficients to the expected number of zero coefficients `par_ratio` rather than `scale_global` directly. As proposed by Piironen and Vehtari (2017), an additional regularization is applied that only affects non-zero coefficients. The amount of regularization can be controlled via `scale_slab` and `df_slab`. To make sure that shrinkage can equally affect all coefficients, predictors should be on the same scale. Generally, models with horseshoe priors are more likely than other models to have divergent transitions so that increasing `adapt_delta` from 0.8 to values closer to 1 will often be necessary. See the documentation of [brm](#) for instructions on how to increase `adapt_delta`.

Value

A character string obtained by `match.call()` with additional arguments.

References

- Carvalho, C. M., Polson, N. G., & Scott, J. G. (2009). Handling sparsity via the horseshoe. In International Conference on Artificial Intelligence and Statistics (pp. 73-80).
- Piironen J. & Vehtari A. (2016). On the Hyperprior Choice for the Global Shrinkage Parameter in the Horseshoe Prior. <https://arxiv.org/pdf/1610.05559v1.pdf>
- Piironen, J., and Vehtari, A. (2017). Sparsity information and regularization in the horseshoe and other shrinkage priors. <https://arxiv.org/abs/1707.01694>

See Also

[set_prior](#)

Examples

```
set_prior(horseshoe(df = 3, par_ratio = 0.1))
```

hypothesis.brmsfit *Non-Linear Hypothesis Testing*

Description

Perform non-linear hypothesis testing for all model parameters.

Usage

```
## S3 method for class 'brmsfit'
hypothesis(x, hypothesis, class = "b", group = "",
  scope = c("standard", "ranef", "coef"), alpha = 0.05, seed = NULL, ...)

hypothesis(x, ...)

## Default S3 method:
hypothesis(x, hypothesis, alpha = 0.05, ...)
```

Arguments

x	An R object. If it is no brmsfit object, it must be coercible to a data frame.
hypothesis	A character vector specifying one or more non-linear hypothesis concerning parameters of the model.
class	A string specifying the class of parameters being tested. Default is "b" for population-level effects. Other typical options are "sd" or "cor". If class = NULL, all parameters can be tested against each other, but have to be specified with their full name (see also parnames)
group	Name of a grouping factor to evaluate only group-level effects parameters related to this grouping factor. Ignored if class is not "sd" or "cor".
scope	Indicates where to look for the variables specified in hypothesis. If "standard", use the full parameter names (subject to the restriction given by class). If "coef" or "ranef" compute the hypothesis for all levels of the grouping factor given in "group", based on the output of coef.brmsfit and ranef.brmsfit , respectively.
alpha	The alpha-level of the tests (default is 0.05; see 'Details' for more information).
seed	A single numeric value passed to set.seed to make results reproducible.
...	Currently ignored.

Details

Among others, hypothesis computes an evidence ratio (Evid.Ratio) for each hypothesis. For a directed hypothesis, this is just the posterior probability under the hypothesis against its alternative. That is, when the hypothesis is of the form $a > b$, the evidence ratio is the ratio of the posterior probability of $a > b$ and the posterior probability of $a < b$. In this example, values greater than one indicate that the evidence in favor of $a > b$ is larger than evidence in favor of $a < b$. For an undirected (point) hypothesis, the evidence ratio is a Bayes factor between the hypothesis and its alternative computed via the Savage-Dickey density ratio method. That is the posterior density at the point of interest divided by the prior density at that point. Values greater than one indicate that evidence in favor of the point hypothesis has increased after seeing the data. In order to calculate this Bayes factor, all parameters related to the hypothesis must have proper priors and argument `sample_prior` of function `brm` must be set to `TRUE`. When interpreting Bayes factors, make sure that your priors are reasonable and carefully chosen, as the result will depend heavily on the priors. In particular, avoid using default priors.

The argument `alpha` specifies the size of the credible interval (i.e., Bayesian confidence interval). For instance, if `alpha = 0.05` (5%), the credible interval will contain $1 - \alpha = 0.95$ (95%) of the posterior values. Hence, $\alpha * 100\%$ of the posterior values will lie outside of the credible interval. Although this allows testing of hypotheses in a similar manner as in the frequentist null-hypothesis testing framework, we strongly argue against using arbitrary cutoffs (e.g., $p < .05$) to determine the 'existence' of an effect.

Value

A `brmshypothesis` object.

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

See Also

[brmshypothesis](#)

Examples

```
## Not run:
## define priors
prior <- c(set_prior("normal(0,2)", class = "b"),
          set_prior("student_t(10,0,1)", class = "sigma"),
          set_prior("student_t(10,0,1)", class = "sd"))

## fit a linear mixed effects models
fit <- brm(time ~ age + sex + disease + (1 + age|patient),
          data = kidney, family = lognormal(),
          prior = prior, sample_prior = TRUE,
          control = list(adapt_delta = 0.95))

## perform two-sided hypothesis testing
(hyp1 <- hypothesis(fit, "sexfemale = age + diseasePKD"))
plot(hyp1)
```

```

hypothesis(fit, "exp(age) - 3 = 0", alpha = 0.01)

## perform one-sided hypothesis testing
hypothesis(fit, "diseasePKD + diseaseGN - 3 < 0")

hypothesis(fit, "age < Intercept",
           class = "sd", group = "patient")

## test the amount of random intercept variance on all variance
h <- paste("sd_patient__Intercept^2 / (sd_patient__Intercept^2 +",
          "sd_patient__age^2 + sigma^2) = 0")
(hyp2 <- hypothesis(fit, h, class = NULL))
plot(hyp2)

## test more than one hypothesis at once
h <- c("diseaseGN = diseaseAN", "2 * diseaseGN - diseasePKD = 0")
(hyp3 <- hypothesis(fit, h))
plot(hyp3, ignore_prior = TRUE)

## compute hypotheses for all levels of a grouping factor
hypothesis(fit, "age = 0", scope = "coef", group = "patient")

## use the default method
dat <- as.data.frame(fit)
hypothesis(dat, "b_age > 0")

## End(Not run)

```

inhaler

Clarity of inhaler instructions

Description

Ezzet and Whitehead (1991) analyze data from a two-treatment, two-period crossover trial to compare 2 inhalation devices for delivering the drug salbutamol in 286 asthma patients. Patients were asked to rate the clarity of leaflet instructions accompanying each device, using a 4-point ordinal scale.

Usage

```
inhaler
```

Format

A dataframe of 572 observations containing information on the following 5 variables.

subject The subject number

rating The rating of the inhaler instructions on a scale ranging from 1 to 4

- treat** A contrast to indicate which of the two inhaler devices was used
- period** A contrast to indicate the time of administration
- carry** A contrast to indicate possible carry over effects

Source

Ezzet, F., & Whitehead, J. (1991). A random effects model for ordinal responses from a crossover trial. *Statistics in Medicine*, 10(6), 901-907.

Examples

```
## Not run:
## ordinal regression with family "sratio"
fit1 <- brm(rating ~ treat + period + carry,
           data = inhaler, family = sratio(),
           prior = set_prior("normal(0,5)"))
summary(fit1)
plot(fit1)

## ordinal regression with family "cumulative"
## and random intercept over subjects
fit2 <- brm(rating ~ treat + period + carry + (1|subject),
           data = inhaler, family = cumulative(),
           prior = set_prior("normal(0,5)"))
summary(fit2)
plot(fit2)

## End(Not run)
```

Description

Density, distribution function, and random generation for the inverse Gaussian distribution with location μ , and shape shape .

Usage

```
dinv_gaussian(x, mu = 1, shape = 1, log = FALSE)

pinv_gaussian(q, mu = 1, shape = 1, lower.tail = TRUE, log.p = FALSE)

rinv_gaussian(n, mu = 1, shape = 1)
```

Arguments

<code>x, q</code>	Vector of quantiles.
<code>mu</code>	Vector of locations.
<code>shape</code>	Vector of shapes.
<code>log</code>	Logical; If TRUE, values are returned on the log scale.
<code>lower.tail</code>	Logical; If TRUE (default), return $P(X \leq x)$. Else, return $P(X > x)$.
<code>log.p</code>	Logical; If TRUE, values are returned on the log scale.
<code>n</code>	Number of samples to draw from the distribution.

Details

See `vignette("brms_families")` for details on the parameterization.

<code>inv_logit_scaled</code>	<i>Scaled inverse logit-link</i>
-------------------------------	----------------------------------

Description

Computes $\text{inv_logit}(x) * (ub - lb) + lb$

Usage

```
inv_logit_scaled(x, lb = 0, ub = 1)
```

Arguments

<code>x</code>	A numeric or complex vector.
<code>lb</code>	Lower bound defaulting to 0.
<code>ub</code>	Upper bound defaulting to 1.

Value

A numeric or complex vector between `lb` and `ub`.

is.brmsfit	<i>Checks if argument is a brmsfit object</i>
------------	---

Description

Checks if argument is a brmsfit object

Usage

```
is.brmsfit(x)
```

Arguments

x	An R object
---	-------------

is.brmsformula	<i>Checks if argument is a brmsformula object</i>
----------------	---

Description

Checks if argument is a brmsformula object

Usage

```
is.brmsformula(x)
```

Arguments

x	An R object
---	-------------

is.brmsprior	<i>Checks if argument is a brmsprior object</i>
--------------	---

Description

Checks if argument is a brmsprior object

Usage

```
is.brmsprior(x)
```

Arguments

x	An R object
---	-------------

is.brmsterms	<i>Checks if argument is a brmsterms object</i>
--------------	---

Description

Checks if argument is a brmsterms object

Usage

```
is.brmsterms(x)
```

Arguments

x	An R object
---	-------------

See Also

[parse_bf](#)

is.cor_brms	<i>Check if argument is a correlation structure</i>
-------------	---

Description

Check if argument is one of the correlation structures used in **brms**.

Usage

```
is.cor_brms(x)
```

```
is.cor_arma(x)
```

```
is.cor_sar(x)
```

```
is.cor_car(x)
```

```
is.cor_fixed(x)
```

```
is.cor_bsts(x)
```

Arguments

x	An R object.
---	--------------

is.mvbrmsformula *Checks if argument is a mvbrmsformula object*

Description

Checks if argument is a mvbrmsformula object

Usage

is.mvbrmsformula(x)

Arguments

x An R object

is.mvbrmsterms *Checks if argument is a mvbrmsterms object*

Description

Checks if argument is a mvbrmsterms object

Usage

is.mvbrmsterms(x)

Arguments

x An R object

See Also

[parse_bf](#)

kfold.brmsfit

*K-Fold Cross-Validation***Description**

Perform exact K -fold cross-validation by refitting the model K times each leaving out one- K th of the original data.

Usage

```
## S3 method for class 'brmsfit'
kfold(x, ..., compare = TRUE, K = 10, Ksub = NULL,
      exact_loo = FALSE, group = NULL, resp = NULL, model_names = NULL,
      save_fits = FALSE)

kfold(x, ...)
```

Arguments

x	A fitted model object.
...	More fitted model objects or further arguments passed to the underlying post-processing functions.
compare	A flag indicating if the information criteria of the models should be compared to each other via compare_ic .
K	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.
Ksub	Optional number of subsets (of those subsets defined by K) to be evaluated. If NULL (the default), K -fold cross-validation will be performed on all subsets. If $Ksub$ is a single integer, $Ksub$ subsets (out of all K) subsets will be randomly chosen. If $Ksub$ consists of multiple integers, the corresponding subsets will be used. This argument is primarily useful, if evaluation of all subsets is infeasible for some reason.
exact_loo	Logical; If TRUE, exact leave-one-out cross-validation will be performed and K will be ignored. This argument alters the way argument <code>group</code> is handled as described below. Defaults to FALSE.
group	Optional name of a grouping variable or factor in the model. How this variable is handled depends on argument <code>exact_loo</code> . If <code>exact_loo</code> is FALSE, the data is split up into subsets, each time omitting all observations of one of the factor levels, while ignoring argument K . If <code>exact_loo</code> is TRUE, all observations corresponding to the factor level of the currently predicted single value are omitted. Thus, in this case, the predicted values are only a subset of the omitted ones.

resp	Optional names of response variables. If specified, fitted values of these response variables are returned.
model_names	If NULL (the default) will use model names derived from parsing the call. Otherwise will use the passed values as model names.
save_fits	If TRUE, a component fits is added to the returned object to store the cross-validated brmsfit objects and the indices of the omitted observations for each fold. Defaults to FALSE.

Details

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_ic function is also compatible with the objects returned by kfold.

Value

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

Methods (by class)

- brmsfit: kfold method for brmsfit objects

See Also

[loo](#), [reloo](#)

Examples

```
## Not run:
fit1 <- brm(count ~ log_Age_c + log_Base4_c * Trt +
            (1|patient) + (1|obs),
            data = epilepsy, family = poisson())
# throws warning about some pareto k estimates being too high
(loo1 <- loo(fit1))
# perform 10-fold cross validation
(kfold1 <- kfold(fit1, chains = 2, cores = 2))

## End(Not run)
```

kidney

*Infections in kidney patients***Description**

This dataset, originally discussed in McGilchrist and Aisbett (1991), describes the first and second (possibly right censored) recurrence time of infection in kidney patients using portable dialysis equipment. In addition, information on the risk variables age, sex and disease type is provided.

Usage

kidney

Format

A dataframe of 76 observations containing information on the following 7 variables.

time The time to first or second recurrence of the infection, or the time of censoring

recur A factor of levels 1 or 2 indicating if the infection recurred for the first or second time for this patient

censored Either 0 or 1, where 0 indicates no censoring of recurrence time and 1 indicates right censoring

patient The patient number

age The age of the patient

sex The sex of the patient

disease A factor of levels other, GN, AN, and PKD specifying the type of disease

Source

McGilchrist, C. A., & Aisbett, C. W. (1991). Regression with frailty in survival analysis. *Biometrics*, 47(2), 461-466.

Examples

```
## Not run:
## performing survival analysis using the "weibull" family
fit1 <- brm(time | cens(censored) ~ age + sex + disease,
            data = kidney, family = weibull, inits = "0")
summary(fit1)
plot(fit1)

## adding random intercepts over patients
fit2 <- brm(time | cens(censored) ~ age + sex + disease + (1|patient),
            data = kidney, family = weibull(), inits = "0",
            prior = set_prior("cauchy(0,2)", class = "sd"))
summary(fit2)
plot(fit2)
```

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

lasso *Set up a lasso prior in brms*

Description

Function used to set up a lasso prior for population-level effects in **brms**. The function does not evaluate its arguments – it exists purely to help set up the model.

Usage

```
lasso(df = 1, scale = 1)
```

Arguments

df	Degrees of freedom of the chi-square prior of the inverse tuning parameter. Defaults to 1.
scale	Scale of the lasso prior. Defaults to 1.

Details

The lasso prior is the Bayesian equivalent to the LASSO method for performing variable selection (Park & Casella, 2008). With this prior, independent Laplace (i.e. double exponential) priors are placed on the population-level effects. The scale of the Laplace priors depends on a tuning parameter that controls the amount of shrinkage. In **brms**, the inverse of the tuning parameter is used so that smaller values imply more shrinkage. The inverse tuning parameter has a chi-square distribution and with degrees of freedom controlled via argument *df* of function `lasso` (defaults to 1). For instance, one can specify a lasso prior using `set_prior("lasso(1)")`. To make sure that shrinkage can equally affect all coefficients, predictors should be on the same scale. If you do not want to standardize all variables, you can adjust the general scale of the lasso prior via argument *scale*, for instance, `lasso(1, scale = 10)`.

Value

A character string obtained by `match.call()` with additional arguments.

References

Park, T., & Casella, G. (2008). The Bayesian Lasso. *Journal of the American Statistical Association*, 103(482), 681-686.

See Also

[set_prior](#)

Examples

```
set_prior(lasso(df = 1, scale = 10))
```

launch_shinystan.brmsfit

Interface to shinystan

Description

Provide an interface to **shinystan** for models fitted with **brms**

Usage

```
## S3 method for class 'brmsfit'  
launch_shinystan(object,  
  rstudio = getOption("shinystan.rstudio"), ...)
```

Arguments

object	A fitted model object typically of class <code>brmsfit</code> .
rstudio	Only relevant for RStudio users. The default (<code>rstudio=FALSE</code>) is to launch the app in the default web browser rather than RStudio's pop-up Viewer. Users can change the default to <code>TRUE</code> by setting the global option <code>options(shinystan.rstudio = TRUE)</code> .
...	Optional arguments to pass to runApp

Value

An S4 shinystan object

See Also

[launch_shinystan](#)

Examples

```
## Not run:  
fit <- brm(rating ~ treat + period + carry + (1|subject),  
  data = inhaler, family = "gaussian")  
launch_shinystan(fit)  
  
## End(Not run)
```

logit_scaled	<i>Scaled logit-link</i>
--------------	--------------------------

Description

Computes $\text{logit}((x - lb) / (ub - lb))$

Usage

```
logit_scaled(x, lb = 0, ub = 1)
```

Arguments

x	A numeric or complex vector.
lb	Lower bound defaulting to 0.
ub	Upper bound defaulting to 1.

Value

A numeric or complex vector.

logm1	<i>Logarithm with a minus one offset.</i>
-------	---

Description

Computes $\log(x - 1)$.

Usage

```
logm1(x, base = exp(1))
```

Arguments

x	A numeric or complex vector.
base	A positive or complex number: the base with respect to which logarithms are computed. Defaults to $e = \exp(1)$.

log_lik.brmsfit *Compute the Pointwise Log-Likelihood*

Description

Compute the Pointwise Log-Likelihood

Usage

```
## S3 method for class 'brmsfit'
log_lik(object, newdata = NULL, re_formula = NULL,
        allow_new_levels = FALSE, sample_new_levels = "uncertainty",
        new_objects = list(), incl_autocor = TRUE, resp = NULL, subset = NULL,
        nsamples = NULL, pointwise = FALSE, nug = NULL, combine = TRUE, ...)
```

Arguments

object	A fitted model object of class <code>brmsfit</code> .
newdata	An optional <code>data.frame</code> for which to evaluate predictions. If <code>NULL</code> (default), the original data of the model is used.
re_formula	formula containing group-level effects to be considered in the prediction. If <code>NULL</code> (default), include all group-level effects; if <code>NA</code> , include no group-level effects.
allow_new_levels	A flag indicating if new levels of group-level effects are allowed (defaults to <code>FALSE</code>). Only relevant if <code>newdata</code> is provided.
sample_new_levels	Indicates how to sample new levels for grouping factors specified in <code>re_formula</code> . This argument is only relevant if <code>newdata</code> is provided and <code>allow_new_levels</code> is set to <code>TRUE</code> . If <code>"uncertainty"</code> (default), include group-level uncertainty in the predictions based on the variation of the existing levels. If <code>"gaussian"</code> , sample new levels from the (multivariate) normal distribution implied by the group-level standard deviations and correlations. This options may be useful for conducting Bayesian power analysis. If <code>"old_levels"</code> , directly sample new levels from the existing levels.
new_objects	A named list of objects containing new data, which cannot be passed via argument <code>newdata</code> . Currently, only required for objects passed to <code>cor_sar</code> and <code>cor_fixed</code> .
incl_autocor	A flag indicating if ARMA autocorrelation parameters should be included in the predictions. Defaults to <code>TRUE</code> . Setting it to <code>FALSE</code> will not affect other correlation structures such as <code>cor_bsts</code> , or <code>cor_fixed</code> .
resp	Optional names of response variables. If specified, fitted values of these response variables are returned.
subset	A numeric vector specifying the posterior samples to be used. If <code>NULL</code> (the default), all samples are used.

nsamples	Positive integer indicating how many posterior samples should be used. If NULL (the default) all samples are used. Ignored if subset is not NULL.
pointwise	A flag indicating whether to compute the full log-likelihood matrix at once (the default), or just return the likelihood function along with all data and samples required to compute the log-likelihood separately for each observation. The latter option is rarely useful when calling log_lik directly, but rather when computing waic or loo.
nug	Small positive number for Gaussian process terms only. For numerical reasons, the covariance matrix of a Gaussian process might not be positive definite. Adding a very small number to the matrix's diagonal often solves this problem. If NULL (the default), nug is chosen internally.
combine	Only relevant in multivariate models. Indicates if the log-likelihoods of the sub-models should be combined per observation (i.e. added together; the default) or if the log-likelihoods should be returned separately.
...	Currently ignored

Value

Usually, an S x N matrix containing the pointwise log-likelihood samples, where S is the number of samples and N is the number of observations in the data. For multivariate models and if combine is FALSE, an S x N x R array is returned, where R is the number of response variables. If pointwise = TRUE, the output is a function with a draws attribute containing all relevant data and posterior samples.

log_posterior.brmsfit *Extract Diagnostic Quantities of brms Models*

Description

Extract quantities that can be used to diagnose sampling behavior of the algorithms applied by **Stan** at the back-end of **brms**.

Usage

```
## S3 method for class 'brmsfit'
log_posterior(object, ...)

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

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

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

Arguments

object	A brmsfit object.
...	Arguments passed to individual methods.
pars	An optional character vector of parameter names. For nuts_params these will be NUTS sampler parameter names rather than model parameters. If pars is omitted all parameters are included.

Details

For more details see [bayesplot-extractors](#).

Value

The exact form of the output depends on the method.

Examples

```
## Not run:
fit <- brm(time ~ age * sex, data = kidney)

lp <- log_posterior(fit)
head(lp)

np <- nuts_params(fit)
str(np)
# extract the number of divergence transitions
sum(subset(np, Parameter == "divergent_")$Value)

head(rhat(fit))
head(neff_ratio(fit))

## End(Not run)
```

 loo.brmsfit

Compute the LOO information criterion

Description

Perform approximate leave-one-out cross-validation based on the posterior likelihood using the **loo** package. For more details see [loo](#).

Usage

```
## S3 method for class 'brmsfit'
loo(x, ..., compare = TRUE, resp = NULL,
    pointwise = FALSE, reloo = FALSE, k_threshold = 0.7,
    model_names = NULL)
```

Arguments

<code>x</code>	A fitted model object.
<code>...</code>	More fitted model objects or further arguments passed to the underlying post-processing functions.
<code>compare</code>	A flag indicating if the information criteria of the models should be compared to each other via <code>compare_ic</code> .
<code>resp</code>	Optional names of response variables. If specified, fitted values of these response variables are returned.
<code>pointwise</code>	A flag indicating whether to compute the full log-likelihood matrix at once or separately for each observation. The latter approach is usually considerably slower but requires much less working memory. Accordingly, if one runs into memory issues, <code>pointwise = TRUE</code> is the way to go.
<code>reloo</code>	Logical; Indicate whether <code>reloo</code> should be applied on problematic observations. Defaults to <code>FALSE</code> .
<code>k_threshold</code>	The threshold at which pareto k estimates are treated as problematic. Defaults to <code>0.7</code> . Only used if argument <code>reloo</code> is <code>TRUE</code> . See <code>pareto_k_ids</code> for more details.
<code>model_names</code>	If <code>NULL</code> (the default) will use model names derived from parsing the call. Otherwise will use the passed values as model names.

Details

When comparing models fitted to the same data, the smaller the LOO, the better the fit. For `brmsfit` objects, `LOO` is an alias of `loo`. Use method `add_ic` to store information criteria in the fitted model object for later usage.

Value

If just one object is provided, an object of class `ic`. If multiple objects are provided, an object of class `iclist`.

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

References

- Vehtari, A., Gelman, A., & Gabry J. (2016). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. In *Statistics and Computing*, doi:10.1007/s11222-016-9696-4. arXiv preprint arXiv:1507.04544.
- Gelman, A., Hwang, J., & Vehtari, A. (2014). Understanding predictive information criteria for Bayesian models. *Statistics and Computing*, 24, 997-1016.
- Watanabe, S. (2010). Asymptotic equivalence of Bayes cross validation and widely applicable information criterion in singular learning theory. *The Journal of Machine Learning Research*, 11, 3571-3594.

Examples

```
## Not run:
# model with population-level effects only
fit1 <- brm(rating ~ treat + period + carry,
            data = inhaler, family = "gaussian")
loo(fit1)

# model with an additional varying intercept for subjects
fit2 <- brm(rating ~ treat + period + carry + (1|subject),
            data = inhaler, family = "gaussian")
# compare both models
loo(fit1, fit2)

## End(Not run)
```

```
loo_model_weights.brmsfit
```

Model averaging via stacking or pseudo-BMA weighting.

Description

Compute model weights for brmsfit objects via stacking or pseudo-BMA weighting. For more details, see [loo::loo_model_weights](#).

Usage

```
## S3 method for class 'brmsfit'
loo_model_weights(x, ..., model_names = NULL)
```

Arguments

x	A fitted model object.
...	More fitted model objects or further arguments passed to the underlying post-processing functions.
model_names	If NULL (the default) will use model names derived from deparsing the call. Otherwise will use the passed values as model names.

Value

A named vector of model weights.

Examples

```
## Not run:
# model with population-level effects only
fit1 <- brm(rating ~ treat + period + carry,
            data = inhaler, family = "gaussian")
# model with an additional varying intercept for subjects
fit2 <- brm(rating ~ treat + period + carry + (1|subject),
            data = inhaler, family = "gaussian")
loo_model_weights(fit1, fit2)

## End(Not run)
```

loo_predict.brmsfit *Compute Weighted Expectations Using LOO*

Description

These functions are wrappers around the [E_loo](#) function of the **loo** package.

Usage

```
## S3 method for class 'brmsfit'
loo_predict(object, type = c("mean", "var", "quantile"),
            probs = 0.5, psis_object = NULL, resp = NULL, ...)

## S3 method for class 'brmsfit'
loo_linpred(object, type = c("mean", "var", "quantile"),
            probs = 0.5, psis_object = NULL, resp = NULL, scale = "linear", ...)

## S3 method for class 'brmsfit'
loo_predictive_interval(object, prob = 0.9,
                        psis_object = NULL, ...)
```

Arguments

object	An object of class <code>brmsfit</code> .
type	The statistic to be computed on the results. Can be either "mean" (default), "var", or "quantile".
probs	A vector of quantiles to compute. Only used if <code>type = quantile</code> .
psis_object	An optional object returned by psis . If <code>psis_object</code> is missing then psis is executed internally, which may be time consuming for models fit to very large datasets.
resp	Optional names of response variables. If specified, fitted values of these response variables are returned.

...	Optional arguments passed to the underlying methods that is <code>log_lik</code> , as well as <code>predict</code> or <code>fitted</code> .
scale	Passed to <code>fitted</code> .
prob	For <code>loo_predictive_interval</code> , a scalar in $(0, 1)$ indicating the desired probability mass to include in the intervals. The default is <code>prob = 0.9</code> (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. `loo_predictive_interval()` is equivalent to `loo_predict(..., type = "quantile", probs = c(a, 1-a))` with $a = (1 - p)/2$, except it transposes the result and adds informative column names.

Examples

```
## Not run:
## data from help("lm")
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
d <- data.frame(
  weight = c(ctl, trt),
  group = gl(2, 10, 20, labels = c("Ctl", "Trt"))
)
fit <- brm(weight ~ group, data = d)
loo_predictive_interval(fit, prob = 0.8)

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

## End(Not run)
```

make_conditions

Prepare Fully Crossed Conditions

Description

This is a helper function to prepare fully crossed conditions primarily for use with the `conditions` argument of `marginal_effects`. Automatically creates labels for each row in the `cond__` column.

Usage

```
make_conditions(x, vars, ...)
```


Arguments

x	An R object from which to extract the variables that should be part of the conditions.
vars	Names of the variables that should be part of the conditions.
...	Arguments passed to rows2labels .

Details

For factor like variables, all levels are used as conditions. For numeric variables, mean + (-1:1) * SD are used as conditions.

Value

A `data.frame` where each row indicates a condition.

See Also

[marginal_effects](#), [rows2labels](#)

Examples

```
df <- data.frame(x = c("a", "b"), y = rnorm(10))
make_conditions(df, vars = c("x", "y"))
```

make_stancode	<i>Stan Code for brms Models</i>
---------------	---

Description

Generate Stan code for **brms** models

Usage

```
make_stancode(formula, data, family = gaussian(), prior = NULL,
  autocor = NULL, cov_ranef = NULL, sparse = FALSE,
  sample_prior = c("no", "yes", "only"), stanvars = NULL,
  stan_funs = NULL, save_model = NULL, silent = FALSE, ...)
```

Arguments

formula	An object of class formula , brmsformula , or mvbrmsformula (or one that can be coerced to that classes): A symbolic description of the model to be fitted. The details of model specification are explained in brmsformula .
data	An object of class <code>data.frame</code> (or one that can be coerced to that class) containing data of all variables used in the model.

family	A description of the response distribution and link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a <code>link</code> argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details of supported families see brmsfamily . By default, a linear gaussian model is applied. In multivariate models, family might also be a list of families.
prior	One or more <code>brmsprior</code> objects created by set_prior or related functions and combined using the <code>c</code> method or the <code>+</code> operator. See also get_prior for more help.
autocor	An optional <code>cor_brms</code> object describing the correlation structure within the response variable (i.e., the 'autocorrelation'). See the documentation of cor_brms for a description of the available correlation structures. Defaults to <code>NULL</code> , corresponding to no correlations. In multivariate models, autocor might also be a list of autocorrelation structures.
cov_ranef	A list of matrices that are proportional to the (within) covariance structure of the group-level effects. The names of the matrices should correspond to columns in data that are used as grouping factors. All levels of the grouping factor should appear as rownames of the corresponding matrix. This argument can be used, among others to model pedigrees and phylogenetic effects. See <code>vignette("brms_phylogenetics")</code> for more details.
sparse	Logical; indicates whether the population-level design matrices should be treated as sparse (defaults to <code>FALSE</code>). For design matrices with many zeros, this can considerably reduce required memory. Sampling speed is currently not improved or even slightly decreased.
sample_prior	Indicate if samples from all specified proper priors should be drawn additionally to the posterior samples (defaults to <code>"no"</code>). Among others, these samples can be used to calculate Bayes factors for point hypotheses. If set to <code>"only"</code> , samples are drawn solely from the priors ignoring the likelihood. In this case, all parameters must have proper priors.
stanvars	An optional <code>stanvars</code> object generated by function stanvar to define additional variables in the data block of Stan .
stan_funs	An optional character string containing self-defined Stan functions, which will be included in the functions block of the generated Stan code.
save_model	Either <code>NULL</code> or a character string. In the latter case, the model code is saved in a file named after the string supplied in <code>save_model</code> , which may also contain the full path where to save the file. If only a name is given, the file is saved in the current working directory.
silent	logical; If <code>TRUE</code> , warnings of the Stan parser will be suppressed.
...	Other arguments for internal usage only

Value

A character string containing the fully commented **Stan** code to fit a **brms** model.

Examples

```
make_stancode(rating ~ treat + period + carry + (1|subject),
              data = inhaler, family = "cumulative")
```

```
make_stancode(count ~ log_Age_c + log_Base4_c * Trt_c
              + (1|patient) + (1|visit),
              data = epilepsy, family = "poisson")
```

 make_standata

Data for brms Models

Description

Generate data for **brms** models to be passed to **Stan**

Usage

```
make_standata(formula, data, family = gaussian(), prior = NULL,
              autocor = NULL, cov_ranef = NULL, sample_prior = c("no", "yes", "only"),
              stanvars = NULL, knots = NULL, check_response = TRUE,
              only_response = FALSE, control = list(), ...)
```

Arguments

- | | |
|---------|--|
| formula | An object of class formula , brmsformula , or mvbrmsformula (or one that can be coerced to that classes): A symbolic description of the model to be fitted. The details of model specification are explained in brmsformula . |
| data | An object of class <code>data.frame</code> (or one that can be coerced to that class) containing data of all variables used in the model. |
| family | A description of the response distribution and link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a <code>link</code> argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details of supported families see brmsfamily . By default, a linear gaussian model is applied. In multivariate models, family might also be a list of families. |
| prior | One or more <code>brmsprior</code> objects created by set_prior or related functions and combined using the <code>c</code> method or the <code>+</code> operator. See also get_prior for more help. |
| autocor | An optional cor_brms object describing the correlation structure within the response variable (i.e., the 'autocorrelation'). See the documentation of cor_brms for a description of the available correlation structures. Defaults to <code>NULL</code> , corresponding to no correlations. In multivariate models, <code>autocor</code> might also be a list of autocorrelation structures. |

cov_ranef	A list of matrices that are proportional to the (within) covariance structure of the group-level effects. The names of the matrices should correspond to columns in data that are used as grouping factors. All levels of the grouping factor should appear as rownames of the corresponding matrix. This argument can be used, among others to model pedigrees and phylogenetic effects. See <code>vignette("brms_phylogenetics")</code> for more details.
sample_prior	Indicate if samples from all specified proper priors should be drawn additionally to the posterior samples (defaults to "no"). Among others, these samples can be used to calculate Bayes factors for point hypotheses. If set to "only", samples are drawn solely from the priors ignoring the likelihood. In this case, all parameters must have proper priors.
stanvars	An optional <code>stanvars</code> object generated by function <code>stanvar</code> to define additional variables in the data block of Stan .
knots	Optional list containing user specified knot values to be used for basis construction of smoothing terms. See <code>gamm</code> for more details.
check_response	Logical; check validity of the response?
only_response	Logical; extract data related to the response only?
control	A named list currently for internal usage only
...	Other potential arguments

Value

A named list of objects containing the required data to fit a **brms** model with **Stan**.

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

Examples

```
data1 <- make_standata(rating ~ treat + period + carry + (1|subject),
                      data = inhaler, family = "cumulative")
names(data1)

data2 <- make_standata(count ~ log_Age_c + log_Base4_c * Trt_c
                      + (1|patient) + (1|visit),
                      data = epilepsy, family = "poisson")
names(data2)
```

marginal_effects.brmsfit

Display Marginal Effects of Predictors

Description

Display marginal effects of one or more numeric and/or categorical predictors including two-way interaction effects.

Usage

```
## S3 method for class 'brmsfit'
marginal_effects(x, effects = NULL, conditions = NULL,
  int_conditions = NULL, re_formula = NA, robust = TRUE,
  probs = c(0.025, 0.975), method = c("fitted", "predict"),
  spaghetti = FALSE, surface = FALSE, ordinal = FALSE, transform = NULL,
  resolution = 100, select_points = 0, too_far = 0, ...)

marginal_effects(x, ...)

## S3 method for class 'brmsMarginalEffects'
plot(x, ncol = NULL, points = FALSE,
  rug = FALSE, mean = TRUE, jitter_width = 0, stype = c("contour",
  "raster"), line_args = list(), cat_args = list(),
  errorbar_args = list(), surface_args = list(), spaghetti_args = list(),
  point_args = list(), rug_args = list(), theme = NULL, ask = TRUE,
  plot = TRUE, ...)
```

Arguments

x	An R object usually of class <code>brmsfit</code> .
effects	An optional character vector naming effects (main effects or interactions) for which to compute marginal plots. Interactions are specified by a <code>:</code> between variable names. If <code>NULL</code> (the default), plots are generated for all main effects and two-way interactions estimated in the model. When specifying effects manually, <i>all</i> two-way interactions may be plotted even if not originally modeled.
conditions	An optional data frame containing variable values to condition on. Each effect defined in <code>effects</code> will be plotted separately for each row of conditions. Values in the <code>cond__</code> column will be used as titles of the subplots. If <code>cond__</code> is not given, the row names will be used for this purpose instead. It is recommended to only define a few rows in order to keep the plots clear. See make_conditions for an easy way to define conditions. If <code>NULL</code> (the default), numeric variables will be marginalized by using their means and factors will get their reference level assigned.

int_conditions	An optional named list whose elements are numeric vectors of values of the second variables in two-way interactions. At these values, predictions are evaluated. The names of int_conditions have to match the variable names exactly. Additionally, the elements of the numeric vectors may be named themselves, in which case their names appear as labels for the conditions in the plots. Instead of vectors, functions returning vectors may be passed and are applied on the original values of the corresponding variable. If NULL (the default), predictions are evaluated at the <i>mean</i> and at <i>mean + / - sd</i> .
re_formula	A formula containing random effects to be considered in the marginal predictions. If NULL, include all random effects; if NA (default), include no random effects.
robust	If TRUE (the default) the median is used as the measure of central tendency. If FALSE the mean is used instead.
probs	The quantiles to be used in the computation of credible intervals (defaults to 2.5 and 97.5 percent quantiles)
method	Either "fitted" or "predict". If "fitted", plot marginal predictions of the regression curve. If "predict", plot marginal predictions of the responses.
spaghetti	Logical. Indicates if predictions should be visualized via spaghetti plots. Only applied for numeric predictors. If TRUE, it is recommended to set argument nsamples to a relatively small value (e.g. 100) in order to reduce computation time.
surface	Logical. Indicates if interactions or two-dimensional smooths should be visualized as a surface. Defaults to FALSE. The surface type can be controlled via argument stype of the related plotting method.
ordinal	Logical. Indicates if effects in ordinal models should be visualized as a raster with the response categories on the y-axis. Defaults to FALSE and ignored in non-ordinal models.
transform	A function or a character string naming a function to be applied on the predicted responses before summary statistics are computed. Only allowed if method = "predict".
resolution	Number of support points used to generate the plots. Higher resolution leads to smoother plots. Defaults to 100. If surface is TRUE, this implies 10000 support points for interaction terms, so it might be necessary to reduce resolution when only few RAM is available.
select_points	Positive number. Only relevant if points or rug are set to TRUE: Actual data points of numeric variables that are too far away from the values specified in conditions can be excluded from the plot. Values are scaled into the unit interval and then points more than select_points from the values in conditions are excluded. By default, all points are used.
too_far	Positive number. For surface plots only: Grid points that are too far away from the actual data points can be excluded from the plot. too_far determines what is too far. The grid is scaled into the unit square and then grid points more than too_far from the predictor variables are excluded. By default, all grid points are used. Ignored for non-surface plots.
...	Further arguments such as subset or nsamples passed to predict or fitted .

ncol	Number of plots to display per column for each effect. If NULL (default), ncol is computed internally based on the number of rows of conditions.
points	Logical; indicating whether the original data points should be added via geom_jitter . Default is FALSE. Note that only those data points will be added that match the specified conditions defined in conditions. For categorical predictors, the conditions have to match exactly. For numeric predictors, argument select_points is used to determine, which points do match a condition.
rug	Logical; indicating whether a rug representation of predictor values should be added via geom_rug . Default is FALSE. Depends on select_points in the same way as points does.
mean	Logical; only relevant for spaghetti plots. If TRUE (the default), display the mean regression line on top of the regression lines for each sample.
jitter_width	Only used if points = TRUE: Amount of horizontal jittering of the data points. Mainly useful for ordinal models. Defaults to 0 that is no jittering.
stype	Indicates how surface plots should be displayed. Either "contour" or "raster".
line_args	Only used in plots of continuous predictors: A named list of arguments passed to geom_smooth .
cat_args	Only used in plots of categorical predictors: A named list of arguments passed to geom_point .
errorbar_args	Only used in plots of categorical predictors: A named list of arguments passed to geom_errorbar .
surface_args	Only used in surface plots: A named list of arguments passed to geom_contour or geom_raster (depending on argument stype).
spaghetti_args	Only used in spaghetti plots: A named list of arguments passed to geom_smooth .
point_args	Only used if points = TRUE: A named list of arguments passed to geom_jitter .
rug_args	Only used if rug = TRUE: A named list of arguments passed to geom_rug .
theme	A theme object modifying the appearance of the plots. For some basic themes see ggtheme and theme_default .
ask	logical; indicates if the user is prompted before a new page is plotted. Only used if plot is TRUE.
plot	logical; indicates if plots should be plotted directly in the active graphic device. Defaults to TRUE.

Details

When creating `marginal_effects` for a particular predictor (or interaction of two predictors), one has to choose the values of all other predictors to condition on. By default, the mean is used for continuous variables and the reference category is used for factors, but you may change these values via argument `conditions`. This also has an implication for the `points` argument: In the created plots, only those points will be shown that correspond to the factor levels actually used in the conditioning, in order not to create the false impression of bad model fit, where it is just due to conditioning on certain factor levels. Since we condition on rather than actually marginalizing variables, the name `marginal_effects` is possibly not ideally chosen in retrospect.

NA values within factors in conditions, are interpreted as if all dummy variables of this factor are zero. This allows, for instance, to make predictions of the grand mean when using sum coding.

To fully change colors of the created plots, one has to amend both `scale_colour` and `scale_fill`. See [scale_colour_grey](#) or [scale_colour_gradient](#) for more details.

Value

An object of class `brmsMarginalEffects`, which is a named list with one `data.frame` per effect containing all information required to generate marginal effects plots. Among others, these `data.frames` contain some special variables, namely `estimate__` (predicted values of the response), `se__` (standard error of the predicted response), `lower__` and `upper__` (lower and upper bounds of the uncertainty interval of the response), as well as `cond__` (used in faceting when conditions contains multiple rows).

The corresponding plot method returns a named list of `ggplot` objects, which can be further customized using the `ggplot2` package.

Examples

```
## Not run:
fit <- brm(count ~ log_Age_c + log_Base4_c * Trt + (1 | patient),
           data = epilepsy, family = poisson())

## plot all marginal effects
plot(marginal_effects(fit), ask = FALSE)

## change colours to grey scale
me <- marginal_effects(fit, "log_Base4_c:Trt")
plot(me, plot = FALSE)[[1]] +
  scale_color_grey() +
  scale_fill_grey()

## only plot the marginal interaction effect of 'log_Base4_c:Trt'
## for different values for 'log_Age_c'
conditions <- data.frame(log_Age_c = c(-0.3, 0, 0.3))
plot(marginal_effects(fit, effects = "log_Base4_c:Trt",
                     conditions = conditions))

## also incorporate random effects variance over patients
## also add data points and a rug representation of predictor values
plot(marginal_effects(fit, effects = "log_Base4_c:Trt",
                     conditions = conditions, re_formula = NULL),
     points = TRUE, rug = TRUE)

## change handling of two-way interactions
int_conditions <- list(
  log_Base4_c = setNames(c(-2, 1, 0), c("b", "c", "a"))
)
marginal_effects(fit, effects = "Trt:log_Base4_c",
                 int_conditions = int_conditions)
marginal_effects(fit, effects = "Trt:log_Base4_c",
                 int_conditions = list(log_Base4_c = quantile))
```



```
## fit a model to illustrate how to plot 3-way interactions
fit3way <- brm(count ~ log_Age_c * log_Base4_c * Trt, data = epilepsy)
conditions <- make_conditions(fit3way, "log_Age_c")
marginal_effects(
  fit3way, "log_Base4_c:Trt", conditions = conditions
)
## only include points close to the specified values of log_Age_c
me <- marginal_effects(
  fit3way, "log_Base4_c:Trt", conditions = conditions,
  select_points = 0.1
)
plot(me, points = TRUE)

## End(Not run)
```

marginal_smooths.brmsfit

Display Smooth Terms

Description

Display smooth s and t2 terms of models fitted with **brms**.

Usage

```
## S3 method for class 'brmsfit'
marginal_smooths(x, smooths = NULL, int_conditions = NULL,
  probs = c(0.025, 0.975), spaghetti = FALSE, resolution = 100,
  too_far = 0, subset = NULL, nsamples = NULL, ...)

marginal_smooths(x, ...)
```

Arguments

x	An R object usually of class <code>brmsfit</code> .
smooths	Optional character vector of smooth terms to display. If <code>NULL</code> (the default) all smooth terms are shown.
int_conditions	An optional named list whose elements are numeric vectors of values of the second variables in two-way interactions. At these values, predictions are evaluated. The names of <code>int_conditions</code> have to match the variable names exactly. Additionally, the elements of the numeric vectors may be named themselves, in which case their names appear as labels for the conditions in the plots. Instead of vectors, functions returning vectors may be passed and are applied on the original values of the corresponding variable. If <code>NULL</code> (the default), predictions are evaluated at the <i>mean</i> and at <i>mean + / - sd</i> .

<code>probs</code>	The quantiles to be used in the computation of credible intervals (defaults to 2.5 and 97.5 percent quantiles)
<code>spaghetti</code>	Logical. Indicates if predictions should be visualized via spaghetti plots. Only applied for numeric predictors. If TRUE, it is recommended to set argument <code>nsamples</code> to a relatively small value (e.g. 100) in order to reduce computation time.
<code>resolution</code>	Number of support points used to generate the plots. Higher resolution leads to smoother plots. Defaults to 100. If <code>surface</code> is TRUE, this implies 10000 support points for interaction terms, so it might be necessary to reduce resolution when only few RAM is available.
<code>too_far</code>	Positive number. For surface plots only: Grid points that are too far away from the actual data points can be excluded from the plot. <code>too_far</code> determines what is too far. The grid is scaled into the unit square and then grid points more than <code>too_far</code> from the predictor variables are excluded. By default, all grid points are used. Ignored for non-surface plots.
<code>subset</code>	A numeric vector specifying the posterior samples to be used. If NULL (the default), all samples are used.
<code>nsamples</code>	Positive integer indicating how many posterior samples should be used. If NULL (the default) all samples are used. Ignored if <code>subset</code> is not NULL.
<code>...</code>	Currently ignored.

Details

Two-dimensional smooth terms will be visualized using either contour or raster plots.

Value

For the `brmsfit` method, an object of class `brmsMarginalEffects`. See [marginal_effects](#) for more details and documentation of the related plotting function.

Examples

```
## Not run:
set.seed(0)
dat <- mgcv::gamSim(1, n = 200, scale = 2)
fit <- brm(y ~ s(x0) + s(x1) + s(x2) + s(x3), data = dat)
# show all smooth terms
plot(marginal_smooths(fit), rug = TRUE, ask = FALSE)
# show only the smooth term s(x2)
plot(marginal_smooths(fit, smooths = "s(x2)"), ask = FALSE)

# fit and plot a two-dimensional smooth term
fit2 <- brm(y ~ t2(x0, x2), data = dat)
ms <- marginal_smooths(fit2)
plot(ms, stype = "contour")
plot(ms, stype = "raster")

## End(Not run)
```

Description

Specify predictors with measurement error. This function is almost solely useful when called in formulas passed to the **brms** package.

Usage

```
me(x, sdx = NULL, gr = NULL)
```

Arguments

x	The variable measured with error.
sdx	Known measurement error of x treated as standard deviation.
gr	Optional grouping factor to specify which values of x correspond to the same value of the latent variable. If NULL (the default) each observation will have its own value of the latent variable.

Details

For detailed documentation see `help(brmsformula)`.

By default, latent noise-free variables are assumed to be correlated. To change that, add `set_mecor(FALSE)` to your model formula object (see examples).

See Also

[brmsformula](#), [brmsformula-helpers](#)

Examples

```
## Not run:
# sample some data
N <- 100
dat <- data.frame(
  y = rnorm(N), x1 = rnorm(N),
  x2 = rnorm(N), sdx = abs(rnorm(N, 1))
)
# fit a simple error-in-variables model
fit1 <- brm(y ~ me(x1, sdx) + me(x2, sdx), data = dat,
  save_mevars = TRUE)
summary(fit1)

# turn off modeling of correlations
bform <- bf(y ~ me(x1, sdx) + me(x2, sdx)) + set_mecor(FALSE)
fit2 <- brm(bform, data = dat, save_mevars = TRUE)
summary(fit2)
```

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

mi

*Predictors with Missing Values in **brms** Models*

Description

Predictors with Missing Values in **brms** Models

Usage

```
mi(x)
```

Arguments

x The variable containing missings.

Details

For detailed documentation see `help(brmsformula)`.

This function is almost solely useful when called in formulas passed to the **brms** package.

See Also

[brmsformula](#)

Examples

```
## Not run:
data("nhanes", package = "mice")
bform <- bf(bmi | mi() ~ age * mi(chl)) +
  bf(chl | mi() ~ age) + set_rescor(FALSE)
fit <- brm(bform, data = nhanes)
summary(fit)
plot(marginal_effects(fit, resp = "bmi"), ask = FALSE)
LOO(fit, newdata = na.omit(fit$data))

## End(Not run)
```

Description

Set up a finite mixture family for use in **brms**.

Usage

```
mixture(..., flist = NULL, nmix = 1, order = NULL)
```

Arguments

...	One or more objects providing a description of the response distributions to be combined in the mixture model. These can be family functions, calls to family functions or character strings naming the families. For details of supported families see brmsfamily .
flist	Optional list of objects, which are treated in the same way as objects passed via the ... argument.
nmix	Optional numeric vector specifying the number of times each family is repeated. If specified, it must have the same length as the number of families passed via ... or flist.
order	Ordering constraint to identify mixture components. If 'mu' or TRUE, population-level intercepts of the mean parameters are ordered. If 'none' or FALSE, no ordering constraint is applied. If NULL (the default), order is set to 'mu' if all families are the same and 'none' otherwise. Other ordering constraints may be implemented in the future.

Details

Most families supported by **brms** can be used to form mixtures. The response variable has to be valid for all components of the mixture family. Currently, the number of mixture components has to be specified by the user. It is not yet possible to estimate the number of mixture components from the data.

For most mixture models, you may want to specify priors on the population-level intercepts via [set_prior](#) to improve convergence. In addition, it is sometimes necessary to set `inits = 0` in the call to `brm` to allow chains to initialize properly.

For more details on the specification of mixture models, see [brmsformula](#).

Value

An object of class `mixfamily`.

Examples

```

## Not run:
## simulate some data
set.seed(1234)
dat <- data.frame(
  y = c(rnorm(200), rnorm(100, 6)),
  x = rnorm(300),
  z = sample(0:1, 300, TRUE)
)

## fit a simple normal mixture model
mix <- mixture(gaussian, gaussian)
prior <- c(
  prior(normal(0, 7), Intercept, dpar = mu1),
  prior(normal(5, 7), Intercept, dpar = mu2)
)
fit1 <- brm(bf(y ~ x + z), dat, family = mix,
  prior = prior, chains = 2)
summary(fit1)
pp_check(fit1)

## use different predictors for the components
fit2 <- brm(bf(y ~ 1, mu1 ~ x, mu2 ~ z), dat, family = mix,
  prior = prior, chains = 2)
summary(fit2)

## fix the mixing proportions
fit3 <- brm(bf(y ~ x + z, theta1 = 1, theta2 = 2),
  dat, family = mix, prior = prior,
  inits = 0, chains = 2)
summary(fit3)
pp_check(fit3)

## predict the mixing proportions
fit4 <- brm(bf(y ~ x + z, theta2 ~ x),
  dat, family = mix, prior = prior,
  inits = 0, chains = 2)
summary(fit4)
pp_check(fit4)

## compare model fit
L00(fit1, fit2, fit3, fit4)

## End(Not run)

```

Description

Function to set up a multi-membership grouping term in **brms**. The function does not evaluate its arguments – it exists purely to help set up a model with grouping terms.

Usage

```
mm(..., weights = NULL, scale = TRUE, dist = "gaussian")
```

Arguments

...	One or more terms containing grouping factors.
weights	A matrix specifying the weights of each member. It should have as many columns as grouping terms specified in If NULL (the default), equally weights are used.
scale	Logical; if TRUE (the default), weights are standardized in order to sum to one per row. If negative weights are specified, scale needs to be set to FALSE.
dist	Name of the distribution of the group-level effects. Currently "gaussian" is the only option.

See Also

[brmsformula](#), [mmc](#)

Examples

```
## Not run:
# simulate some data
dat <- data.frame(
  y = rnorm(100), x1 = rnorm(100), x2 = rnorm(100),
  g1 = sample(1:10, 100, TRUE), g2 = sample(1:10, 100, TRUE)
)

# multi-membership model with two members per group and equal weights
fit1 <- brm(y ~ x1 + (1|mm(g1, g2)), data = dat)
summary(fit1)

# weight the first member two times for than the second member
dat$w1 <- rep(2, 100)
dat$w2 <- rep(1, 100)
fit2 <- brm(y ~ x1 + (1|mm(g1, g2, weights = cbind(w1, w2))), data = dat)
summary(fit2)

# multi-membership model with level specific covariate values
dat$xc <- (dat$x1 + dat$x2) / 2
fit3 <- brm(y ~ xc + (1 + mmc(x1, x2) | mm(g1, g2)), data = dat)
summary(fit3)

## End(Not run)
```

`mmc`*Multi-Membership Covariates*

Description

Specify covariates that vary over different levels of multi-membership grouping factors thus requiring special treatment. This function is almost solely useful, when called in combination with [mm](#). Outside of multi-membership terms it will behave very much like [cbind](#).

Usage

```
mmc(...)
```

Arguments

... One or more terms containing covariates corresponding to the grouping levels specified in [mm](#).

Value

A matrix with covariates as columns.

See Also

[mm](#)

Examples

```
## Not run:
# simulate some data
dat <- data.frame(
  y = rnorm(100), x1 = rnorm(100), x2 = rnorm(100),
  g1 = sample(1:10, 100, TRUE), g2 = sample(1:10, 100, TRUE)
)

# multi-membership model with level specific covariate values
dat$xc <- (dat$x1 + dat$x2) / 2
fit <- brm(y ~ xc + (1 + mmc(x1, x2) | mm(g1, g2)), data = dat)
summary(fit)

## End(Not run)
```


Description

Monotonic Predictors in **brms** Models

Usage

```
mo(x)
```

Arguments

x An integer variable or an ordered factor to be modeled as monotonic.

Details

For detailed documentation see `help(brmsformula)` as well as `vignette("brms_monotonic")`.

This function is almost solely useful when called in formulas passed to the **brms** package.

See Also

[brmsformula](#)

Examples

```
## Not run:
# generate some data
income_options <- c("below_20", "20_to_40", "40_to_100", "greater_100")
income <- factor(sample(income_options, 100, TRUE),
                 levels = income_options, ordered = TRUE)
mean_ls <- c(30, 60, 70, 75)
ls <- mean_ls[income] + rnorm(100, sd = 7)
dat <- data.frame(income, ls)

# fit a simple monotonic model
fit1 <- brm(ls ~ mo(income), data = dat)

# summarise the model
summary(fit1)
plot(fit1, N = 6)
plot(marginal_effects(fit1), points = TRUE)

# model interaction with other variables
dat$x <- sample(c("a", "b", "c"), 100, TRUE)
fit2 <- brm(ls ~ mo(income)*x, data = dat)

# summarise the model
```

```
summary(fit2)
plot(marginal_effects(fit2), points = TRUE)

## End(Not run)
```

model_weights.brmsfit *Model Weighting Methods*

Description

Compute model weights in various ways, for instance via stacking of predictive distributions, Akaike weights, or marginal likelihoods.

Usage

```
## S3 method for class 'brmsfit'
model_weights(x, ..., weights = "loo2",
  model_names = NULL)

model_weights(x, ...)
```

Arguments

x	A fitted model object.
...	More fitted model objects or further arguments passed to the underlying post-processing functions.
weights	Name of the criterion to compute weights from. Should be one of "loo", "waic", "kfold", "loo2" (current default), or "marglik". For the former three options, Akaike weights will be computed based on the information criterion values returned by the respective methods. For "loo2", method loo_model_weights will be used to obtain weights. For "marglik", method post_prob will be used to compute weights based on log marginal likelihood values (make sure to specify reasonable priors in this case). Alternatively, weights can be a numeric vector of pre-specified weights.
model_names	If NULL (the default) will use model names derived from deparsing the call. Otherwise will use the passed values as model names.

Value

A numeric vector of weights for the models.

Examples

```
## Not run:
# model with 'treat' as predictor
fit1 <- brm(rating ~ treat + period + carry, data = inhaler)
summary(fit1)

# model without 'treat' as predictor
fit2 <- brm(rating ~ period + carry, data = inhaler)
summary(fit2)

# obtain Akaike weights based on the WAIC
model_weights(fit1, fit2, weights = "waic")

## End(Not run)
```

MultiNormal

The Multivariate Normal Distribution

Description

Density function and random generation for the multivariate normal distribution with mean vector μ and covariance matrix Σ .

Usage

```
dmulti_normal(x, mu, Sigma, log = FALSE, check = FALSE)

rmulti_normal(n, mu, Sigma, check = FALSE)
```

Arguments

x	Vector or matrix of quantiles. If x is a matrix, each row is taken to be a quantile.
mu	Mean vector with length equal to the number of dimensions.
Sigma	Covariance matrix.
log	Logical; If TRUE, values are returned on the log scale.
check	Logical; Indicates whether several input checks should be performed. Defaults to FALSE to improve efficiency.
n	Number of samples to draw from the distribution.

Details

See the Stan user's manual <http://mc-stan.org/documentation/> for details on the parameterization

MultiStudentT

The Multivariate Student-t Distribution

Description

Density function and random generation for the multivariate Student-t distribution with location vector μ , covariance matrix Σ , and degrees of freedom df .

Usage

```
dmulti_student_t(x, df, mu, Sigma, log = FALSE, check = FALSE)
```

```
rmulti_student_t(n, df, mu, Sigma, check = FALSE)
```

Arguments

<code>x</code>	Vector or matrix of quantiles. If <code>x</code> is a matrix, each row is taken to be a quantile.
<code>df</code>	Vector of degrees of freedom.
<code>mu</code>	Location vector with length equal to the number of dimensions.
<code>Sigma</code>	Covariance matrix.
<code>log</code>	Logical; If TRUE, values are returned on the log scale.
<code>check</code>	Logical; Indicates whether several input checks should be performed. Defaults to FALSE to improve efficiency.
<code>n</code>	Number of samples to draw from the distribution.

Details

See the Stan user's manual <http://mc-stan.org/documentation/> for details on the parameterization

mvbrmsformula

*Set up a multivariate model formula for use in **brms***

Description

Set up a multivariate model formula for use in the **brms** package allowing to define (potentially non-linear) additive multilevel models for all parameters of the assumed response distributions.

Usage

```
mvbrmsformula(..., flist = NULL, rescor = NULL)
```

Arguments

...	Objects of class <code>formula</code> or <code>brmsformula</code> , each specifying a univariate model. See brmsformula for details on how to specify univariate models.
<code>flist</code>	Optional list of formulas, which are treated in the same way as formulas passed via the ... argument.
<code>rescor</code>	Logical; Indicates if residual correlation between the response variables should be modeled. Currently, this is only possible in multivariate gaussian and student models. If NULL (the default), <code>rescor</code> is internally set to TRUE when possible.

Details

See `vignette("brms_multivariate")` for a case study.

Value

An object of class `mvbrmsformula`, which is essentially a list containing all model formulas as well as some additional information for multivariate models.

See Also

[brmsformula](#), [brmsformula-helpers](#)

Examples

```
bf1 <- bf(y1 ~ x + (1|g))
bf2 <- bf(y2 ~ s(z))
mvbf(bf1, bf2)
```

<code>ngrps.brmsfit</code>	<i>Number of levels</i>
----------------------------	-------------------------

Description

Extract the number of levels of one or more grouping factors.

Usage

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

ngrps(object, ...)
```

Arguments

<code>object</code>	An R object.
...	Currently ignored.

Value

A named list containing the number of levels per grouping factor.

nsamples.brmsfit	<i>Number of Posterior Samples</i>
------------------	------------------------------------

Description

Extract the number of posterior samples stored in a fitted Bayesian model.

Usage

```
## S3 method for class 'brmsfit'
nsamples(x, subset = NULL, incl_warmup = FALSE, ...)

nsamples(x, ...)
```

Arguments

x	An R object
subset	An optional integer vector defining a subset of samples to be considered.
incl_warmup	A flag indicating whether to also count warmup / burn-in samples.
...	Further arguments passed to or from other methods.

Details

Currently there are methods for brmsfit objects.

pairs.brmsfit	<i>Create a matrix of output plots from a brmsfit object</i>
---------------	--

Description

A [pairs](#) method that is customized for MCMC output.

Usage

```
## S3 method for class 'brmsfit'
pairs(x, pars = NA, exact_match = FALSE, ...)
```

Arguments

<code>x</code>	An object of class <code>brmsfit</code>
<code>pars</code>	Names of parameters for which posterior samples should be returned, as given by a character vector or regular expressions. By default, all posterior samples of all parameters are extracted.
<code>exact_match</code>	Indicates whether parameter names should be matched exactly or treated as regular expression. Default is <code>FALSE</code> .
<code>...</code>	Further arguments to be passed to <code>mcmc_pairs</code> .

Details

For a detailed description see `mcmc_pairs`.

Examples

```
## Not run:
fit <- brm(count ~ log_Age_c + log_Base4_c * Trt_c
           + (1|patient) + (1|visit),
           data = epilepsy, family = "poisson")
pairs(fit, pars = parnames(fit)[1:3], exact_match = TRUE)
pairs(fit, pars = "^sd_")

## End(Not run)
```

parnames

Extract Parameter Names

Description

Extract all parameter names of a given model.

Usage

```
parnames(x, ...)
```

Arguments

<code>x</code>	An R object
<code>...</code>	Further arguments passed to or from other methods.

Details

Currently there are methods for `brmsfit` objects.

Value

A character vector containing the parameter names of the model.

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

parse_bf

*Parse Formulas of **brms** Models*

Description

Parse formulas objects for use in **brms**.

Usage

```
parse_bf(formula, ...)

## Default S3 method:
parse_bf(formula, family = NULL, autocor = NULL, ...)

## S3 method for class 'brmsformula'
parse_bf(formula, family = NULL, autocor = NULL,
  check_response = TRUE, resp_rhs_all = TRUE, mv = FALSE, ...)

## S3 method for class 'mvbrmsformula'
parse_bf(formula, family = NULL, autocor = NULL,
  ...)
```

Arguments

formula	An object of class <code>formula</code> , <code>brmsformula</code> , or <code>mvbrmsformula</code> (or one that can be coerced to that classes): A symbolic description of the model to be fitted. The details of model specification are explained in <code>brmsformula</code> .
...	Further arguments passed to or from other methods.
family	A description of the response distribution and link function to be used in the model. This can be a family function, a call to a family function or a character string naming the family. Every family function has a <code>link</code> argument allowing to specify the link function to be applied on the response variable. If not specified, default links are used. For details of supported families see <code>brmsfamily</code> . By default, a linear gaussian model is applied. In multivariate models, family might also be a list of families.
autocor	An optional <code>cor_brms</code> object describing the correlation structure within the response variable (i.e., the 'autocorrelation'). See the documentation of <code>cor_brms</code> for a description of the available correlation structures. Defaults to <code>NULL</code> , corresponding to no correlations. In multivariate models, autocor might also be a list of autocorrelation structures.
check_response	Logical; Indicates whether the left-hand side of formula (i.e. response variables and addition arguments) should be parsed. If <code>FALSE</code> , formula may also be one-sided.

resp_rhs_all	Logical; Indicates whether to also include response variables on the right-hand side of formula . \$allvars, where . represents the output of parse_bf.
mv	Indicates if the univariate model is part of a multivariate model.

Details

This is the main formula parsing function of **brms**. It should usually not be called directly, but is exported to allow package developers making use of the formula syntax implemented in **brms**. As long as no other packages depend on this functions, it may be changed without deprecation warnings, when new features make this necessary.

Value

An object of class `brmsterms` or `mvbrmsterms` (for multivariate models), which is a list containing all required information initially stored in `formula` in an easier to use format, basically a list of formulas (not an abstract syntax tree).

See Also

[brm](#), [brmsformula](#), [mvbrmsformula](#)

plot.brmsfit	<i>Trace and Density Plots for MCMC Samples</i>
--------------	---

Description

Trace and Density Plots for MCMC Samples

Usage

```
## S3 method for class 'brmsfit'
plot(x, pars = NA, combo = c("dens", "trace"), N = 5,
     exact_match = FALSE, theme = NULL, plot = TRUE, ask = TRUE,
     newpage = TRUE, ...)
```

Arguments

x	An object of class <code>brmsfit</code> .
pars	Names of the parameters to plot, as given by a character vector or a regular expression. By default, all parameters except for group-level and smooth effects are plotted.
combo	A character vector with at least two elements. Each element of <code>combo</code> corresponds to a column in the resulting graphic and should be the name of one of the available MCMC functions (omitting the <code>mcmc_</code> prefix).
N	The number of parameters plotted per page.
exact_match	Indicates whether parameter names should be matched exactly or treated as regular expression. Default is <code>FALSE</code> .

theme	A theme object modifying the appearance of the plots. For some basic themes see ggtheme and theme_default .
plot	logical; indicates if plots should be plotted directly in the active graphic device. Defaults to TRUE.
ask	logical; indicates if the user is prompted before a new page is plotted. Only used if plot is TRUE.
newpage	logical; indicates if the first set of plots should be plotted to a new page. Only used if plot is TRUE.
...	Further arguments passed to mcmc_combo .

Value

An invisible list of [gtable](#) objects.

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

Examples

```
## Not run:
fit <- brm(count ~ log_Age_c + log_Base4_c * Trt_c
          + (1|patient) + (1|visit),
          data = epilepsy, family = "poisson")
plot(fit)
## plot population-level effects only
plot(fit, pars = "^b_")

## End(Not run)
```

posterior_average.brmsfit

Posterior samples of parameters averaged across models

Description

Extract posterior samples of parameters averaged across models. Weighting can be done in various ways, for instance using Akaike weights based on information criteria or marginal likelihoods.

Usage

```
## S3 method for class 'brmsfit'
posterior_average(x, ..., pars = NULL, weights = "loo2",
                 nsamples = NULL, missing = NULL, model_names = NULL, control = list(),
                 seed = NULL)

posterior_average(x, ...)
```

Arguments

x	A fitted model object.
...	More fitted model objects or further arguments passed to the underlying post-processing functions.
pars	Names of parameters for which to average across models. Only those parameters can be averaged that appear in every model. Defaults to all overlapping parameters.
weights	Name of the criterion to compute weights from. Should be one of "loo", "waic", "kfold", "loo2" (current default), or "marglik". For the former three options, Akaike weights will be computed based on the information criterion values returned by the respective methods. For "loo2", method loo_model_weights will be used to obtain weights. For "marglik", method post_prob will be used to compute weights based on log marginal likelihood values (make sure to specify reasonable priors in this case). Alternatively, weights can be a numeric vector of pre-specified weights.
nsamples	Total number of posterior samples to use.
missing	An optional numeric value or a named list of numeric values to use if a model does not contain a parameter for which posterior samples should be averaged. Defaults to NULL, in which case only those parameters can be averaged that are present in all of the models.
model_names	If NULL (the default) will use model names derived from parsing the call. Otherwise will use the passed values as model names.
control	Optional list of further arguments passed to the function specified in weights.
seed	A single numeric value passed to set.seed to make results reproducible.

Details

Weights are computed with the [model_weights](#) method.

Value

A data frame of posterior samples. Samples are rows and parameters are columns.

See Also

[model_weights](#), [pp_average](#)

Examples

```
## Not run:
# model with 'treat' as predictor
fit1 <- brm(rating ~ treat + period + carry, data = inhaler)
summary(fit1)

# model without 'treat' as predictor
fit2 <- brm(rating ~ period + carry, data = inhaler)
summary(fit2)
```

```
# compute model-averaged posteriors of overlapping parameters
posterior_average(fit1, fit2, weights = "waic")

## End(Not run)
```

```
posterior_interval.brmsfit
  Compute posterior uncertainty intervals
```

Description

Compute posterior uncertainty intervals for brmsfit objects.

Usage

```
## S3 method for class 'brmsfit'
posterior_interval(object, pars = NA, prob = 0.95, ...)
```

Arguments

object	An object of class brmsfit
pars	Names of parameters for which posterior samples should be returned, as given by a character vector or regular expressions. By default, all posterior samples of all parameters are extracted.
prob	A value between 0 and 1 indicating the desired probability to be covered by the uncertainty intervals. The default is 0.95.
...	More arguments passed to as.matrix.brmsfit .

Value

A matrix with lower and upper interval bounds as columns and as many rows as selected parameters.

Examples

```
## Not run:
fit <- brm(count ~ log_Age_c + log_Base4_c * Trt_c,
           data = epilepsy, family = negbinomial())
posterior_interval(fit)

## End(Not run)
```

```
posterior_samples.brmsfit
  Extract posterior samples
```

Description

Extract posterior samples of specified parameters

Usage

```
## S3 method for class 'brmsfit'
posterior_samples(x, pars = NA, exact_match = FALSE,
  add_chain = FALSE, subset = NULL, as.matrix = FALSE, as.array = FALSE,
  ...)

## S3 method for class 'brmsfit'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'brmsfit'
as.matrix(x, ...)

## S3 method for class 'brmsfit'
as.array(x, ...)

posterior_samples(x, pars = NA, ...)
```

Arguments

<code>x</code>	An R object typically of class <code>brmsfit</code>
<code>pars</code>	Names of parameters for which posterior samples should be returned, as given by a character vector or regular expressions. By default, all posterior samples of all parameters are extracted.
<code>exact_match</code>	Indicates whether parameter names should be matched exactly or treated as regular expression. Default is <code>FALSE</code> .
<code>add_chain</code>	A flag indicating if the returned <code>data.frame</code> should contain two additional columns. The <code>chain</code> column indicates the chain in which each sample was generated, the <code>iter</code> column indicates the iteration number within each chain.
<code>subset</code>	A numeric vector indicating the rows (i.e., posterior samples) to be returned. If <code>NULL</code> (the default), all posterior samples are returned.
<code>as.matrix</code>	Should the output be a matrix instead of a <code>data.frame</code> ? Defaults to <code>FALSE</code> .
<code>as.array</code>	Should the output be an array instead of a <code>data.frame</code> ? Defaults to <code>FALSE</code> .
<code>...</code>	For <code>as.data.frame</code> , <code>as.matrix</code> , and <code>as.array</code> : Further arguments to be passed to <code>posterior_samples</code> .
<code>row.names</code> , <code>optional</code>	See as.data.frame .

Details

Currently there are methods for brmsfit objects. `as.data.frame.brmsfit`, `as.matrix.brmsfit`, and `as.array.brmsfit` are basically aliases of `posterior_samples.brmsfit` and differ from each other only in type of the returned object.

Value

A data frame (matrix or array) containing the posterior samples, with one column per parameter. In case an array is returned, it contains one additional dimension for the chains.

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

Examples

```
## Not run:
fit <- brm(rating ~ treat + period + carry + (1|subject),
          data = inhaler, family = "cumulative")

# extract posterior samples of population-level effects
samples1 <- posterior_samples(fit, "^b")
head(samples1)

# extract posterior samples of group-level standard deviations
samples2 <- posterior_samples(fit, "^sd_")
head(samples2)

## End(Not run)
```

posterior_summary.brmsfit

Summarize Posterior Samples

Description

Summarizes posterior samples based on point estimates (mean or median), estimation errors (SD or MAD) and quantiles.

Usage

```
## S3 method for class 'brmsfit'
posterior_summary(x, pars = NA, probs = c(0.025, 0.975),
                 robust = FALSE, ...)

posterior_summary(x, ...)
```

```
## Default S3 method:
posterior_summary(x, probs = c(0.025, 0.975),
  robust = FALSE, ...)
```

Arguments

x	An R object.
pars	Names of parameters for which posterior samples should be returned, as given by a character vector or regular expressions. By default, all posterior samples of all parameters are extracted.
probs	The percentiles to be computed by the quantile function.
robust	If FALSE (the default) the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If TRUE, the median and the median absolute deviation (MAD) are applied instead.
...	More arguments passed to or from other methods.

Value

A matrix where rows indicate parameters and columns indicate the summary estimates.

Examples

```
## Not run:
fit <- brm(time ~ age * sex, data = kidney)
posterior_summary(fit)

## End(Not run)
```

posterior_table	<i>Table Creation for Posterior Samples</i>
-----------------	---

Description

Create a table for unique values of posterior samples. This is usually only useful when summarizing predictions of ordinal models.

Usage

```
posterior_table(x, levels = NULL)
```

Arguments

x	A matrix of posterior samples where rows indicate samples and columns indicate parameters.
levels	Optional values of possible posterior values. Defaults to all unique values in x.

Value

A matrix where rows indicate parameters and columns indicate the unique values of posterior samples.

Examples

```
## Not run:
fit <- brm(rating ~ period + carry + treat,
           data = inhaler, family = cumulative())
pr <- predict(fit, summary = FALSE)
posterior_table(pr)

## End(Not run)
```

post_prob.brmsfit *Posterior Model Probabilities from Marginal Likelihoods*

Description

Compute posterior model probabilities from marginal likelihoods. The `brmsfit` method is just a thin wrapper around the corresponding method for bridge objects.

Usage

```
## S3 method for class 'brmsfit'
post_prob(x, ..., prior_prob = NULL, model_names = NULL)
```

Arguments

<code>x</code>	A fitted model object.
<code>...</code>	More fitted model objects or further arguments passed to the underlying post-processing functions.
<code>prior_prob</code>	Numeric vector with prior model probabilities. If omitted, a uniform prior is used (i.e., all models are equally likely a priori). The default <code>NULL</code> corresponds to equal prior model weights.
<code>model_names</code>	If <code>NULL</code> (the default) will use model names derived from deparsing the call. Otherwise will use the passed values as model names.

Details

Computing the marginal likelihood requires samples of all variables defined in Stan's parameters block to be saved. Otherwise `post_prob` cannot be computed. Thus, please set `save_all_pars = TRUE` in the call to `brm`, if you are planning to apply `post_prob` to your models.

More details are provided under [post_prob](#).

See Also

[bridge_sampler](#), [bayes_factor](#)

Examples

```
## Not run:
# model with the treatment effect
fit1 <- brm(
  count ~ log_Age_c + log_Base4_c + Trt_c,
  data = epilepsy, family = negbinomial(),
  prior = prior(normal(0, 1), class = b),
  save_all_pars = TRUE
)
summary(fit1)

# model without the treatent effect
fit2 <- brm(
  count ~ log_Age_c + log_Base4_c,
  data = epilepsy, family = negbinomial(),
  prior = prior(normal(0, 1), class = b),
  save_all_pars = TRUE
)
summary(fit2)

# compute the posterior model probabilities
post_prob(fit1, fit2)

# specify prior model probabilities
post_prob(fit1, fit2, prior_prob = c(0.8, 0.2))

## End(Not run)
```

pp_average.brmsfit *Posterior predictive samples averaged across models*

Description

Compute posterior predictive samples averaged across models. Weighting can be done in various ways, for instance using Akaike weights based on information criteria or marginal likelihoods.

Usage

```
## S3 method for class 'brmsfit'
pp_average(x, ..., weights = "loo2", method = c("predict",
  "fitted", "residuals"), nsamples = NULL, summary = TRUE,
  probs = c(0.025, 0.975), robust = FALSE, model_names = NULL,
  control = list(), seed = NULL)

pp_average(x, ...)
```

Arguments

x	A fitted model object.
...	More fitted model objects or further arguments passed to the underlying post-processing functions.
weights	Name of the criterion to compute weights from. Should be one of "loo", "waic", "kfold", "loo2" (current default), or "marglik". For the former three options, Akaike weights will be computed based on the information criterion values returned by the respective methods. For "loo2", method loo_model_weights will be used to obtain weights. For "marglik", method post_prob will be used to compute weights based on log marginal likelihood values (make sure to specify reasonable priors in this case). Alternatively, weights can be a numeric vector of pre-specified weights.
method	Type of predictions to average. Should be one of "predict" (default), "fitted", or "residuals".
nsamples	Total number of posterior samples to use.
summary	Should summary statistics (i.e. means, sds, and 95% intervals) be returned instead of the raw values? Default is TRUE.
probs	The percentiles to be computed by the <code>quantile</code> function. Only used if <code>summary</code> is TRUE.
robust	If FALSE (the default) the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If TRUE, the median and the median absolute deviation (MAD) are applied instead. Only used if <code>summary</code> is TRUE.
model_names	If NULL (the default) will use model names derived from deparsing the call. Otherwise will use the passed values as model names.
control	Optional list of further arguments passed to the function specified in <code>weights</code> .
seed	A single numeric value passed to set.seed to make results reproducible.

Details

Weights are computed with the [model_weights](#) method.

Value

Same as the output of the method specified in argument `method`.

See Also

[model_weights](#), [posterior_average](#)

Examples

```
## Not run:
# model with 'treat' as predictor
fit1 <- brm(rating ~ treat + period + carry, data = inhaler)
summary(fit1)
```

```

# model without 'treat' as predictor
fit2 <- brm(rating ~ period + carry, data = inhaler)
summary(fit2)

# compute model-averaged predicted values
(df <- unique(inhaler[, c("treat", "period", "carry")]))
pp_average(fit1, fit2, newdata = df)

# compute model-averaged fitted values
pp_average(fit1, fit2, method = "fitted", newdata = df)

## End(Not run)

```

pp_check.brmsfit

*Posterior Predictive Checks for brmsfit Objects***Description**

Perform posterior predictive checks with the help of the **bayesplot** package.

Usage

```

## S3 method for class 'brmsfit'
pp_check(object, type, nsamples, group = NULL, x = NULL,
  newdata = NULL, resp = NULL, subset = NULL, ...)

```

Arguments

object	An object of class <code>brmsfit</code> .
type	Type of the ppc plot as given by a character string. See PPC for an overview of currently supported types. You may also use an invalid type (e.g. <code>type = "xyz"</code>) to get a list of supported types in the resulting error message.
nsamples	Positive integer indicating how many posterior samples should be used. If <code>NULL</code> all samples are used. If not specified, the number of posterior samples is chosen automatically. Ignored if <code>subset</code> is not <code>NULL</code> .
group	Optional name of a factor variable in the model by which to stratify the ppc plot. This argument is required for ppc <code>*_grouped</code> types and ignored otherwise.
x	Optional name of a variable in the model. Only used for ppc types having an <code>x</code> argument and ignored otherwise.
newdata	An optional data.frame for which to evaluate predictions. If <code>NULL</code> (default), the original data of the model is used.
resp	Optional names of response variables. If specified, fitted values of these response variables are returned.

subset A numeric vector specifying the posterior samples to be used. If NULL (the default), all samples are used.

... Further arguments passed to `predict.brmsfit` as well as to the PPC function specified in `type`.

Details

For a detailed explanation of each of the ppc functions, see the [PPC](#) documentation of the `bayesplot` package.

Value

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

Examples

```
## Not run:
fit <- brm(count ~ log_Age_c + log_Base4_c * Trt_c
           + (1|patient) + (1|obs),
           data = epilepsy, family = poisson())

pp_check(fit) # shows dens_overlay plot by default
pp_check(fit, type = "error_hist", nsamples = 11)
pp_check(fit, type = "scatter_avg", nsamples = 100)
pp_check(fit, type = "stat_2d")
pp_check(fit, type = "rootogram")
pp_check(fit, type = "loo_pit")

## get an overview of all valid types
pp_check(fit, type = "xyz")

## End(Not run)
```

pp_mixture.brmsfit *Posterior Probabilities of Mixture Component Memberships*

Description

Compute the posterior probabilities of mixture component memberships for each observation including uncertainty estimates.

Usage

```
## S3 method for class 'brmsfit'
pp_mixture(x, newdata = NULL, re_formula = NULL,
           allow_new_levels = FALSE, sample_new_levels = "uncertainty",
           new_objects = list(), incl_autocor = TRUE, resp = NULL, subset = NULL,
           nsamples = NULL, nug = NULL, summary = TRUE, robust = FALSE,
```

```

  probs = c(0.025, 0.975), log = FALSE, ...)

pp_mixture(x, ...)

```

Arguments

<code>x</code>	An R object usually of class <code>brmsfit</code> .
<code>newdata</code>	An optional <code>data.frame</code> for which to evaluate predictions. If <code>NULL</code> (default), the original data of the model is used.
<code>re_formula</code>	formula containing group-level effects to be considered in the prediction. If <code>NULL</code> (default), include all group-level effects; if <code>NA</code> , include no group-level effects.
<code>allow_new_levels</code>	A flag indicating if new levels of group-level effects are allowed (defaults to <code>FALSE</code>). Only relevant if <code>newdata</code> is provided.
<code>sample_new_levels</code>	Indicates how to sample new levels for grouping factors specified in <code>re_formula</code> . This argument is only relevant if <code>newdata</code> is provided and <code>allow_new_levels</code> is set to <code>TRUE</code> . If "uncertainty" (default), include group-level uncertainty in the predictions based on the variation of the existing levels. If "gaussian", sample new levels from the (multivariate) normal distribution implied by the group-level standard deviations and correlations. This options may be useful for conducting Bayesian power analysis. If "old_levels", directly sample new levels from the existing levels.
<code>new_objects</code>	A named list of objects containing new data, which cannot be passed via argument <code>newdata</code> . Currently, only required for objects passed to <code>cor_sar</code> and <code>cor_fixed</code> .
<code>incl_autocor</code>	A flag indicating if ARMA autocorrelation parameters should be included in the predictions. Defaults to <code>TRUE</code> . Setting it to <code>FALSE</code> will not affect other correlation structures such as <code>cor_bsts</code> , or <code>cor_fixed</code> .
<code>resp</code>	Optional names of response variables. If specified, fitted values of these response variables are returned.
<code>subset</code>	A numeric vector specifying the posterior samples to be used. If <code>NULL</code> (the default), all samples are used.
<code>nsamples</code>	Positive integer indicating how many posterior samples should be used. If <code>NULL</code> (the default) all samples are used. Ignored if <code>subset</code> is not <code>NULL</code> .
<code>nug</code>	Small positive number for Gaussian process terms only. For numerical reasons, the covariance matrix of a Gaussian process might not be positive definite. Adding a very small number to the matrix's diagonal often solves this problem. If <code>NULL</code> (the default), <code>nug</code> is chosen internally.
<code>summary</code>	Should summary statistics (i.e. means, sds, and 95% intervals) be returned instead of the raw values? Default is <code>TRUE</code> .
<code>robust</code>	If <code>FALSE</code> (the default) the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If <code>TRUE</code> , the median and the median absolute deviation (MAD) are applied instead. Only used if <code>summary</code> is <code>TRUE</code> .

probs	The percentiles to be computed by the <code>quantile</code> function. Only used if <code>summary</code> is <code>TRUE</code> .
log	Logical; Indicates whether to return probabilities on the log-scale.
...	Currently ignored.

Details

The returned probabilities can be written as $P(K_n = k|Y_n)$, that is the posterior probability that observation n originates from component k . They are computed using Bayes' Theorem

$$P(K_n = k|Y_n) = P(Y_n|K_n = k)P(K_n = k)/P(Y_n),$$

where $P(Y_n|K_n = k)$ is the (posterior) likelihood of observation n for component k , $P(K_n = k)$ is the (posterior) mixing probability of component k (i.e. parameter `theta<k>`), and

$$P(Y_n) = \sum_{k=1, \dots, K} P(Y_n|K_n = k)P(K_n = k)$$

is a normalizing constant.

Value

If `summary = TRUE`, an $N \times E \times K$ array, where N is the number of observations, K is the number of mixture components, and E is equal to `length(probs) + 2`. If `summary = FALSE`, an $S \times N \times K$ array, where S is the number of posterior samples.

Examples

```
## Not run:
## simulate some data
set.seed(1234)
dat <- data.frame(
  y = c(rnorm(100), rnorm(50, 2)),
  x = rnorm(150)
)
## fit a simple normal mixture model
mix <- mixture(gaussian, nmix = 2)
prior <- c(
  prior(normal(0, 5), Intercept, nlpar = mu1),
  prior(normal(0, 5), Intercept, nlpar = mu2),
  prior(dirichlet(2, 2), theta)
)
fit1 <- brm(bf(y ~ x), dat, family = mix,
            prior = prior, chains = 2, inits = 0)
summary(fit1)

## compute the membership probabilities
ppm <- pp_mixture(fit1)
str(ppm)

## extract point estimates for each observation
head(ppm[, 1, ])
```

```
## classify every observation according to
## the most likely component
apply(ppm[, 1, ], 1, which.max)

## End(Not run)
```

predict.brmsfit

Model Predictions of brmsfit Objects

Description

Predict responses based on the fitted model. Can be performed for the data used to fit the model (posterior predictive checks) or for new data. By definition, these predictions have higher variance than predictions of the fitted values (i.e., the 'regression line') performed by the `fitted` method. This is because the measurement error is incorporated. The estimated means of both methods should, however, be very similar.

Usage

```
## S3 method for class 'brmsfit'
predict(object, newdata = NULL, re_formula = NULL,
        transform = NULL, allow_new_levels = FALSE,
        sample_new_levels = "uncertainty", new_objects = list(),
        incl_autocor = TRUE, resp = NULL, negative_rt = FALSE, subset = NULL,
        nsamples = NULL, sort = FALSE, nug = NULL, ntrys = 5,
        summary = TRUE, robust = FALSE, probs = c(0.025, 0.975), ...)

## S3 method for class 'brmsfit'
posterior_predict(object, newdata = NULL,
                  re_formula = NULL, re.form = NULL, transform = NULL,
                  allow_new_levels = FALSE, sample_new_levels = "uncertainty",
                  new_objects = list(), incl_autocor = TRUE, resp = NULL,
                  negative_rt = FALSE, subset = NULL, nsamples = NULL, sort = FALSE,
                  nug = NULL, ntrys = 5, robust = FALSE, probs = c(0.025, 0.975), ...)
```

Arguments

<code>object</code>	An object of class <code>brmsfit</code>
<code>newdata</code>	An optional <code>data.frame</code> for which to evaluate predictions. If <code>NULL</code> (default), the original data of the model is used.
<code>re_formula</code>	formula containing group-level effects to be considered in the prediction. If <code>NULL</code> (default), include all group-level effects; if <code>NA</code> , include no group-level effects.
<code>transform</code>	A function or a character string naming a function to be applied on the predicted responses before summary statistics are computed.

<code>allow_new_levels</code>	A flag indicating if new levels of group-level effects are allowed (defaults to FALSE). Only relevant if <code>newdata</code> is provided.
<code>sample_new_levels</code>	Indicates how to sample new levels for grouping factors specified in <code>re_formula</code> . This argument is only relevant if <code>newdata</code> is provided and <code>allow_new_levels</code> is set to TRUE. If "uncertainty" (default), include group-level uncertainty in the predictions based on the variation of the existing levels. If "gaussian", sample new levels from the (multivariate) normal distribution implied by the group-level standard deviations and correlations. This options may be useful for conducting Bayesian power analysis. If "old_levels", directly sample new levels from the existing levels.
<code>new_objects</code>	A named list of objects containing new data, which cannot be passed via argument <code>newdata</code> . Currently, only required for objects passed to <code>cor_sar</code> and <code>cor_fixed</code> .
<code>incl_autocor</code>	A flag indicating if ARMA autocorrelation parameters should be included in the predictions. Defaults to TRUE. Setting it to FALSE will not affect other correlation structures such as <code>cor_bsts</code> , or <code>cor_fixed</code> .
<code>resp</code>	Optional names of response variables. If specified, fitted values of these response variables are returned.
<code>negative_rt</code>	Only relevant for Wiener diffusion models. A flag indicating whether response times of responses on the lower boundary should be returned as negative values. This allows to distinguish responses on the upper and lower boundary. Defaults to FALSE.
<code>subset</code>	A numeric vector specifying the posterior samples to be used. If NULL (the default), all samples are used.
<code>nsamples</code>	Positive integer indicating how many posterior samples should be used. If NULL (the default) all samples are used. Ignored if <code>subset</code> is not NULL.
<code>sort</code>	Logical. Only relevant for time series models. Indicating whether to return predicted values in the original order (FALSE; default) or in the order of the time series (TRUE).
<code>nug</code>	Small positive number for Gaussian process terms only. For numerical reasons, the covariance matrix of a Gaussian process might not be positive definite. Adding a very small number to the matrix's diagonal often solves this problem. If NULL (the default), <code>nug</code> is chosen internally.
<code>ntrys</code>	Parameter used in rejection sampling for truncated discrete models only (defaults to 5). See Details for more information.
<code>summary</code>	Should summary statistics (i.e. means, sds, and 95% intervals) be returned instead of the raw values? Default is TRUE.
<code>robust</code>	If FALSE (the default) the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If TRUE, the median and the median absolute deviation (MAD) are applied instead. Only used if <code>summary</code> is TRUE.
<code>probs</code>	The percentiles to be computed by the <code>quantile</code> function. Only used if <code>summary</code> is TRUE.


```
...           Currently ignored.
re.form       Alias of re_formula.
```

Details

NA values within factors in `newdata`, are interpreted as if all dummy variables of this factor are zero. This allows, for instance, to make predictions of the grand mean when using sum coding.

Method `posterior_predict.brmsfit` is an alias of `predict.brmsfit` with `summary = FALSE`.

For truncated discrete models only: In the absence of any general algorithm to sample from truncated discrete distributions, rejection sampling is applied in this special case. This means that values are sampled until a value lies within the defined truncation boundaries. In practice, this procedure may be rather slow (especially in R). Thus, we try to do approximate rejection sampling by sampling each value `ntrys` times and then select a valid value. If all values are invalid, the closest boundary is used, instead. If there are more than a few of these pathological cases, a warning will occur suggesting to increase argument `ntrys`.

Value

Predicted values of the response variable. If `summary = TRUE` the output depends on the family: For categorical and ordinal families, it is a $N \times C$ matrix, where N is the number of observations and C is the number of categories. For all other families, it is a $N \times E$ matrix where E is equal to `length(probs) + 2`. If `summary = FALSE`, the output is as a $S \times N$ matrix, where S is the number of samples. In multivariate models, the output is an array of 3 dimensions, where the third dimension indicates the response variables.

Examples

```
## Not run:
## fit a model
fit <- brm(time | cens(censored) ~ age + sex + (1+age||patient),
           data = kidney, family = "exponential", inits = "0")

## predicted responses
pp <- predict(fit)
head(pp)

## predicted responses excluding the group-level effect of age
pp2 <- predict(fit, re_formula = ~ (1|patient))
head(pp2)

## predicted responses of patient 1 for new data
newdata <- data.frame(sex = factor(c("male", "female")),
                    age = c(20, 50),
                    patient = c(1, 1))
predict(fit, newdata = newdata)

## End(Not run)
```

```
print.brmsfit          Print a summary for a fitted model represented by a brmsfit object
```

Description

Print a summary for a fitted model represented by a `brmsfit` object

Usage

```
## S3 method for class 'brmsfit'
print(x, digits = 2, ...)
```

Arguments

<code>x</code>	An object of class <code>brmsfit</code>
<code>digits</code>	The number of significant digits for printing out the summary; defaults to 2. The effective sample size is always rounded to integers.
<code>...</code>	Additional arguments that would be passed to method <code>summary</code> of <code>brmsfit</code> .

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

```
print.brmsprior      Print method for brmsprior objects
```

Description

Print method for `brmsprior` objects

Usage

```
## S3 method for class 'brmsprior'
print(x, show_df, ...)
```

Arguments

<code>x</code>	An object of class <code>brmsprior</code> .
<code>show_df</code>	Logical; Print priors as a single data frame (TRUE) or as a sequence of sampling statements (FALSE)?
<code>...</code>	Currently ignored.

prior_samples.brmsfit *Extract prior samples*

Description

Extract prior samples of specified parameters

Usage

```
## S3 method for class 'brmsfit'
prior_samples(x, pars = NA, ...)

prior_samples(x, pars = NA, ...)
```

Arguments

x	An R object typically of class brmsfit
pars	Names of parameters for which prior samples should be returned, as given by a character vector or regular expressions. By default, all prior samples are extracted
...	Currently ignored

Details

To make use of this function, the model must contain samples of prior distributions. This can be ensured by setting `sample_prior = TRUE` in function `brm`. Currently there are methods for `brmsfit` objects.

Value

A data frame containing the prior samples.

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

Examples

```
## Not run:
fit <- brm(rating ~ treat + period + carry + (1|subject),
          data = inhaler, family = "cumulative",
          prior = set_prior("normal(0,2)", class = "b"),
          sample_prior = TRUE)

# extract all prior samples
samples1 <- prior_samples(fit)
head(samples1)
```

```
# extract prior samples for the population-level effects of 'treat'
samples2 <- posterior_samples(fit, "b_treat")
head(samples2)

## End(Not run)
```

prior_summary.brmsfit *Extract Priors of a Bayesian Model Fitted with brms*

Description

Extract Priors of a Bayesian Model Fitted with **brms**

Usage

```
## S3 method for class 'brmsfit'
prior_summary(object, all = TRUE, ...)
```

Arguments

object	A brmsfit object
all	Logical; Show all parameters in the model which may have priors (TRUE) or only those with proper priors (FALSE)?
...	Further arguments passed to or from other methods.

Value

For brmsfit objects, an object of class brmsprior.

Examples

```
## Not run:
fit <- brm(count ~ log_Age_c + log_Base4_c * Trt_c
  + (1|patient) + (1|obs),
  data = epilepsy, family = poisson(),
  prior = c(prior(student_t(5,0,10), class = b),
    prior(cauchy(0,2), class = sd))

prior_summary(fit)
prior_summary(fit, all = FALSE)
print(prior_summary(fit, all = FALSE), show_df = FALSE)

## End(Not run)
```

ranef.brmsfit	<i>Extract Group-Level Estimates</i>
---------------	--------------------------------------

Description

Extract the group-level ('random') effects of each level from a brmsfit object.

Usage

```
## S3 method for class 'brmsfit'
ranef(object, summary = TRUE, robust = FALSE,
       probs = c(0.025, 0.975), ...)
```

Arguments

object	An object of class brmsfit.
summary	Should summary statistics (i.e. means, sds, and 95% intervals) be returned instead of the raw values? Default is TRUE.
robust	If FALSE (the default) the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If TRUE, the median and the median absolute deviation (MAD) are applied instead. Only used if summary is TRUE.
probs	The percentiles to be computed by the quantile function. Only used if summary is TRUE.
...	Currently ignored.

Value

If old is FALSE: A list of arrays (one per grouping factor). If summary is TRUE, names of the first dimension are the factor levels and names of the third dimension are the group-level effects. If summary is FALSE, names of the second dimension are the factor levels and names of the third dimension are the group-level effects.

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

Examples

```
## Not run:
fit <- brm(count ~ log_Age_c + log_Base4_c * Trt_c + (1+Trt_c|visit),
          data = epilepsy, family = gaussian(), chains = 2)
ranef(fit)

## End(Not run)
```

reloo

*Compute exact cross-validation for problematic observations***Description**

Compute exact cross-validation for problematic observations for which approximate leave-one-out cross-validation may return incorrect results.

Usage

```
reloo(x, ...)

## S3 method for class 'loo'
reloo(x, fit, k_threshold = 0.7, check = TRUE, resp = NULL,
      ...)
```

Arguments

<code>x</code>	An R object typically of class <code>loo</code> .
<code>...</code>	Further arguments passed to <code>update.brmsfit</code> such as <code>iter</code> , <code>chains</code> , or <code>cores</code> .
<code>fit</code>	An R object typically of class <code>brmsfit</code> .
<code>k_threshold</code>	The threshold at which pareto k estimates are treated as problematic. Defaults to 0.7. See pareto_k_ids for more details.
<code>check</code>	Logical; If TRUE (the default), a crude check is performed if the <code>loo</code> object was generated from the <code>brmsfit</code> object passed to argument <code>fit</code> .
<code>resp</code>	Optional names of response variables. If specified, fitted values of these response variables are returned.

Details

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 (2017) and the `loo` package documentation). If there are J observations with k estimates above `k_threshold`, then `reloo` 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 original `loo` object.

Value

An object of the class as `x`.

See Also

[loo](#), [kfold](#)

Examples

```
## Not run:
fit1 <- brm(count ~ log_Age_c + log_Base4_c * Trt + (1|patient),
            data = epilepsy, family = poisson())
# throws warning about some pareto k estimates being too high
(loo1 <- loo(fit1))
(loo1 <- reloo(loo1, fit1))

## End(Not run)
```

residuals.brmsfit *Extract Model Residuals from brmsfit Objects*

Description

Extract Model Residuals from brmsfit Objects

Usage

```
## S3 method for class 'brmsfit'
residuals(object, newdata = NULL, re_formula = NULL,
           type = c("ordinary", "pearson"), method = c("fitted", "predict"),
           allow_new_levels = FALSE, sample_new_levels = "uncertainty",
           new_objects = list(), incl_autocor = TRUE, resp = NULL, subset = NULL,
           nsamples = NULL, sort = FALSE, nug = NULL, summary = TRUE,
           robust = FALSE, probs = c(0.025, 0.975), ...)

## S3 method for class 'brmsfit'
predictive_error(object, newdata = NULL,
                 re_formula = NULL, re.form = NULL, allow_new_levels = FALSE,
                 sample_new_levels = "uncertainty", new_objects = list(),
                 incl_autocor = TRUE, resp = NULL, subset = NULL, nsamples = NULL,
                 sort = FALSE, nug = NULL, robust = FALSE, probs = c(0.025, 0.975),
                 ...)
```

Arguments

object	An object of class brmsfit
newdata	An optional data.frame for which to evaluate predictions. If NULL (default), the original data of the model is used.
re_formula	formula containing group-level effects to be considered in the prediction. If NULL (default), include all group-level effects; if NA, include no group-level effects.
type	The type of the residuals, either "ordinary" or "pearson". More information is provided under 'Details'.

method	Indicates the method to compute model implied values. Either "fitted" (predicted values of the regression curve) or "predict" (predicted response values). Using "predict" is recommended but "fitted" is the current default for reasons of backwards compatibility.
allow_new_levels	A flag indicating if new levels of group-level effects are allowed (defaults to FALSE). Only relevant if newdata is provided.
sample_new_levels	Indicates how to sample new levels for grouping factors specified in re_formula. This argument is only relevant if newdata is provided and allow_new_levels is set to TRUE. If "uncertainty" (default), include group-level uncertainty in the predictions based on the variation of the existing levels. If "gaussian", sample new levels from the (multivariate) normal distribution implied by the group-level standard deviations and correlations. This options may be useful for conducting Bayesian power analysis. If "old_levels", directly sample new levels from the existing levels.
new_objects	A named list of objects containing new data, which cannot be passed via argument newdata. Currently, only required for objects passed to cor_sar and cor_fixed.
incl_autocor	A flag indicating if ARMA autocorrelation parameters should be included in the predictions. Defaults to TRUE. Setting it to FALSE will not affect other correlation structures such as cor_bsts, or cor_fixed.
resp	Optional names of response variables. If specified, fitted values of these response variables are returned.
subset	A numeric vector specifying the posterior samples to be used. If NULL (the default), all samples are used.
nsamples	Positive integer indicating how many posterior samples should be used. If NULL (the default) all samples are used. Ignored if subset is not NULL.
sort	Logical. Only relevant for time series models. Indicating whether to return predicted values in the original order (FALSE; default) or in the order of the time series (TRUE).
nug	Small positive number for Gaussian process terms only. For numerical reasons, the covariance matrix of a Gaussian process might not be positive definite. Adding a very small number to the matrix's diagonal often solves this problem. If NULL (the default), nug is chosen internally.
summary	Should summary statistics (i.e. means, sds, and 95% intervals) be returned instead of the raw values? Default is TRUE.
robust	If FALSE (the default) the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If TRUE, the median and the median absolute deviation (MAD) are applied instead. Only used if summary is TRUE.
probs	The percentiles to be computed by the quantile function. Only used if summary is TRUE.
...	Currently ignored.
re.form	Alias of re_formula.

Details

Residuals of type ordinary are of the form $R = Y - Yp$, where Y is the observed and Yp is the predicted response. Residuals of type pearson are of the form $R = (Y - Yp)/SD(Y)$, where $SD(Y)$ is an estimation of the standard deviation of Y .

Currently, `residuals.brmsfit` does not support categorical or ordinal models.

Method `predictive_error.brmsfit` is an alias of `residuals.brmsfit` with `method = "predict"` and `summary = FALSE`.

Value

Model residuals. If `summary = TRUE` this is a $N \times C$ matrix and if `summary = FALSE` a $S \times N$ matrix, where S is the number of samples, N is the number of observations, and C is equal to `length(probs) + 2`.

Examples

```
## Not run:
## fit a model
fit <- brm(rating ~ treat + period + carry + (1|subject),
          data = inhaler, cores = 2)

## extract residuals
res <- residuals(fit, summary = TRUE)
head(res)

## End(Not run)
```

restructure

Restructure Old brmsfit Objects

Description

Restructure old `brmsfit` objects to work with the latest **brms** version. This function is called internally when applying post-processing methods. However, in order to avoid unnecessary run time caused by the restructuring, I recommend explicitly calling `restructure` once per model after updating **brms**.

Usage

```
restructure(x, rstr_summary = FALSE)
```

Arguments

`x` An object of class `brmsfit`.

`rstr_summary` Logical; If `TRUE`, the cached summary stored by **rstan** is restructured as well.

Value

A `brmsfit` object compatible with the latest version of **brms**.

<code>rows2labels</code>	<i>Convert Rows to Labels</i>
--------------------------	-------------------------------

Description

Convert information in rows to labels for each row.

Usage

```
rows2labels(x, digits = 2, sep = " & ", incl_vars = TRUE, ...)
```

Arguments

<code>x</code>	A <code>data.frame</code> for which to extract labels.
<code>digits</code>	Minimal number of decimal places shown in the labels of numeric variables.
<code>sep</code>	A single character string defining the separator between variables used in the labels.
<code>incl_vars</code>	Indicates if variable names should be part of the labels. Defaults to <code>TRUE</code> .
<code>...</code>	Currently unused.

Value

A character vector of the same length as the number of rows of `x`.

See Also

[make_conditions](#), [codemarginal_effects](#)

<code>s</code>	<i>Defining smooths in brms formulas</i>
----------------	---

Description

Functions used in definition of smooth terms within a model formulas. The function does not evaluate a (spline) smooth - it exists purely to help set up a model using spline based smooths.

Usage

```
s(...)
```

```
t2(...)
```

Arguments

... Arguments passed to `mgcv::s` or `mgcv::t2`.

Details

The function defined here are just simple wrappers of the respective functions of the **mgcv** package.

See Also

[brmsformula](#), [mgcv::s](#), [mgcv::t2](#)

Examples

```
## Not run:
# simulate some data
dat <- mgcv::gamSim(1, n = 200, scale = 2)

# fit univariate smooths for all predictors
fit1 <- brm(y ~ s(x0) + s(x1) + s(x2) + s(x3),
           data = dat, chains = 2)
summary(fit1)
plot(marginal_smooths(fit1), ask = FALSE)

# fit a more complicated smooth model
fit2 <- brm(y ~ t2(x0, x1) + s(x2, by = x3),
           data = dat, chains = 2)
summary(fit2)
plot(marginal_smooths(fit2), ask = FALSE)

## End(Not run)
```

set_prior

Prior Definitions for brms Models

Description

Define priors for specific parameters or classes of parameters

Usage

```
set_prior(prior, class = "b", coef = "", group = "", resp = "",
         dpar = "", nlpar = "", lb = NA, ub = NA, check = TRUE)

prior(prior, ...)

prior_(prior, ...)

prior_string(prior, ...)
```

Arguments

prior	A character string defining a distribution in Stan language
class	The parameter class. Defaults to "b" (i.e. population-level effects). See 'Details' for other valid parameter classes.
coef	Name of the (population- or group-level) parameter.
group	Grouping factor of group-level parameters.
resp	Name of the response variable / category. Only used in multivariate and categorical models.
dpar	Name of a distributional parameter. Only used in distributional models.
nlprior	Name of a non-linear parameter. Only used in non-linear models.
lb	Lower bound for parameter restriction. Currently only allowed for classes "b", "ar", "ma", and "arr". Defaults to NULL, that is no restriction.
ub	Upper bound for parameter restriction. Currently only allowed for classes "b", "ar", "ma", and "arr". Defaults to NULL, that is no restriction.
check	Logical; Indicates whether priors should be checked for validity (as far as possible). Defaults to TRUE. If FALSE, prior is passed to the Stan code as is, and all other arguments are ignored.
...	Arguments passed to set_prior.

Details

set_prior is used to define prior distributions for parameters in **brms** models. The functions prior, prior_, and prior_string are aliases of set_prior each allowing for a different kind of argument specification. prior allows specifying arguments as expression without quotation marks using non-standard evaluation. prior_ allows specifying arguments as one-sided formulas or wrapped in quote. prior_string allows specifying arguments as strings just as set_prior itself.

Below, we explain its usage and list some common prior distributions for parameters. A complete overview on possible prior distributions is given in the Stan Reference Manual available at <http://mc-stan.org/>.

To combine multiple priors, use c(...) or the + operator (see 'Examples'). **brms** does not check if the priors are written in correct **Stan** language. Instead, **Stan** will check their syntactical correctness when the model is parsed to C++ and returns an error if they are not. This, however, does not imply that priors are always meaningful if they are accepted by **Stan**. Although **brms** tries to find common problems (e.g., setting bounded priors on unbounded parameters), there is no guarantee that the defined priors are reasonable for the model. Below, we list the types of parameters in **brms** models, for which the user can specify prior distributions.

1. Population-level ('fixed') effects

Every Population-level effect has its own regression parameter represents the name of the corresponding population-level effect. Suppose, for instance, that y is predicted by x_1 and x_2 (i.e., $y \sim x_1 + x_2$ in formula syntax). Then, x_1 and x_2 have regression parameters b_{x_1} and b_{x_2} respectively. The default prior for population-level effects (including monotonic and category specific effects) is an improper flat prior over the reals. Other common options are normal priors or student-t priors. If we want to have a normal prior with mean 0 and standard deviation

5 for x_1 , and a unit student-t prior with 10 degrees of freedom for x_2 , we can specify this via `set_prior("normal(0,5)", class = "b", coef = "x1")` and `set_prior("student_t(10,0,1)", class = "b", coef = "x2")`. To put the same prior on all population-level effects at once, we may write as a shortcut `set_prior("<prior>", class = "b")`. This also leads to faster sampling, because priors can be vectorized in this case. Both ways of defining priors can be combined using for instance `set_prior("normal(0,2)", class = "b")` and `set_prior("normal(0,10)", class = "b", coef = "x1")` at the same time. This will set a $\text{normal}(0,10)$ prior on the effect of x_1 and a $\text{normal}(0,2)$ prior on all other population-level effects. However, this will break vectorization and may slow down the sampling procedure a bit.

In case of the default intercept parameterization (discussed in the 'Details' section of [brmsformula](#)), general priors on class "b" will *not* affect the intercept. Instead, the intercept has its own parameter class named "Intercept" and priors can thus be specified via `set_prior("<prior>", class = "Intercept")`. Setting a prior on the intercept will not break vectorization of the other population-level effects. Note that technically, this prior is set on an intercept that results when internally centering all population-level predictors around zero to improve sampling efficiency. On this centered intercept, specifying a prior is actually much easier and intuitive than on the original intercept, since the former represents the expected response value when all predictors are at their means. To treat the intercept as an ordinary population-level effect and avoid the centering parameterization, use `0 + intercept` on the right-hand side of the model formula.

A special shrinkage prior to be applied on population-level effects is the horseshoe prior. See [horseshoe](#) for details. Another shrinkage prior is the so-called lasso prior. See [lasso](#) for details.

In non-linear models, population-level effects are defined separately for each non-linear parameter. Accordingly, it is necessary to specify the non-linear parameter in `set_prior` so that priors we can be assigned correctly. If, for instance, α is the parameter and x the predictor for which we want to define the prior, we can write `set_prior("<prior>", coef = "x", nlpar = "alpha")`. As a shortcut we can use `set_prior("<prior>", nlpar = "alpha")` to set the same prior on all population-level effects of α at once.

If desired, population-level effects can be restricted to fall only within a certain interval using the `lb` and `ub` arguments of `set_prior`. This is often required when defining priors that are not defined everywhere on the real line, such as uniform or gamma priors. When defining a `uniform(2,4)` prior, you should write `set_prior("uniform(2,4)", lb = 2, ub = 4)`. When using a prior that is defined on the positive reals only (such as a gamma prior) set `lb = 0`. In most situations, it is not useful to restrict population-level parameters through bounded priors (non-linear models are an important exception), but if you really want to this is the way to go.

2. Standard deviations of group-level ('random') effects

Each group-level effect of each grouping factor has a standard deviation named `sd_<group>_<coef>`. Consider, for instance, the formula $y \sim x_1 + x_2 + (1 + x_1 \mid g)$. We see that the intercept as well as x_1 are group-level effects nested in the grouping factor g . The corresponding standard deviation parameters are named as `sd_g_Intercept` and `sd_g_x1` respectively. These parameters are restricted to be non-negative and, by default, have a half student-t prior with 3 degrees of freedom and a scale parameter that depends on the standard deviation of the response after applying the link function. Minimally, the scale parameter is 10. This prior is used (a) to be only very weakly informative in order to influence results as few as possible, while (b) providing at least some regularization to considerably improve convergence and sampling efficiency. To define a prior distribution only for standard deviations of a specific grouping factor, use `set_prior("<prior>", class = "sd", group = "<group>")`. To define a prior distribution only for a specific standard deviation of a specific grouping factor, you may write

`set_prior("<prior>", class = "sd", group = "<group>", coef = "<coef>")`. Recommendations on useful prior distributions for standard deviations are given in Gelman (2006), but note that he is no longer recommending uniform priors, anymore.

When defining priors on group-level parameters in non-linear models, please make sure to specify the corresponding non-linear parameter through the `nlpar` argument in the same way as for population-level effects.

3. Correlations of group-level ('random') effects

If there is more than one group-level effect per grouping factor, the correlations between those effects have to be estimated. The prior `"lkj_corr_cholesky(eta)"` or in short `"lkj(eta)"` with $\eta > 0$ is essentially the only prior for (Cholesky factors) of correlation matrices. If $\eta = 1$ (the default) all correlation matrices are equally likely a priori. If $\eta > 1$, extreme correlations become less likely, whereas $0 < \eta < 1$ results in higher probabilities for extreme correlations. Correlation matrix parameters in `brms` models are named as `cor_<group>`, (e.g., `cor_g` if `g` is the grouping factor). To set the same prior on every correlation matrix, use for instance `set_prior("lkj(2)", class = "cor")`. Internally, the priors are transformed to be put on the Cholesky factors of the correlation matrices to improve efficiency and numerical stability. The corresponding parameter class of the Cholesky factors is `L`, but it is not recommended to specify priors for this parameter class directly.

4. Smooth terms (splines)

GAMMs are implemented in `brms` using the 'random effects' formulation of smooth terms (for details see [gamm](#)). Thus, each smooth term has its corresponding standard deviation modeling the variability within this term. In `brms`, this parameter class is called `sds` and priors can be specified via `set_prior("<prior>", class = "sds", coef = "<term label>")`. The default prior is the same as for standard deviations of group-level effects.

5. Gaussian processes

Gaussian processes as currently implemented in `brms` have two parameters, the standard deviation parameter `sdgp`, and characteristic length-scale parameter `lscale` (see [gp](#) for more details). The default prior of `sdgp` is the same as for standard deviations of group-level effects. The default prior of `lscale` is an informative inverse-gamma prior specifically tuned to the covariates of the Gaussian process (for more details see https://betanalpha.github.io/assets/case_studies/gp_part3/part3.html). This tuned prior may be overly informative in some cases, so please consider other priors as well to make sure inference is robust to the prior specification. If tuning fails, a half-normal prior is used instead.

6. Autocorrelation parameters

The autocorrelation parameters currently implemented are named `ar` (autoregression), `ma` (moving average), and `arr` (autoregression of the response).

Priors can be defined by `set_prior("<prior>", class = "ar")` for `ar` and similar for `ma` and `arr` effects. By default, `ar` and `ma` are bounded between -1 and 1 and `arr` is unbounded (you may change this by using the arguments `lb` and `ub`). The default prior is flat over the definition area.

7. Distance parameters of monotonic effects

As explained in the details section of [brm](#), monotonic effects make use of a special parameter vector to estimate the 'normalized distances' between consecutive predictor categories. This is realized in `Stan` using the `simplex` parameter type. This class is named `"simo"` (short for simplex monotonic) in `brms`. The only valid prior for simplex parameters is the `dirichlet` prior, which accepts a vector

of length $K - 1$ ($K =$ number of predictor categories) as input defining the ‘concentration’ of the distribution. Explaining the dirichlet prior is beyond the scope of this documentation, but we want to describe how to define this prior syntactically correct. If a predictor x with K categories is modeled as monotonic, we can define a prior on its corresponding simplex via `prior(dirichlet(<vector>), class = simo, coef = mox1)`. The 1 in the end of `coef` indicates that this is the first simplex in this term. If interactions between multiple monotonic variables are modeled, multiple simplexes per term are required. For `<vector>`, we can put in any R expression defining a vector of length $K - 1$. The default is a uniform prior (i.e. `<vector> = rep(1, K-1)`) over all simplexes of the respective dimension.

8. Parameters for specific families

Some families need additional parameters to be estimated. Families `gaussian`, `student`, `skew_normal`, `lognormal`, and `gen_extreme_value` need the parameter `sigma` to account for the residual standard deviation. By default, `sigma` has a half student-t prior that scales in the same way as the group-level standard deviations. Further, family `student` needs the parameter `nu` representing the degrees of freedom of students-t distribution. By default, `nu` has prior `"gamma(2, 0.1)"` and a fixed lower bound of 1. Families `gamma`, `weibull`, `inverse.gaussian`, and `negbinomial` need a shape parameter that has a `"gamma(0.01, 0.01)"` prior by default. For families `cumulative`, `cratio`, `sratio`, and `acat`, and only if `threshold = "equidistant"`, the parameter `delta` is used to model the distance between two adjacent thresholds. By default, `delta` has an improper flat prior over the reals. The `von_mises` family needs the parameter `kappa`, representing the concentration parameter. By default, `kappa` has prior `"gamma(2, 0.01)"`.

Every family specific parameter has its own prior class, so that `set_prior("<prior>", class = "<parameter>")` is the right way to go. All of these priors are chosen to be weakly informative, having only minimal influence on the estimations, while improving convergence and sampling efficiency.

Often, it may not be immediately clear, which parameters are present in the model. To get a full list of parameters and parameter classes for which priors can be specified (depending on the model) use function `get_prior`.

Value

An object of class `brmsprior` to be used in the `prior` argument of `brm`.

Functions

- `prior`: Alias of `set_prior` allowing to specify arguments as expressions without quotation marks.
- `prior_`: Alias of `set_prior` allowing to specify arguments as as one-sided formulas or wrapped in quote.
- `prior_string`: Alias of `set_prior` allowing to specify arguments as strings.

References

Gelman A. (2006). Prior distributions for variance parameters in hierarchical models. *Bayesian analysis*, 1(3), 515 – 534.

See Also

`get_prior`

Examples

```

## use alias functions
(prior1 <- prior(cauchy(0, 1), class = sd))
(prior2 <- prior_(~cauchy(0, 1), class = ~sd))
(prior3 <- prior_string("cauchy(0, 1)", class = "sd"))
identical(prior1, prior2)
identical(prior1, prior3)

## check which parameters can have priors
get_prior(rating ~ treat + period + carry + (1|subject),
          data = inhaler, family = cumulative())

## define some priors
prior <- c(prior_string("normal(0,10)", class = "b"),
          prior(normal(1,2), class = b, coef = treat),
          prior_(~cauchy(0,2), class = ~sd,
                group = ~subject, coef = ~Intercept))

## verify that the priors indeed found their way into Stan's model code
make_stancode(rating ~ treat + period + carry + (1|subject),
              data = inhaler, family = cumulative(),
              prior = prior)

## use the horseshoe prior to model sparsity in population-level effects
make_stancode(count ~ log_Age_c + log_Base4_c * Trt_c,
              data = epilepsy, family = poisson(),
              prior = set_prior("horseshoe(3)"))

## alternatively use the lasso prior
make_stancode(count ~ log_Age_c + log_Base4_c * Trt_c,
              data = epilepsy, family = poisson(),
              prior = set_prior("lasso(1)"))

## pass priors to Stan without checking
prior <- prior_string("target += normal_lpdf(b[1] | 0, 1)", check = FALSE)
make_stancode(count ~ Trt_c, data = epilepsy, prior = prior)

```

Shifted_Lognormal

The Shifted Log Normal Distribution

Description

Density, distribution function, quantile function and random generation for the shifted log normal distribution with mean `meanlog`, standard deviation `sdlog`, and shift parameter `shift`.

Usage

```
dshifted_lnorm(x, meanlog = 0, sdlog = 1, shift = 0, log = FALSE)
```



```
pshifted_lnorm(q, meanlog = 0, sdlog = 1, shift = 0, lower.tail = TRUE,
  log.p = FALSE)
```

```
qshifted_lnorm(p, meanlog = 0, sdlog = 1, shift = 0, lower.tail = TRUE,
  log.p = FALSE)
```

```
rshifted_lnorm(n, meanlog = 0, sdlog = 1, shift = 0)
```

Arguments

<code>x, q</code>	Vector of quantiles.
<code>meanlog</code>	Vector of means.
<code>sdlog</code>	Vector of standard deviations.
<code>shift</code>	Vector of shifts.
<code>log</code>	Logical; If TRUE, values are returned on the log scale.
<code>lower.tail</code>	Logical; If TRUE (default), return $P(X \leq x)$. Else, return $P(X > x)$.
<code>log.p</code>	Logical; If TRUE, values are returned on the log scale.
<code>p</code>	Vector of probabilities.
<code>n</code>	Number of samples to draw from the distribution.

Details

See vignette("brms_families") for details on the parameterization.

SkewNormal

The Skew-Normal Distribution

Description

Density, distribution function, and random generation for the skew-normal distribution with mean μ , standard deviation σ , and skewness α .

Usage

```
dskew_normal(x, mu = 0, sigma = 1, alpha = 0, xi = NULL, omega = NULL,
  log = FALSE)
```

```
pskew_normal(q, mu = 0, sigma = 1, alpha = 0, xi = NULL, omega = NULL,
  lower.tail = TRUE, log.p = FALSE)
```

```
qskew_normal(p, mu = 0, sigma = 1, alpha = 0, xi = NULL, omega = NULL,
  lower.tail = TRUE, log.p = FALSE, tol = 1e-08)
```

```
rskew_normal(n, mu = 0, sigma = 1, alpha = 0, xi = NULL, omega = NULL)
```

Arguments

<code>x, q</code>	Vector of quantiles.
<code>mu</code>	Vector of mean values.
<code>sigma</code>	Vector of standard deviation values.
<code>alpha</code>	Vector of skewness values.
<code>xi</code>	Optional vector of location values. If NULL (the default), will be computed internally.
<code>omega</code>	Optional vector of scale values. If NULL (the default), will be computed internally.
<code>log</code>	Logical; If TRUE, values are returned on the log scale.
<code>lower.tail</code>	Logical; If TRUE (default), return $P(X \leq x)$. Else, return $P(X > x)$.
<code>log.p</code>	Logical; If TRUE, values are returned on the log scale.
<code>p</code>	Vector of probabilities.
<code>tol</code>	Tolerance of the approximation used in the computation of quantiles.
<code>n</code>	Number of samples to draw from the distribution.

Details

See `vignette("brms_families")` for details on the parameterization.

<code>stancode.brmsfit</code>	<i>Extract Stan model code</i>
-------------------------------	--------------------------------

Description

Extract Stan model code

Usage

```
## S3 method for class 'brmsfit'
stancode(object, version = TRUE, ...)

stancode(object, ...)
```

Arguments

<code>object</code>	An object of class <code>brmsfit</code>
<code>version</code>	Logical; indicates if the first line containing the brms version number should be included. Defaults to TRUE.
<code>...</code>	Currently ignored

Value

Stan model code for further processing.

standata.brmsfit	<i>Extract Data passed to Stan</i>
------------------	------------------------------------

Description

Extract all data that was used by Stan to fit the model

Usage

```
## S3 method for class 'brmsfit'
standata(object, newdata = NULL, re_formula = NULL,
  incl_autocor = TRUE, new_objects = list(), internal = FALSE,
  control = list(), ...)

standata(object, ...)
```

Arguments

object	An object of class <code>brmsfit</code> .
newdata	An optional <code>data.frame</code> for which to evaluate predictions. If <code>NULL</code> (default), the original data of the model is used.
re_formula	formula containing group-level effects to be considered in the prediction. If <code>NULL</code> (default), include all group-level effects; if <code>NA</code> , include no group-level effects.
incl_autocor	A flag indicating if ARMA autocorrelation parameters should be included in the predictions. Defaults to <code>TRUE</code> . Setting it to <code>FALSE</code> will not affect other correlation structures such as <code>cor_bsts</code> , or <code>cor_fixed</code> .
new_objects	A named list of objects containing new data, which cannot be passed via argument <code>newdata</code> . Currently, only required for objects passed to <code>cor_sar</code> and <code>cor_fixed</code> .
internal	Logical, indicates if the data should be prepared for internal use in other post-processing methods.
control	A named list currently for internal usage only.
...	More arguments passed to <code>make_standata</code> .

Value

A named list containing the data originally passed to Stan.

stanplot.brmsfit *MCMC Plots Implemented in **bayesplot***

Description

Convenient way to call MCMC plotting functions implemented in the **bayesplot** package.

Usage

```
## S3 method for class 'brmsfit'
stanplot(object, pars = NA, type = "intervals",
  exact_match = FALSE, ...)

stanplot(object, ...)
```

Arguments

object	An R object typically of class <code>brmsfit</code>
pars	Names of parameters to be plotted, as given by a character vector or regular expressions. By default, all parameters except for group-level and smooth effects are plotted. May be ignored for some plots.
type	The type of the plot. Supported types are (as names) <code>hist</code> , <code>dens</code> , <code>hist_by_chain</code> , <code>dens_overlay</code> , <code>violin</code> , <code>intervals</code> , <code>areas</code> , <code>acf</code> , <code>acf_bar</code> , <code>trace</code> , <code>trace_highlight</code> , <code>scatter</code> , <code>rhat</code> , <code>rhat_hist</code> , <code>neff</code> , <code>neff_hist</code> , <code>nuts_acceptance</code> , <code>nuts_divergence</code> , <code>nuts_stepsize</code> , <code>nuts_treedepth</code> , and <code>nuts_energy</code> . For an overview on the various plot types see MCMC-overview .
exact_match	Indicates whether parameter names should be matched exactly or treated as regular expression. Default is <code>FALSE</code> .
...	Additional arguments passed to the plotting functions. See MCMC-overview for more details.

Details

Also consider using the **shinystan** package available via method `launch_shinystan`. in **brms** for flexible and interactive visual analysis.

Value

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

Examples

```
## Not run:
model <- brm(count ~ log_Age_c + log_Base4_c * Trt
  + (1|patient) + (1|visit),
  data = epilepsy, family = "poisson")
```

```

# plot posterior intervals
stanplot(model)

# only show population-level effects in the plots
stanplot(model, pars = "^b_")

# show histograms of the posterior distributions
stanplot(model, type = "hist")

# plot some diagnostics of the sampler
stanplot(model, type = "neff")
stanplot(model, type = "rhat")

# plot some diagnostics specific to the NUTS sampler
stanplot(model, type = "nuts_acceptance")
stanplot(model, type = "nuts_divergence")

## End(Not run)

```

stanvar

User-defined variables passed to Stan

Description

Prepare user-defined variables to be passed to Stan's data block. This is primarily useful for defining more complex priors and for refitting models without recompilation despite changing priors.

Usage

```
stanvar(x, name = NULL, scode = NULL)
```

Arguments

x	An R object containing data to be passed to Stan.
name	Optional character string providing the desired variable name of the object in x. If NULL (the default) the variable name is directly inferred from x.
scode	Optional line of Stan code to define the variable in Stan's data block. If NULL (the default), the Stan code is inferred based on the class of x.

Value

An object of class `stanvars`.

Examples

```

bprior <- prior(normal(mean_intercept, 10), class = "Intercept")
stanvars <- stanvar(5, name = "mean_intercept")
make_stancode(count ~ Trt, epilepsy, prior = bprior,
              stanvars = stanvars)

# define a multi-normal prior with known covariance matrix
bprior <- prior(multi_normal(M, V), class = "b")
stanvars <- stanvar(rep(0, 2), "M", scode = " vector[K] M;") +
  stanvar(diag(2), "V", scode = " matrix[K, K] V;")
make_stancode(count ~ Trt + log_Base4_c, epilepsy,
              prior = bprior, stanvars = stanvars)

```

StudentT

*The Student-t Distribution***Description**

Density, distribution function, quantile function and random generation for the Student-t distribution with location μ , scale σ , and degrees of freedom df .

Usage

```

dstudent_t(x, df, mu = 0, sigma = 1, log = FALSE)

pstudent_t(q, df, mu = 0, sigma = 1, lower.tail = TRUE, log.p = FALSE)

qstudent_t(p, df, mu = 0, sigma = 1)

rstudent_t(n, df, mu = 0, sigma = 1)

```

Arguments

<code>x, q</code>	Vector of quantiles.
<code>df</code>	Vector of degrees of freedom.
<code>mu</code>	Vector of location values.
<code>sigma</code>	Vector of scale values.
<code>log, log.p</code>	Logical; If TRUE, values are returned on the log scale.
<code>lower.tail</code>	Logical; If TRUE (default), return $P(X \leq x)$. Else, return $P(X > x)$.
<code>p</code>	Vector of probabilities.
<code>n</code>	Number of samples to draw from the distribution.

Details

See vignette("brms_families") for details on the parameterization.

See Also[TDist](#)

summary.brmsfit	<i>Create a summary of a fitted model represented by a brmsfit object</i>
-----------------	---

Description

Create a summary of a fitted model represented by a brmsfit object

Usage

```
## S3 method for class 'brmsfit'  
summary(object, priors = FALSE, prob = 0.95,  
        mc_se = FALSE, use_cache = TRUE, ...)
```

Arguments

object	An object of class brmsfit
priors	Logical; Indicating if priors should be included in the summary. Default is FALSE.
prob	A value between 0 and 1 indicating the desired probability to be covered by the uncertainty intervals. The default is 0.95.
mc_se	Logical; Indicating if the uncertainty caused by the MCMC sampling should be shown in the summary. Defaults to FALSE.
use_cache	Logical; Indicating if summary results should be cached for future use by rstan . Defaults to TRUE. For models fitted with earlier versions of brms , it may be necessary to set use_cache to FALSE in order to get the summary method working correctly.
...	Other potential arguments

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

`theme_black`*Black Theme for **ggplot2** Graphics*

Description

A black theme for ggplot graphics inspired by a blog post of Jon Lefcheck (<https://jonlefccheck.net/2013/03/11/black-theme-for-ggplot2-2/>).

Usage

```
theme_black(base_size = 12, base_family = "")
```

Arguments

<code>base_size</code>	base font size
<code>base_family</code>	base font family

Details

When using `theme_black` in plots powered by the **bayesplot** package such as `pp_check` or `stanplot`, I recommend using the "viridisC" color scheme (see examples).

Value

A theme object used in **ggplot2** graphics.

Examples

```
## Not run:
# change default ggplot theme
theme_set(theme_black())

# change default bayesplot color scheme
bayesplot::color_scheme_set("viridisC")

# fit a simple model
fit <- brm(count ~ log_Age_c + log_Base4_c * Trt + (1|patient),
           data = epilepsy, family = poisson(), chains = 2)
summary(fit)

# create various plots
plot(marginal_effects(fit), ask = FALSE)
pp_check(fit)
stanplot(fit, type = "hex", pars = c("b_Intercept", "b_Tr1"))

## End(Not run)
```

theme_default	<i>Default bayesplot Theme for ggplot2 Graphics</i>
---------------	---

Description

This theme is imported from the **bayesplot** package. See [theme_default](#) for a complete documentation.

Arguments

base_size	base font size
base_family	base font family

Value

A theme object used in **ggplot2** graphics.

update.brmsfit	<i>Update brms models</i>
----------------	----------------------------------

Description

This method allows to update an existing brmsfit object

Usage

```
## S3 method for class 'brmsfit'
update(object, formula., newdata = NULL, recompile = NULL,
  ...)
```

Arguments

object	An object of class brmsfit.
formula.	Changes to the formula; for details see update.formula and brmsformula .
newdata	Optional data.frame to update the model with new data.
recompile	Logical, indicating whether the Stan model should be recompiled. If NULL (the default), update tries to figure out internally, if recompilation is necessary. Setting it to FALSE will cause all Stan code changing arguments to be ignored.
...	Other arguments passed to brm .

Details

Sometimes, when updating the model formula, it may happen that R complains about a mismatch between model frame and formula. This error can be avoided by supplying your original data again via argument newdata.

Examples

```
## Not run:
fit1 <- brm(time | cens(censored) ~ age * sex + disease + (1|patient),
            data = kidney, family = gaussian("log"))
summary(fit1)

## remove effects of 'disease'
fit2 <- update(fit1, formula. = ~ . - disease)
summary(fit2)

## remove the group specific term of 'patient' and
## change the data (just take a subset in this example)
fit3 <- update(fit1, formula. = ~ . - (1|patient),
              newdata = kidney[1:38, ])
summary(fit3)

## use another family and add population-level priors
fit4 <- update(fit1, family = weibull(), inits = "0",
              prior = set_prior("normal(0,5)"))
summary(fit4)

## End(Not run)
```

VarCorr.brmsfit

Extract Variance and Correlation Components

Description

This function calculates the estimated standard deviations, correlations and covariances of the group-level terms in a multilevel model of class `brmsfit`. For linear models, the residual standard deviations, correlations and covariances are also returned.

Usage

```
## S3 method for class 'brmsfit'
VarCorr(x, sigma = 1, summary = TRUE, robust = FALSE,
        probs = c(0.025, 0.975), ...)
```

Arguments

<code>x</code>	An object of class <code>brmsfit</code> .
<code>sigma</code>	Ignored (included for compatibility with <code>VarCorr</code>).
<code>summary</code>	Should summary statistics (i.e. means, sds, and 95% intervals) be returned instead of the raw values? Default is TRUE.
<code>robust</code>	If FALSE (the default) the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If TRUE, the median and the median absolute deviation (MAD) are applied instead. Only used if <code>summary</code> is TRUE.

probs	The percentiles to be computed by the quantile function. Only used if summary is TRUE.
...	Currently ignored.

Value

A list of lists (one per grouping factor), each with three elements: a matrix containing the standard deviations, an array containing the correlation matrix, and an array containing the covariance matrix with variances on the diagonal.

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

Examples

```
## Not run:
fit <- brm(count ~ log_Age_c + log_Base4_c * Trt_c + (1+Trt_c|visit),
           data = epilepsy, family = gaussian(), chains = 2)
VarCorr(fit)

## End(Not run)
```

vcov.brmsfit

*Covariance and Correlation Matrix of Population-Level Effects***Description**

Get a point estimate of the covariance or correlation matrix of population-level parameters

Usage

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

Arguments

object	An object of class brmsfit
correlation	logical; if FALSE (the default), compute the covariance matrix, if TRUE, compute the correlation matrix
...	Currently ignored

Details

Estimates are obtained by calculating the maximum likelihood covariances (correlations) of the posterior samples.

Value

covariance or correlation matrix of population-level parameters

Examples

```
## Not run:
fit <- brm(count ~ log_Age_c + log_Base4_c * Trt_c + (1+Trt_c|visit),
           data = epilepsy, family = gaussian(), chains = 2)
vcov(fit)

## End(Not run)
```

 VonMises

The von Mises Distribution

Description

Density, distribution function, and random generation for the von Mises distribution with location μ , and precision κ .

Usage

```
dvon_mises(x, mu, kappa, log = FALSE)

pvon_mises(q, mu, kappa, lower.tail = TRUE, log.p = FALSE, acc = 1e-20)

rvon_mises(n, mu, kappa)
```

Arguments

<code>x, q</code>	Vector of quantiles.
<code>mu</code>	Vector of location values.
<code>kappa</code>	Vector of precision values.
<code>log</code>	Logical; If TRUE, values are returned on the log scale.
<code>lower.tail</code>	Logical; If TRUE (default), return $P(X \leq x)$. Else, return $P(X > x)$.
<code>log.p</code>	Logical; If TRUE, values are returned on the log scale.
<code>acc</code>	Accuracy of numerical approximations.
<code>n</code>	Number of samples to draw from the distribution.

Details

See `vignette("brms_families")` for details on the parameterization.

waic.brmsfit	<i>Compute the WAIC</i>
--------------	-------------------------

Description

Compute the widely applicable information criterion (WAIC) based on the posterior likelihood using the **loo** package. For more details see [waic](#)

Usage

```
## S3 method for class 'brmsfit'
waic(x, ..., compare = TRUE, resp = NULL,
      pointwise = FALSE, model_names = NULL)
```

Arguments

<code>x</code>	A fitted model object.
<code>...</code>	More fitted model objects or further arguments passed to the underlying post-processing functions.
<code>compare</code>	A flag indicating if the information criteria of the models should be compared to each other via compare_ic .
<code>resp</code>	Optional names of response variables. If specified, fitted values of these response variables are returned.
<code>pointwise</code>	A flag indicating whether to compute the full log-likelihood matrix at once or separately for each observation. The latter approach is usually considerably slower but requires much less working memory. Accordingly, if one runs into memory issues, <code>pointwise = TRUE</code> is the way to go.
<code>model_names</code>	If NULL (the default) will use model names derived from parsing the call. Otherwise will use the passed values as model names.

Details

When comparing models fitted to the same data, the smaller the WAIC, the better the fit. For `brmsfit` objects, WAIC is an alias of `waic`. Use method [add_ic](#) to store information criteria in the fitted model object for later usage.

Value

If just one object is provided, an object of class `ic`. If multiple objects are provided, an object of class `iclist`.

Author(s)

Paul-Christian Buerkner <paul.buerkner@gmail.com>

References

- Vehtari, A., Gelman, A., & Gabry J. (2016). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. In *Statistics and Computing*, doi:10.1007/s11222-016-9696-4. arXiv preprint arXiv:1507.04544.
- Gelman, A., Hwang, J., & Vehtari, A. (2014). Understanding predictive information criteria for Bayesian models. *Statistics and Computing*, 24, 997-1016.
- Watanabe, S. (2010). Asymptotic equivalence of Bayes cross validation and widely applicable information criterion in singular learning theory. *The Journal of Machine Learning Research*, 11, 3571-3594.

Examples

```
## Not run:
# model with population-level effects only
fit1 <- brm(rating ~ treat + period + carry,
           data = inhaler, family = "gaussian")
waic(fit1)

# model with an additional varying intercept for subjects
fit2 <- brm(rating ~ treat + period + carry + (1|subject),
           data = inhaler, family = "gaussian")
# compare both models
waic(fit1, fit2)

## End(Not run)
```

Wiener

The Wiener Diffusion Model Distribution

Description

Density function and random generation for the Wiener diffusion model distribution with boundary separation α , non-decision time τ , bias β and drift rate δ .

Usage

```
dwiener(x, alpha, tau, beta, delta, resp = 1, log = FALSE)

rwiener(n, alpha, tau, beta, delta, types = c("q", "resp"))
```

Arguments

x	Vector of quantiles.
alpha	Boundary separation parameter.
tau	Non-decision time parameter.

beta	Bias parameter.
delta	Drift rate parameter.
resp	Response: "upper" or "lower". If no character vector, it is coerced to logical where TRUE indicates "upper" and FALSE indicates "lower".
log	Logical; If TRUE, values are returned on the log scale.
n	Number of samples to draw from the distribution.
types	Which types of responses to return? By default, return both the response times "q" and the dichotomous responses "resp". If either "q" or "resp", return only one of the two types.

Details

These are wrappers around functions of the **RWiener** package. See `vignette("brms_families")` for details on the parameterization.

See Also

[wienerdist](#)

Index

- *Topic **datasets**
 - epilepsy, [53](#)
 - inhaler, [70](#)
 - kidney, [78](#)
- acat (brmsfamily), [21](#)
- add_ic, [8](#), [40](#), [85](#), [157](#)
- add_ic<- (add_ic), [8](#)
- add_loo (add_ic), [8](#)
- add_waic (add_ic), [8](#)
- addition-terms, [6](#)
- as.array.brmsfit
 - (posterior_samples.brmsfit), [117](#)
- as.data.frame, [117](#)
- as.data.frame.brmsfit
 - (posterior_samples.brmsfit), [117](#)
- as.matrix.brmsfit, [116](#)
- as.matrix.brmsfit
 - (posterior_samples.brmsfit), [117](#)
- as.mcmc (as.mcmc.brmsfit), [9](#)
- as.mcmc.brmsfit, [9](#)
- asym_laplace (brmsfamily), [21](#)
- AsymLaplace, [10](#)
- autocor, [11](#)

- bayes_factor, [5](#), [11](#), [15](#), [121](#)
- bayes_factor (bayes_factor.brmsfit), [11](#)
- bayes_factor.brmsfit, [11](#)
- bayes_R2, [26](#)
- bayes_R2 (bayes_R2.brmsfit), [12](#)
- bayes_R2.brmsfit, [12](#)
- bayesplot, [5](#), [124](#)
- bernoulli (brmsfamily), [21](#)
- Beta (brmsfamily), [21](#)
- bf (brmsformula), [26](#)
- bf-helpers (brmsformula-helpers), [34](#)
- binomial, [25](#)

- bridge_sampler, [11](#), [12](#), [15](#), [26](#), [121](#)
- bridge_sampler
 - (bridge_sampler.brmsfit), [14](#)
- bridge_sampler.brmsfit, [14](#)
- bridge_sampler.stanfit, [14](#)
- brm, [5–7](#), [15](#), [25–27](#), [37](#), [50](#), [67](#), [101](#), [113](#), [142](#), [143](#), [153](#)
- brm_multiple, [29](#), [37](#)
- brms, [19](#), [25](#), [26](#)
- brms (brms-package), [5](#)
- brms-package, [5](#)
- brmsfamily, [6](#), [16](#), [18](#), [19](#), [21](#), [26](#), [32](#), [52](#), [53](#), [62](#), [90](#), [91](#), [101](#), [112](#)
- brmsfit, [6](#), [19](#)
- brmsfit (brmsfit-class), [25](#)
- brmsfit-class, [25](#)
- brmsformula, [5–7](#), [16](#), [18](#), [19](#), [24](#), [26](#), [26](#), [34](#), [35](#), [37](#), [51](#), [62](#), [64](#), [65](#), [89](#), [91](#), [99–101](#), [103](#), [105](#), [109](#), [112](#), [113](#), [139](#), [141](#), [153](#)
- brmsformula-helpers, [34](#)
- brmshypothesis, [36](#), [69](#)
- brmsprior, [26](#)
- brmsprior (set_prior), [139](#)
- brmsprior-class (set_prior), [139](#)

- categorical (brmsfamily), [21](#)
- cbind, [104](#)
- coef.brmsfit, [38](#), [68](#)
- combine_models, [37](#), [39](#)
- compare, [40](#)
- compare_ic, [40](#), [76](#), [85](#), [157](#)
- control_params
 - (control_params.brmsfit), [41](#)
- control_params.brmsfit, [41](#)
- cor_ar, [42](#), [44](#), [45](#)
- cor_arma, [42](#), [43](#), [45](#), [49](#)
- cor_arma-class (cor_arma), [43](#)
- cor_arr, [44](#), [44](#), [45](#)
- cor_brms, [16](#), [26](#), [45](#), [62](#), [90](#), [91](#), [112](#)

- cor_brms-class (cor_brms), 45
- cor_bsts, 13, 45, 46, 57, 82, 125, 128, 136, 147
- cor_bsts-class (cor_bsts), 46
- cor_car, 45, 47
- cor_errorsar (cor_sar), 50
- cor_fixed, 13, 45, 48, 57, 82, 125, 128, 136, 147
- cor_icar (cor_car), 47
- cor_lagsar (cor_sar), 50
- cor_ma, 44, 45, 49
- cor_sar, 13, 45, 50, 57, 82, 125, 128, 136, 147
- cov_fixed (cor_fixed), 48
- cratio (brmsfamily), 21
- cs, 51
- cse (cs), 51
- cumulative (brmsfamily), 21
- custom_family, 21, 52
- customfamily, 25
- customfamily (custom_family), 52

- dasym_laplace (AsymLaplace), 10
- dexgaussian (ExGaussian), 55
- dfrechet (Frechet), 60
- dgen_extreme_value (GenExtremeValue), 61
- diagnostic-quantities (log_posterior.brmsfit), 83
- dinv_gaussian (InvGaussian), 71
- dmulti_normal (MultiNormal), 107
- dmulti_student_t (MultiStudentT), 108
- dshifted_lnorm (Shifted_Lognormal), 144
- dskew_normal (SkewNormal), 145
- dstudent_t (StudentT), 150
- dvon_mises (VonMises), 156
- dwiener (Wiener), 158

- E_loo, 87
- environment, 52
- epilepsy, 53
- ExGaussian, 55
- exgaussian (brmsfamily), 21
- exponential (brmsfamily), 21
- expose_functions (expose_functions.brmsfit), 55
- expose_functions.brmsfit, 55
- expose_stan_functions, 55, 56
- expp1, 56

- family, 21, 25

- fitted, 52, 88, 94, 127
- fitted.brmsfit, 56
- fixef (fixef.brmsfit), 59
- fixef.brmsfit, 59
- formula, 16, 37, 62, 89, 91, 112
- Frechet, 60
- frechet (brmsfamily), 21
- future, 18

- gam, 28
- gamm, 17, 28, 92, 142
- Gamma, 25
- gen_extreme_value (brmsfamily), 21
- GenExtremeValue, 61
- geom_contour, 95
- geom_errorbar, 95
- geom_jitter, 95
- geom_point, 95
- geom_raster, 95
- geom_rug, 95
- geom_smooth, 95
- geometric (brmsfamily), 21
- get_prior, 16, 18, 62, 90, 91, 143
- ggplot, 96, 148
- ggtheme, 37, 95, 114
- gp, 28, 63, 142
- gr, 65
- gtable, 114

- horseshoe, 66, 141
- hurdle_gamma (brmsfamily), 21
- hurdle_lognormal (brmsfamily), 21
- hurdle_negbinomial (brmsfamily), 21
- hurdle_poisson (brmsfamily), 21
- hypothesis, 36, 37
- hypothesis (hypothesis.brmsfit), 68
- hypothesis.brmsfit, 68

- inhaler, 70
- inv_logit_scaled, 72
- InvGaussian, 71
- is.brmsfit, 73
- is.brmsformula, 73
- is.brmsprior, 73
- is.brmsterms, 74
- is.cor_arma (is.cor_brms), 74
- is.cor_brms, 74
- is.cor_bsts (is.cor_brms), 74
- is.cor_car (is.cor_brms), 74

- is.cor_fixed(is.cor_brms), 74
- is.cor_sar(is.cor_brms), 74
- is.mvbrmsformula, 75
- is.mvbrmsterms, 75
- kfold, 134
- kfold(kfold.brmsfit), 76
- kfold.brmsfit, 76
- kidney, 78
- lasso, 79, 141
- launch_shinystan, 80, 148
- launch_shinystan
 - (launch_shinystan.brmsfit), 80
- launch_shinystan.brmsfit, 80
- lf(brmsformula-helpers), 34
- log_lik, 52, 88
- log_lik(log_lik.brmsfit), 82
- log_lik.brmsfit, 82
- log_posterior(log_posterior.brmsfit),
 - 83
- log_posterior.brmsfit, 83
- logit_scaled, 81
- logLik.brmsfit(log_lik.brmsfit), 82
- logm1, 81
- lognormal(brmsfamily), 21
- LOO(loo.brmsfit), 84
- loo, 5, 26, 40, 77, 83, 84, 134
- loo(loo.brmsfit), 84
- LOO.brmsfit(loo.brmsfit), 84
- loo.brmsfit, 84
- loo::loo_model_weights, 86
- loo_linpred(loo_predict.brmsfit), 87
- loo_model_weights, 106, 115, 122
- loo_model_weights
 - (loo_model_weights.brmsfit), 86
- loo_model_weights.brmsfit, 86
- loo_predict(loo_predict.brmsfit), 87
- loo_predict.brmsfit, 87
- loo_predictive_interval
 - (loo_predict.brmsfit), 87
- make_conditions, 88, 93, 138
- make_stancode, 5, 89
- make_standata, 5, 91, 147
- marginal_effects, 5, 88, 89, 98, 138
- marginal_effects
 - (marginal_effects.brmsfit), 93
- marginal_effects.brmsfit, 93
- marginal_smooths
 - (marginal_smooths.brmsfit), 97
- marginal_smooths.brmsfit, 97
- MCMC, 113
- mcmc_combo, 114
- mcmc_pairs, 111
- me, 35, 99
- mgcv::s, 139
- mgcv::t2, 139
- mi, 29, 100
- mixture, 32, 101
- mm, 28, 102, 104
- mmc, 103, 104
- mo, 105
- model_weights, 115, 122
- model_weights(model_weights.brmsfit),
 - 106
- model_weights.brmsfit, 106
- mono(mo), 105
- monotonic(mo), 105
- MultiNormal, 107
- MultiStudentT, 108
- mvbf, 32
- mvbf(mvbrmsformula), 108
- mvbrmsformula, 16, 33, 35, 37, 62, 89, 91,
 - 108, 112, 113
- neff_ratio(log_posterior.brmsfit), 83
- negbinomial(brmsfamily), 21
- ngrps(ngrps.brmsfit), 109
- ngrps.brmsfit, 109
- nlf(brmsformula-helpers), 34
- nsamples(nsamples.brmsfit), 110
- nsamples.brmsfit, 110
- nuts_params(log_posterior.brmsfit), 83
- pairs, 110
- pairs.brmsfit, 110
- par.names(parnames), 111
- pareto_k_ids, 85, 134
- parnames, 68, 111
- parse_bf, 74, 75, 112
- pasym_laplace(AsymLaplace), 10
- pexgaussian(ExGaussian), 55
- pfrechet(Frechet), 60
- pgen_extreme_value(GenExtremeValue), 61
- pinv_gaussian(InvGaussian), 71
- plan, 18
- plot.brmsfit, 113

- plot.brmsHypothesis (brmsHypothesis), 36
- plot.brmsMarginalEffects
 - (marginal_effects.brmsfit), 93
- post_prob, 12, 15, 106, 115, 120, 122
- post_prob (post_prob.brmsfit), 120
- post_prob.brmsfit, 120
- posterior.samples
 - (posterior_samples.brmsfit), 117
- posterior_average, 122
- posterior_average
 - (posterior_average.brmsfit), 114
- posterior_average.brmsfit, 114
- posterior_interval
 - (posterior_interval.brmsfit), 116
- posterior_interval.brmsfit, 116
- posterior_linpred, 58
- posterior_linpred (fitted.brmsfit), 56
- posterior_predict (predict.brmsfit), 127
- posterior_samples
 - (posterior_samples.brmsfit), 117
- posterior_samples.brmsfit, 117
- posterior_summary
 - (posterior_summary.brmsfit), 118
- posterior_summary.brmsfit, 118
- posterior_table, 119
- pp_average, 115
- pp_average (pp_average.brmsfit), 121
- pp_average.brmsfit, 121
- pp_check, 5
- pp_check (pp_check.brmsfit), 123
- pp_check.brmsfit, 123
- pp_mixture (pp_mixture.brmsfit), 124
- pp_mixture.brmsfit, 124
- PPC, 123, 124
- predict, 52, 56, 88, 94
- predict.brmsfit, 124, 127
- predictive_error (residuals.brmsfit), 135
- print.brmsfit, 130
- print.brmsHypothesis (brmsHypothesis), 36
- print.brmsprior, 130
- print.brmssummary (print.brmsfit), 130
- print.default, 36
- prior (set_prior), 139
- prior_ (set_prior), 139
- prior_samples (prior_samples.brmsfit), 131
- prior_samples.brmsfit, 131
- prior_string (set_prior), 139
- prior_summary (prior_summary.brmsfit), 132
- prior_summary.brmsfit, 132
- pshifted_lnorm (Shifted_Lognormal), 144
- psis, 87
- pskew_normal (SkewNormal), 145
- pstudent_t (StudentT), 150
- pvon_mises (VonMises), 156
- qasym_laplace (AsymLaplace), 10
- qfrechet (Frechet), 60
- qshifted_lnorm (Shifted_Lognormal), 144
- qskew_normal (SkewNormal), 145
- qstudent_t (StudentT), 150
- ranef (ranef.brmsfit), 133
- ranef.brmsfit, 68, 133
- rasym_laplace (AsymLaplace), 10
- reloo, 77, 85, 134
- residuals.brmsfit, 135
- resp_cat (addition-terms), 6
- resp_cens (addition-terms), 6
- resp_dec (addition-terms), 6
- resp_mi (addition-terms), 6
- resp_se (addition-terms), 6
- resp_trials (addition-terms), 6
- resp_trunc (addition-terms), 6
- resp_weights (addition-terms), 6
- restructure, 137
- rexgaussian (ExGaussian), 55
- rfrechet (Frechet), 60
- rgen_extreme_value (GenExtremeValue), 61
- rhat (log_posterior.brmsfit), 83
- rinv_gaussian (InvGaussian), 71
- rmulti_normal (MultiNormal), 107
- rmulti_student_t (MultiStudentT), 108
- rows2labels, 89, 138
- rshifted_lnorm (Shifted_Lognormal), 144
- rskew_normal (SkewNormal), 145
- rstudent_t (StudentT), 150
- runApp, 80
- rvon_mises (VonMises), 156

- rwienner (Wiener), 158
- s, 28, 138
- scale_colour_gradient, 96
- scale_colour_grey, 96
- set.seed, 68, 115, 122
- set_mecor (brmsformula-helpers), 34
- set_nl (brmsformula-helpers), 34
- set_prior, 16, 18, 30, 31, 62, 67, 79, 90, 91, 101, 139
- set_rescor (brmsformula-helpers), 34
- Shifted_Lognormal, 144
- shifted_lognormal (brmsfamily), 21
- skew_normal (brmsfamily), 21
- SkewNormal, 145
- sratio (brmsfamily), 21
- Stan, 5
- stan, 18, 19, 41
- stancode (stancode.brmsfit), 146
- stancode.brmsfit, 146
- standata (standata.brmsfit), 147
- standata.brmsfit, 147
- stanfit, 26
- stanplot, 5
- stanplot (stanplot.brmsfit), 148
- stanplot.brmsfit, 148
- stanvar, 17, 52, 53, 90, 92, 149
- stanvars, 26
- stanvars (stanvar), 149
- student (brmsfamily), 21
- StudentT, 150
- summary, 5
- summary.brmsfit, 151
- t2, 28
- t2 (s), 138
- TDist, 151
- theme, 37, 95, 114
- theme_black, 152
- theme_default, 37, 95, 114, 153, 153
- update, 17
- update.brmsfit, 134, 153
- update.formula, 153
- VarCorr, 154
- VarCorr (VarCorr.brmsfit), 154
- VarCorr.brmsfit, 154
- vcov.brmsfit, 155
- Vectorize, 56
- von_mises (brmsfamily), 21
- VonMises, 156
- WAIC, 40
- WAIC (waic.brmsfit), 157
- waic, 5, 26, 83, 157
- waic (waic.brmsfit), 157
- WAIC.brmsfit (waic.brmsfit), 157
- waic.brmsfit, 157
- weibull (brmsfamily), 21
- Wiener, 158
- wiener (brmsfamily), 21
- wienerdist, 159
- zero_inflated_beta (brmsfamily), 21
- zero_inflated_binomial (brmsfamily), 21
- zero_inflated_negbinomial (brmsfamily), 21
- zero_inflated_poisson (brmsfamily), 21
- zero_one_inflated_beta (brmsfamily), 21