Package 'brms'

September 9, 2025

Encoding UTF-8

Type Package

Title Bayesian Regression Models using 'Stan'

Version 2.23.0

Date 2025-09-08

Depends R (>= 3.6.0), Rcpp (>= 0.12.0), methods

Imports rstan (>= 2.29.0), ggplot2 (>= 2.0.0), loo (>= 2.8.0), posterior (>= 1.6.0), Matrix (>= 1.1.1), mgcv (>= 1.8-13), rstantools (>= 2.1.1), bayesplot (>= 1.5.0), bridgesampling (>= 0.3-0), glue (>= 1.3.0), rlang (>= 1.0.0), future (>= 1.19.0), future.apply (>= 1.0.0), matrixStats, nleqslv, nlme, coda, abind, stats, utils, parallel, grDevices, backports

Suggests testthat (>= 3.1.9), emmeans (>= 1.4.2), cmdstanr (>= 0.5.0), projpred (>= 2.0.0), priorsense (>= 1.0.0), shinystan (>= 2.4.0), splines2 (>= 0.5.0), RWiener, rtdists, extraDistr, processx, mice, spdep, mnormt, lme4, MCMCglmm, ape, arm, statmod, digest, diffobj, betareg, R.rsp, gtable, shiny, knitr, rmarkdown, ragg, colorspace, mirai, future.mirai

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 both theory-driven and data-driven non-linear terms, auto-correlation structures, censoring and truncation, 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 prior knowledge. Models can easily be evaluated and compared using several methods assessing posterior or prior predictions. References: Bürkner (2017) <doi:10.18637/jss.v080.i01>; Bürkner (2018) <doi:10.32614/RJ-2018-017>; Bürkner (2021) <doi:10.18637/jss.v100.i05>; Carpenter et al. (2017)

<doi:10.18637/jss.v076.i01>.

LazyData true

NeedsCompilation no

License GPL-2

URL https://github.com/paul-buerkner/brms, https://discourse.mc-stan.org/, https://paulbuerkner.com/brms/

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

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

SystemRequirements pngquant

VignetteBuilder knitr, R.rsp

RoxygenNote 7.3.2

Author Paul-Christian Bürkner [aut, cre],
Jonah Gabry [ctb],
Sebastian Weber [ctb],
Andrew Johnson [ctb],
Martin Modrak [ctb],
Hamada S. Badr [ctb],
Frank Weber [ctb],
Aki Vehtari [ctb],
Mattan S. Ben-Shachar [ctb],
Hayden Rabel [ctb],
Simon C. Mills [ctb],
Stephen Wild [ctb],
Ven Popov [ctb],
Ioannis Kosmidis [ctb],
Ben Schneider [ctb],
Noa Kallioinen [ctb],
Wellington J. Silva [ctb],
Luiz Carvalho [ctb],
Sermet Pekin [ctb]

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

Repository CRAN

Date/Publication 2025-09-09 10:50:02 UTC

Contents

brms-package 6
addition-terms 8
add_criterion 11
add_loo 12
add_rstan_model 13
ar 13

arma . . . 14
as.brmsprior . . . 15
as.data.frame.brmsfit . . . 16
as.mcmc.brmsfit . . . 17
AsymLaplace . . . 18
autocor-terms . . . 19
autocor.brmsfit . . . 20
bayes_factor.brmsfit . . . 20
bayes_R2.brmsfit . . . 21
BetaBinomial . . . 23
bridge_sampler.brmsfit . . . 24
brm . . . 25
brmsfamily . . . 34
brmsfit-class . . . 40
brmsformula . . . 41
brmsformula-helpers . . . 51
brmshypothesis . . . 53
brmsterms . . . 54
brm_multiple . . . 56
car . . . 60
coef.brmsfit . . . 61
combine_models . . . 63
compare_ic . . . 63
conditional_effects.brmsfit . . . 64
conditional_smooths.brmsfit . . . 69
constant . . . 72
control_params . . . 73
cor_ar . . . 73
cor_arma . . . 74
cor_brms . . . 75
cor_car . . . 76
cor_cosy . . . 77
cor_fixed . . . 78
cor_ma . . . 79
cor_sar . . . 80
cosy . . . 81
create_priorsense_data.brmsfit . . . 82
cs . . . 83
custom_family . . . 83
default_prior . . . 86
default_prior.default . . . 87
density_ratio . . . 89
diagnostic-quantities . . . 90
Dirichlet . . . 91
draws-brms . . . 92
draws-index-brms . . . 93
emmeans-brms-helpers . . . 94
epilepsy . . . 96

ExGaussian . . . 97
expose_functions.brmsfit . . . 98
expp1 . . . 98
family.brmsfit . . . 99
fcor . . . 99
fitted.brmsfit . . . 100
fixef.brmsfit . . . 102
Frechet . . . 103
GenExtremeValue . . . 104
get_dpar . . . 105
get_refmodel.brmsfit . . . 106
gp . . . 108
gr . . . 110
horseshoe . . . 112
Hurdle . . . 114
hypothesis.brmsfit . . . 115
inhaler . . . 118
inits.brmsfit . . . 119
InvGaussian . . . 120
inv_logit_scaled . . . 121
is.brmsfit . . . 121
is.brmsfit_multiple . . . 122
is.brmsformula . . . 122
is.brmsprior . . . 122
is.brmsterms . . . 123
is.cor_brms . . . 123
is.mvbrmsformula . . . 124
is.mvbrmsterms . . . 124
kfold.brmsfit . . . 125
kfold_predict . . . 128
kidney . . . 130
lasso . . . 131
launch_shinystan.brmsfit . . . 132
LogisticNormal . . . 133
logit_scaled . . . 133
logm1 . . . 134
log_lik.brmsfit . . . 134
loo.brmsfit . . . 136
loo_compare.brmsfit . . . 138
loo_model_weights.brmsfit . . . 139
loo_moment_match.brmsfit . . . 140
loo_predict.brmsfit . . . 142
loo_R2.brmsfit . . . 144
loo_subsample.brmsfit . . . 145
loss . . . 146
ma . . . 147
make_conditions . . . 148
mcmc_plot.brmsfit . . . 149

me . . . 151
mi . . . 152
mixture . . . 153
mm . . . 155
mmc . . . 157
mo . . . 158
model_weights.brmsfit . . . 159
MultiNormal . . . 160
MultiStudentT . . . 161
mvbind . . . 162
mvbrmsformula . . . 162
ngrps.brmsfit . . . 163
nsamples.brmsfit . . . 164
opencl . . . 164
pairs.brmsfit . . . 165
parnames . . . 166
plot.brmsfit . . . 166
posterior_average.brmsfit . . . 168
posterior_epred.brmsfit . . . 170
posterior_interval.brmsfit . . . 172
posterior_linpred.brmsfit . . . 173
posterior_predict.brmsfit . . . 174
posterior_samples.brmsfit . . . 177
posterior_smooths.brmsfit . . . 178
posterior_summary . . . 180
posterior_table . . . 181
post_prob.brmsfit . . . 182
pp_average.brmsfit . . . 183
pp_check.brmsfit . . . 185
pp_mixture.brmsfit . . . 187
predict.brmsfit . . . 189
predictive_error.brmsfit . . . 192
predictive_interval.brmsfit . . . 193
prepare_predictions.brmsfit . . . 194
print.brmsfit . . . 196
print.brmsprior . . . 197
prior_draws.brmsfit . . . 197
prior_summary.brmsfit . . . 198
psis.brmsfit . . . 199
R2D2 . . . 201
ranef.brmsfit . . . 202
read_csv_as_stanfit . . . 203
recompile_model . . . 205
reloo.brmsfit . . . 205
rename_pars . . . 207
residuals.brmsfit . . . 208
restructure . . . 210
restructure.brmsfit . . . 211

rows2labels . . . 212
s . . . 212
sar . . . 213
save_pars . . . 215
set_prior . . . 216
Shifted_Lognormal . . . 222
SkewNormal . . . 223
stancode . . . 224
stancode.brmsfit . . . 225
stancode.default . . . 226
standata . . . 228
standata.brmsfit . . . 229
standata.default . . . 230
stanvar . . . 232
StudentT . . . 234
summary.brmsfit . . . 235
theme_black . . . 236
theme_default . . . 237
threading . . . 237
unstr . . . 238
update.brmsfit . . . 239
update.brmsfit_multiple . . . 240
update_adterms . . . 241
validate_newdata . . . 242
validate_prior . . . 243
VarCorr.brmsfit . . . 245
vcov.brmsfit . . . 246
VonMises . . . 247
waic.brmsfit . . . 248
Wiener . . . 249
ZeroInflated . . . 251

Index **253**

`brms-package` *Bayesian Regression Models using 'Stan'*

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 https://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 `stancode`, prepares the data via `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 `conditional_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 my 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")`.

Author(s)

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

Other contributors:

- Jonah Gabry [contributor]
- Sebastian Weber [contributor]
- Andrew Johnson [contributor]
- Martin Modrak [contributor]
- Hamada S. Badr [contributor]
- Frank Weber [contributor]
- Aki Vehtari [contributor]
- Mattan S. Ben-Shachar [contributor]
- Hayden Rabel [contributor]
- Simon C. Mills [contributor]
- Stephen Wild [contributor]
- Ven Popov [contributor]
- Ioannis Kosmidis [contributor]
- Ben Schneider [contributor]

- Noa Kallioinen [contributor]
- Wellington J. Silva [contributor]
- Luiz Carvalho [contributor]
- Sermet Pekin [contributor]

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 (2018). Advanced Bayesian Multilevel Modeling with the R Package brms. *The R Journal*. 10(1), 395–411. doi:10.32614/RJ-2018-017

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

Stan Development Team (2020). RStan: the R interface to Stan. R package version 2.21.2. https://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. Detailed documentation on the use of each of these functions can be found in the Details section of brmsformula (under "Additional response information").

Usage

```
resp_se(x, sigma = FALSE)

resp_weights(x, scale = FALSE)

resp_trials(x)

resp_thres(x, gr = NA)

resp_cat(x)

resp_dec(x)

resp_bhaz(gr = NA, df = 5, ...)
```

```
resp_cens(x, y2 = NA)

resp_trunc(lb = -Inf, ub = Inf)

resp_mi(sdy = NA)

resp_index(x)

resp_rate(denom)

resp_subset(x)

resp_vreal(...)

resp_vint(...)
```

Arguments

x	A vector; Ideally a single variable defined in the data (see Details). Allowed values depend on the function: resp_se and resp_weights require positive numeric values. resp_trials, resp_thres, and resp_cat require positive integers. resp_dec requires 0 and 1, or alternatively 'lower' and 'upper'. resp_subset requires 0 and 1, or alternatively FALSE and TRUE. resp_cens requires 'left', 'none', 'right', and 'interval' (or equivalently -1, 0, 1, and 2) to indicate left, no, right, or interval censoring. resp_index does not make any requirements other than the value being unique for each observation.
sigma	Logical; Indicates whether the residual standard deviation parameter sigma should be included in addition to the known measurement error. Defaults to FALSE for backwards compatibility, but setting it to TRUE is usually the better choice.
scale	Logical; Indicates whether weights should be scaled so that the average weight equals one. Defaults to FALSE.
gr	A vector of grouping indicators.
df	Degrees of freedom of baseline hazard splines for Cox models.
...	For resp_vreal, vectors of real values. For resp_vint, vectors of integer values. In Stan, these variables will be named vreal1, vreal2, ..., and vint1, vint2, ..., respectively.
y2	A vector specifying the upper bounds in interval censoring. Will be ignored for non-interval censored observations. However, it should NOT be NA even for non-interval censored observations to avoid accidental exclusion of these observations.
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.
sdy	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.
denom	A vector of positive numeric values specifying the denominator values from which the response rates are computed.

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` (under "Additional response information").

It is highly recommended to use a single data variable as input for `x` (instead of a more complicated expression) to make sure all post-processing functions work as expected.

Value

A list of additional response information to be processed further by **brms**.

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) ~ zBase * Trt,
            data = epilepsy, family = poisson())
summary(fit4)

## End(Not run)
```

add_criterion *Add model fit criteria to model objects*

Description

Add model fit criteria to model objects

Usage

```
add_criterion(x, ...)

## S3 method for class 'brmsfit'
add_criterion(
  x,
  criterion,
  model_name = NULL,
  overwrite = FALSE,
  file = NULL,
  force_save = FALSE,
  ...
)
```

Arguments

x	An R object typically of class brmsfit.
...	Further arguments passed to the underlying functions computing the model fit criteria. If you are recomputing an already stored criterion with other ... arguments, make sure to set overwrite = TRUE.
criterion	Names of model fit criteria to compute. Currently supported are "loo", "waic", "kfold", "loo_subsample", "bayes_R2" (Bayesian R-squared), "loo_R2" (LOO-adjusted 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.
overwrite	Logical; Indicates if already stored fit indices should be overwritten. Defaults to FALSE. Setting it to TRUE is useful for example when changing additional arguments of an already stored criterion.
file	Either NULL or a character string. In the latter case, the fitted model object including the newly added criterion values is saved via saveRDS in a file named after the string supplied in file. The .rds extension is added automatically. If x was already stored in a file before, the file name will be reused automatically (with a message) unless overwritten by file. In any case, file only applies if new criteria were actually added via add_criterion or if force_save was set to TRUE.
force_save	Logical; only relevant if file is specified and ignored otherwise. If TRUE, the fitted model object will be saved regardless of whether new criteria were added via add_criterion.

Details

Functions add_loo and add_waic are aliases of add_criterion with fixed values for the criterion argument.

Value

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

Examples

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

## End(Not run)
```

---

add_loo *Add model fit criteria to model objects*

Description

Deprecated aliases of add_criterion.

Usage

```
add_loo(x, model_name = NULL, ...)

add_waic(x, model_name = NULL, ...)

add_ic(x, ...)

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

add_ic(x, ...) <- value
```

Arguments

x	An R object typically of class brmsfit.
model_name	Optional name of the model. If NULL (the default) the name is taken from the call to x.
...	Further arguments passed to the underlying functions computing the model fit criteria. If you are recomputing an already stored criterion with other ... arguments, make sure to set overwrite = TRUE.

ic, value — Names of model fit criteria to compute. Currently supported are "loo", "waic", "kfold", "R2" (R-squared), and "marglik" (log marginal likelihood).

Value

An object of the same class as x, but with model fit criteria added for later usage. Previously computed criterion objects will be overwritten.

---

add_rstan_model — *Add compiled* **rstan** *models to* brmsfit *objects*

Description

Compile a stanmodel and add it to a brmsfit object. This enables some advanced functionality of **rstan**, most notably log_prob and friends, to be used with brms models fitted with other Stan backends.

Usage

```
add_rstan_model(x, overwrite = FALSE)
```

Arguments

x	A brmsfit object to be updated.
overwrite	Logical. If TRUE, overwrite any existing stanmodel. Defaults to FALSE.

Value

A (possibly updated) brmsfit object.

---

ar — *Set up AR(p) correlation structures*

Description

Set up an autoregressive (AR) term of order p in **brms**. The function does not evaluate its arguments – it exists purely to help set up a model with AR terms.

Usage

```
ar(time = NA, gr = NA, p = 1, cov = FALSE)
```

Arguments

time	An optional time variable specifying the time ordering of the observations. By default, the existing order of the observations in the data is used.
gr	An optional grouping variable. If specified, the correlation structure is assumed to apply only to observations within the same grouping level.
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. This is currently only possible for stationary ARMA effects of order 1. If the model family does not have natural residuals, latent residuals are added automatically. If FALSE (the default), a regression formulation is used that is considerably faster and allows for ARMA effects of order higher than 1 but is only available for gaussian models and some of its generalizations.

Value

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

See Also

autocor-terms, arma, ma

Examples

```
## Not run:
data("LakeHuron")
LakeHuron <- as.data.frame(LakeHuron)
fit <- brm(x ~ ar(p = 2), data = LakeHuron)
summary(fit)

## End(Not run)
```

---

arma *Set up ARMA(p,q) correlation structures*

Description

Set up an autoregressive moving average (ARMA) term of order (p, q) in **brms**. The function does not evaluate its arguments – it exists purely to help set up a model with ARMA terms.

Usage

```
arma(time = NA, gr = NA, p = 1, q = 1, cov = FALSE)
```

Arguments

time	An optional time variable specifying the time ordering of the observations. By default, the existing order of the observations in the data is used.
gr	An optional grouping variable. If specified, the correlation structure is assumed to apply only to observations within the same grouping level.
p	A non-negative integer specifying the autoregressive (AR) order of the ARMA structure. Default is 1.
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. This is currently only possible for stationary ARMA effects of order 1. If the model family does not have natural residuals, latent residuals are added automatically. If FALSE (the default), a regression formulation is used that is considerably faster and allows for ARMA effects of order higher than 1 but is only available for gaussian models and some of its generalizations.

Value

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

See Also

autocor-terms, ar, ma,

Examples

```
## Not run:
data("LakeHuron")
LakeHuron <- as.data.frame(LakeHuron)
fit <- brm(x ~ arma(p = 2, q = 1), data = LakeHuron)
summary(fit)

## End(Not run)
```

---

as.brmsprior *Transform into a brmsprior object*

Description

Try to transform an object into a brmsprior object.

Usage

```
as.brmsprior(x)
```

Arguments

x	An object to be transformed.

Value

A brmsprior object if the transformation was possible.

---

as.data.frame.brmsfit *Extract Posterior Draws*

Description

Extract posterior draws in conventional formats as data.frames, matrices, or arrays.

Usage

```
## S3 method for class 'brmsfit'
as.data.frame(
  x,
  row.names = NULL,
  optional = TRUE,
  pars = NA,
  variable = NULL,
  draw = NULL,
  subset = NULL,
  ...
)

## S3 method for class 'brmsfit'
as.matrix(x, pars = NA, variable = NULL, draw = NULL, subset = NULL, ...)

## S3 method for class 'brmsfit'
as.array(x, pars = NA, variable = NULL, draw = NULL, subset = NULL, ...)
```

Arguments

x	A brmsfit object or another R object for which the methods are defined.
row.names, optional	Unused and only added for consistency with the as.data.frame generic.
pars	Deprecated alias of variable. For reasons of backwards compatibility, pars is interpreted as a vector of regular expressions by default unless fixed = TRUE is specified.
variable	A character vector providing the variables to extract. By default, all variables are extracted.
draw	The draw indices to be select. Subsetting draw indices will lead to an automatic merging of chains.

subset	Deprecated alias of draw.
...	Further arguments to be passed to the corresponding as_draws_* methods as well as to subset_draws.

Value

A data.frame, matrix, or array containing the posterior draws.

See Also

as_draws, subset_draws

---

as.mcmc.brmsfit *(Deprecated) Extract posterior samples for use with the* **coda** *package*

Description

The as.mcmc method is deprecated. We recommend using the more modern and consistent as_draws_* extractor functions of the **posterior** package instead.

Usage

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

Arguments

x	An R object typically 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.
fixed	Indicates whether parameter names should be matched exactly (TRUE) or treated as regular expressions (FALSE). Default is FALSE.
combine_chains	Indicates whether chains should be combined.
inc_warmup	Indicates if the warmup samples should be included. Default is FALSE. Warmup samples are used to tune the parameters of the sampling algorithm and should not be analyzed.
...	currently unused

Value

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

`AsymLaplace` *The Asymmetric Laplace Distribution*

Description

Density, distribution function, quantile function and random generation for the asymmetric Laplace distribution with location `mu`, scale `sigma` and asymmetry parameter `quantile`.

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

`x, q`	Vector of quantiles.
`mu`	Vector of locations.
`sigma`	Vector of scales.
`quantile`	Asymmetry parameter corresponding to quantiles in quantile regression (hence the name).
`log`	Logical; If `TRUE`, values are returned on the log scale.
`lower.tail`	Logical; If `TRUE` (default), return P(X <= x). Else, return P(X > x) .

log.p	Logical; If TRUE, values are returned on the log scale.
p	Vector of probabilities.
n	Number of draws to sample from the distribution.

Details

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

---

autocor-terms *Autocorrelation structures*

Description

Specify autocorrelation terms in **brms** models. Currently supported terms are arma, ar, ma, cosy, unstr, sar, car, and fcor. Terms can be directly specified within the formula, or passed to the autocor argument of brmsformula in the form of a one-sided formula. For deprecated ways of specifying autocorrelation terms, see cor_brms.

Details

The autocor term functions are almost solely useful when called in formulas passed to the **brms** package. They do not evaluate its arguments – but exist purely to help set up a model with autocorrelation terms.

See Also

brmsformula, acformula, arma, ar, ma, cosy, unstr, sar, car, fcor

Examples

```
# specify autocor terms within the formula
y ~ x + arma(p = 1, q = 1) + car(M)

# specify autocor terms in the 'autocor' argument
bf(y ~ x, autocor = ~ arma(p = 1, q = 1) + car(M))

# specify autocor terms via 'acformula'
bf(y ~ x) + acformula(~ arma(p = 1, q = 1) + car(M))
```

autocor.brmsfit *(Deprecated) Extract Autocorrelation Objects*

Description

(Deprecated) Extract Autocorrelation Objects

Usage

```
## S3 method for class 'brmsfit'
autocor(object, resp = NULL, ...)

autocor(object, ...)
```

Arguments

object	An object of class brmsfit.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
...	Currently unused.

Value

A cor_brms object or a list of such objects for multivariate models. Not supported for models fitted with brms 2.11.1 or higher.

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.

The computation of Bayes factors based on bridge sampling requires a lot more posterior samples than usual. A good conservative rule of thumb is perhaps 10-fold more samples (read: the default of 4000 samples may not be enough in many cases). If not enough posterior samples are provided, the bridge sampling algorithm tends to be unstable, leading to considerably different results each time it is run. We thus recommend running `bayes_factor` multiple times to check the stability of the results.

More details are provided under `bridgesampling::bayes_factor`.

See Also

`bridge_sampler`, `post_prob`

Examples

```
## Not run:
# model with the treatment effect
fit1 <- brm(
  count ~ zAge + zBase + Trt,
  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 ~ zAge + zBase,
  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,
  resp = NULL,
  summary = TRUE,
  robust = FALSE,
  probs = c(0.025, 0.975),
  ...
)
```

Arguments

object	An object of class brmsfit.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
summary	Should summary statistics 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.
...	Further arguments passed to posterior_epred, which is used in the computation of the R-squared values.

Details

For an introduction to the approach, see Gelman et al. (2019) and https://github.com/jgabry/bayes_R2/.

Value

If summary = TRUE, an M x C matrix is returned (M = number of response variables and c = length(probs) + 2) containing summary statistics of the Bayesian R-squared values. If summary = FALSE, the posterior draws of the Bayesian R-squared values are returned in an S x M matrix (S is the number of draws).

References

Andrew Gelman, Ben Goodrich, Jonah Gabry & Aki Vehtari. (2019). R-squared for Bayesian regression models, *The American Statistician*, 73(3):307-309. 10.1080/00031305.2018.1549100 (Preprint available at https://acris.aalto.fi/ws/portalfiles/portal/34206843/bayes_R2_v3.pdf)

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)
```

BetaBinomial *The Beta-binomial Distribution*

Description

Cumulative density & mass functions, and random number generation for the Beta-binomial distribution using the following re-parameterisation of the Stan Beta-binomial definition:

- `mu = alpha * beta` mean probability of trial success.
- `phi = (1 - mu) * beta` precision or over-dispersion, component.

Usage

```
dbeta_binomial(x, size, mu, phi, log = FALSE)

pbeta_binomial(q, size, mu, phi, lower.tail = TRUE, log.p = FALSE)

rbeta_binomial(n, size, mu, phi)
```

Arguments

`x, q`	Vector of quantiles.
`size`	Vector of number of trials (zero or more).
`mu`	Vector of means.
`phi`	Vector of precisions.
`log`	Logical; If `TRUE`, values are returned on the log scale.
`lower.tail`	Logical; If `TRUE` (default), return P(X <= x). Else, return P(X > x) .
`log.p`	Logical; If `TRUE`, values are returned on the log scale.
`n`	Number of draws to sample from the distribution.

`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, recompile = FALSE, ...)
```

Arguments

`samples`	A `brmsfit` object.
`recompile`	Logical, indicating whether the Stan model should be recompiled. This may be necessary if you are running bridge sampling on another machine than the one used to fit the model. No recompilation is done by default.
`...`	Additional arguments passed to `bridge_sampler.stanfit`.

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_pars = save_pars(all = TRUE)` in the call to `brm`, if you are planning to apply `bridge_sampler` to your models.

The computation of marginal likelihoods based on bridge sampling requires a lot more posterior draws than usual. A good conservative rule of thump is perhaps 10-fold more draws (read: the default of 4000 draws may not be enough in many cases). If not enough posterior draws are provided, the bridge sampling algorithm tends to be unstable leading to considerably different results each time it is run. We thus recommend running `bridge_sampler` multiple times to check the stability of the results.

More details are provided under `bridgesampling::bridge_sampler`.

See Also

`bayes_factor`, `post_prob`

Examples

```
## Not run:
# model with the treatment effect
fit1 <- brm(
```

```
  count ~ zAge + zBase + Trt,
  data = epilepsy, family = negbinomial(),
  prior = prior(normal(0, 1), class = b),
  save_pars = save_pars(all = TRUE)
)
summary(fit1)
bridge_sampler(fit1)

# model without the treatment effect
fit2 <- brm(
  count ~ zAge + zBase,
  data = epilepsy, family = negbinomial(),
  prior = prior(normal(0, 1), class = b),
  save_pars = save_pars(all = 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, extended-support beta regression, 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,
  data2 = NULL,
  cov_ranef = NULL,
  sample_prior = "no",
```

```
  sparse = NULL,
  knots = NULL,
  drop_unused_levels = TRUE,
  stanvars = NULL,
  stan_funs = NULL,
  fit = NA,
  save_pars = getOption("brms.save_pars", NULL),
  save_ranef = NULL,
  save_mevars = NULL,
  save_all_pars = NULL,
  init = NULL,
  inits = NULL,
  chains = 4,
  iter = getOption("brms.iter", 2000),
  warmup = floor(iter/2),
  thin = 1,
  cores = getOption("mc.cores", 1),
  threads = getOption("brms.threads", NULL),
  opencl = getOption("brms.opencl", NULL),
  normalize = getOption("brms.normalize", TRUE),
  control = NULL,
  algorithm = getOption("brms.algorithm", "sampling"),
  backend = getOption("brms.backend", "rstan"),
  future = getOption("future", FALSE),
  silent = 1,
  seed = NA,
  save_model = NULL,
  stan_model_args = list(),
  file = NULL,
  file_compress = TRUE,
  file_refit = getOption("brms.file_refit", "never"),
  empty = FALSE,
  rename = 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 data.frame (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 link 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 brmsprior objects created by set_prior or related functions and combined using the c method or the + operator. See also default_prior for more help.

autocor — (Deprecated) 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 NULL, corresponding to no correlations. In multivariate models, autocor might also be a list of autocorrelation structures. It is now recommend to specify autocorrelation terms directly within formula. See brmsformula for more details.

data2 — A named list of objects containing data, which cannot be passed via argument data. Required for some objects used in autocorrelation structures to specify dependency structures as well as for within-group covariance matrices.

cov_ranef — (Deprecated) 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. It is now recommended to specify those matrices in the formula interface using the gr and related functions. See vignette("brms_phylogenetics") for more details.

sample_prior — Indicate if draws from priors should be drawn additionally to the posterior draws. Options are "no" (the default), "yes", and "only". Among others, these draws can be used to calculate Bayes factors for point hypotheses via hypothesis. Please note that improper priors are not sampled, including the default improper priors used by brm. See set_prior on how to set (proper) priors. Please also note that prior draws for the overall intercept are not obtained by default for technical reasons. See brmsformula how to obtain prior draws for the intercept. If sample_prior is set to "only", draws are drawn solely from the priors ignoring the likelihood, which allows among others to generate draws from the prior predictive distribution. In this case, all parameters must have proper priors.

sparse — (Deprecated) 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. It is now recommended to use the sparse argument of brmsformula and related functions.

knots — Optional list containing user specified knot values to be used for basis construction of smoothing terms. See gamm for more details.

drop_unused_levels — Should unused factors levels in the data be dropped? Defaults to TRUE.

stanvars — An optional stanvars object generated by function stanvar to define additional variables for use in **Stan**'s program blocks.

stan_funs — (Deprecated) An optional character string containing self-defined **Stan** functions, which will be included in the functions block of the generated **Stan** code. It is now recommended to use the stanvars argument for this purpose instead.

`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_pars` An object generated by `save_pars` controlling which parameters should be saved in the model. The argument has no impact on the model fitting itself.

`save_ranef` (Deprecated) 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` (Deprecated) A flag to indicate if draws of latent noise-free variables obtained by using `me` and `mi` terms should be saved (default is `FALSE`). Saving these draws 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` (Deprecated) A flag to indicate if draws from all variables defined in Stan's `parameters` block should be saved (default is `FALSE`). Saving these draws is required in order to apply the methods `bridge_sampler`, `bayes_factor`, and `post_prob`. Can be set globally for the current R session via the `"brms.save_pars"` option (see `options`).

`init` Initial values for the sampler. If `NULL` (the default) or `"random"`, Stan will randomly generate initial values for parameters in a reasonable range. If `0`, all parameters are initialized to zero on the unconstrained space. This option is sometimes useful for certain families, as it happens that default random initial values cause draws to be essentially constant. Generally, setting `init = 0` is worth a try, if chains do not initialize or behave well. Alternatively, `init` 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 but are available to users if necessary. If specifying initial values using a list or a function then currently the parameter names must correspond to the names used in the generated Stan code (not the names used in R). For more details on specifying initial values you can consult the documentation of the selected `backend`.

`inits` (Deprecated) Alias of `init`.

`chains` Number of Markov chains (defaults to 4).

`iter` Number of total iterations per chain (including warmup; defaults to 2000). Can be set globally for the current R session via the `"brms.iter"` option (see `options`).

`warmup` A positive integer specifying number of warmup (aka burnin) iterations. This also specifies the number of iterations used for stepsize adaptation, so warmup draws 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 `mc.cores` 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.

threads — Number of threads to use in within-chain parallelization. For more control over the threading process, threads may also be a brmsthreads object created by threading. Within-chain parallelization is experimental! We recommend its use only if you are experienced with Stan's reduce_sum function and have a slow running model that cannot be sped up by any other means. Can be set globally for the current R session via the "brms.threads" option (see options).

opencl — The platform and device IDs of the OpenCL device to use for fitting using GPU support. If you don't know the IDs of your OpenCL device, c(0,0) is most likely what you need. For more details, see opencl. Can be set globally for the current R session via the "brms.opencl" option

normalize — Logical. Indicates whether normalization constants should be included in the Stan code (defaults to TRUE). Setting it to FALSE requires Stan version >= 2.25 to work. If FALSE, sampling efficiency may be increased but some post processing functions such as bridge_sampler will not be available. Can be controlled globally for the current R session via the 'brms.normalize' option.

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 naming the estimation approach to use. Options are "sampling" for MCMC (the default), "meanfield" for variational inference with independent normal distributions, "fullrank" for variational inference with a multivariate normal distribution, "pathfinder" for the pathfinder algorithm, "laplace" for the laplace approximation, or "fixed_param" for sampling from fixed parameter values. Can be set globally for the current R session via the "brms.algorithm" option (see options).

backend — Character string naming the package to use as the backend for fitting the Stan model. Options are "rstan" (the default) or "cmdstanr". Can be set globally for the current R session via the "brms.backend" option (see options). Details on the **rstan** and **cmdstanr** packages are available at https://mc-stan.org/rstan/ and https://mc-stan.org/cmdstanr/, respectively. Additionally a "mock" backend is available to make testing **brms** and packages that depend on it easier. The "mock" backend does not actually do any fitting, it only checks the generated Stan code for correctness and then returns whatever is passed in an additional mock_fit argument as the result of the fit.

future — Logical; If TRUE, the **future** package is used for parallel execution of the chains and argument cores will be ignored. Can be set globally for the current R session via the "future" option. The execution type is controlled via plan (see the examples section below).

silent — Verbosity level between 0 and 2. If 1 (the default), most of the informational messages of compiler and sampler are suppressed. If 2, even more messages are suppressed. The actual sampling progress is still printed. Set refresh = 0 to turn this off as well. If using backend = "rstan" you can also set open_progress = FALSE to prevent opening additional progress bars.

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's Stan code is saved via cat in a text file named after the string supplied in save_model.

stan_model_args — A list of further arguments passed to rstan::stan_model for backend = "rstan" or to cmdstanr::cmdstan_model for backend = "cmdstanr", which allows to change how models are compiled.

file — Either NULL or a character string. In the latter case, the fitted model object is saved via saveRDS in a file named after the string supplied in file. The .rds extension is added automatically. If the file already exists, brm will load and return the saved model object instead of refitting the model. Unless you specify the file_refit argument as well, the existing files won't be overwritten, you have to manually remove the file in order to refit and save the model under an existing file name. The file name is stored in the brmsfit object for later usage.

file_compress — Logical or a character string, specifying one of the compression algorithms supported by saveRDS. If the file argument is provided, this compression will be used when saving the fitted model object.

file_refit — Modifies when the fit stored via the file argument is re-used. Can be set globally for the current R session via the "brms.file_refit" option (see options). For "never" (default) the fit is always loaded if it exists and fitting is skipped. For "always" the model is always refitted. If set to "on_change", brms will refit the model if model, data or algorithm as passed to Stan differ from what is stored in the file. This also covers changes in priors, sample_prior, stanvars, covariance structure, etc. If you believe there was a false positive, you can use brmsfit_needs_refit to see why refit is deemed necessary. Refit will not be triggered for changes in additional parameters of the fit (e.g., initial values, number of iterations, control arguments, ...). A known limitation is that a refit will be triggered if within-chain parallelization is switched on/off.

empty — Logical. If TRUE, the Stan model is not created and compiled and the corresponding 'fit' slot of the brmsfit object will be empty. This is useful if you have estimated a brms-created Stan model outside of **brms** and want to feed it back into the package.

rename — For internal use only.

... — Further arguments passed to Stan. For backend = "rstan" the arguments are passed to sampling or vb. For backend = "cmdstanr" the arguments are passed to the cmdstanr::sample or cmdstanr::variational method.

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 `default_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 draws, 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 draws. 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 draws.

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 draws. 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 draws along with many other useful information about the model. Use `methods(class = "brmsfit")` for an overview on available methods.

Parallelization with multiple CPU cores

brms can make use of multiple CPU cores in parallel to speed up computations in various ways. For efficient use of the available resources it is recommended to only use parallelism to an extend such that the available physical CPUs are not oversubscribed. For example, when you have 8 CPU cores locally available, then you may consider to run 4 chains with 2 threads per chain for best performance if you happen to just run a single model. In case you run a simulation study which requires to run many times a given model, then neither chain nor within-chain parallelization is advisable as the computational resources are already exhausted by the simulation study and any further parallelization beyond the simulation study itself will in fact slow down the overall runtime. Please be aware that for historical reasons the nomenclature of the arguments is possibly confusing. The `cores` argument refers to running different chains in parallel and the within-chain parallelization will allocate for each chain as many threads as requested. The requested threads therefore increase the use of overall CPUs in a multiplicative way.

For more advanced parallelization (including beyond single model fits), **brms** also integrates with the **future** package. Importantly, this enables seamless integration with the **mirai** parallelization framework through the use of the **future.mirai** adapter. With **mirai** local and remote machines can be used in a fully transparent manner to the user. This includes the possibility to use large number

of remote machines running in the context of a computer cluster, which are managed with queuing systems. Please refer to the section on distributed computing of mirai::daemons.

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 (2018). Advanced Bayesian Multilevel Modeling with the R Package brms. *The R Journal*. 10(1), 395–411. doi:10.32614/RJ-2018-017

See Also

brms, brmsformula, brmsfamily, brmsfit

Examples

```
## Not run:
# Poisson regression for the number of seizures in epileptic patients
fit1 <- brm(
  count ~ zBase * Trt + (1|patient),
  data = epilepsy, family = poisson(),
  prior = prior(normal(0, 10), class = b) +
    prior(cauchy(0, 2), class = sd)
)

# generate a summary of the results
summary(fit1)

# plot the MCMC chains as well as the posterior distributions
plot(fit1)

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

# plot conditional effects for each predictor
plot(conditional_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(conditional_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)


# Non-linear Gaussian model
fit5 <- brm(
  bf(cum ~ ult * (1 - exp(-(dev/theta)^omega)),
     ult ~ 1 + (1|AY), omega ~ 1, theta ~ 1,
     nl = TRUE),
  data = loss, family = gaussian(),
  prior = c(
    prior(normal(5000, 1000), nlpar = "ult"),
    prior(normal(1, 2), nlpar = "omega"),
    prior(normal(45, 10), nlpar = "theta")
  ),
  control = list(adapt_delta = 0.9)
)
summary(fit5)
conditional_effects(fit5)


# 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)
conditional_effects(fit6)

# extract estimated residual SDs of both groups
sigmas <- exp(as.data.frame(fit6, variable = "^b_sigma_", regex = TRUE))
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)
conditional_effects(fit7)

# use the future package for more flexible parallelization
library(future)
plan(multisession, workers = 4)
fit7a <- update(fit7, future = TRUE)

# we may also use mirai allowing the use of remote machines as
# well, including the use of computer clusters managed by queuing
# systems
mirai::daemons(4) # distributed computing with url and remote argument
plan(future.mirai::mirai_cluster)
fit7b <- update(fit7, future = TRUE)
mirai::daemons(0) # shuts down the 4 workers

# fit a model manually via rstan
scode <- stancode(count ~ Trt, data = epilepsy)
sdata <- standata(count ~ Trt, data = epilepsy)
stanfit <- rstan::stan(model_code = scode, data = sdata)
# feed the Stan model back into brms
fit8 <- brm(count ~ Trt, data = epilepsy, empty = TRUE)
fit8$fit <- stanfit
fit8 <- rename_pars(fit8)
summary(fit8)

## End(Not run)
```

`brmsfamily` *Special Family Functions for* **brms** *Models*

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 = "flexible",
  refcat = NULL
)

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

bernoulli(link = "logit")

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

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")

xbeta(link = "logit", link_phi = "log", link_kappa = "log")

dirichlet(link = "logit", link_phi = "log", refcat = NULL)

logistic_normal(link = "identity", link_sigma = "log", refcat = NULL)

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

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

cox(link = "log")

hurdle_poisson(link = "log", link_hu = "logit")

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")

hurdle_cumulative(
  link = "logit",
  link_hu = "logit",
  link_disc = "log",
  threshold = "flexible"
)

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")

zero_inflated_beta_binomial(
  link = "logit",
  link_phi = "log",
  link_zi = "logit"
)

categorical(link = "logit", refcat = NULL)

multinomial(link = "logit", refcat = NULL)

dirichlet_multinomial(link = "logit", link_phi = "log", refcat = NULL)

cumulative(link = "logit", link_disc = "log", threshold = "flexible")

sratio(link = "logit", link_disc = "log", threshold = "flexible")

cratio(link = "logit", link_disc = "log", threshold = "flexible")

acat(link = "logit", link_disc = "log", threshold = "flexible")
```

Arguments

family	A character string naming the distribution family of the response variable to be used in the model. Currently, the following families are supported: gaussian, student, binomial, bernoulli, beta-binomial, poisson, negbinomial, geometric, Gamma, skew_normal, lognormal, shifted_lognormal, exgaussian, wiener, inverse.gaussian, exponential, weibull, frechet, Beta, dirichlet, von_mises, asym_laplace, gen_extreme_value, categorical, multinomial, dirichlet_multinomial, cumulative, cratio, sratio, acat, hurdle_poisson, hurdle_negbinomial, hurdle_gamma, hurdle_lognormal, hurdle_cumulative, zero_inflated_binomial, zero_inflated_beta_binomial, zero_inflated_beta, zero_inflated_negbinomial, zero_inflated_poisson, zero_one_inflated_beta, and xbeta.
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, "equidistant" restricts the distance between consecutive thresholds to the same value, and "sum_to_zero" ensures the thresholds sum to zero.
refcat	Optional name of the reference response category used in categorical, multinomial, dirichlet, dirichlet_multinomial and logistic_normal models. If NULL (the default), the first category is used as the reference. If NA, all categories will be predicted, which requires strong priors or carefully specified predictor terms in order to lead to an identified model.

Details

Below, we list common use cases for the different families. This list is not ment to be exhaustive.

- Family gaussian can be used for linear regression.
- Family student can be used for robust linear regression that is less influenced by outliers.
- Family skew_normal can handle skewed responses in linear regression.
- Families poisson, negbinomial, and geometric can be used for regression of unbounded count data.
- Families bernoulli, binomial, and beta_binomial can be used for binary regression (i.e., most commonly logistic regression).
- Families categorical, multinomial and dirichlet_multinomial can be used for 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, inverse.gaussian, and cox (Cox proportional hazards model) can be used (among others) for time-to-event regression also known as survival regression.
- Families weibull, frechet, and gen_extreme_value ('generalized extreme value') allow for modeling extremes.
- Families beta, dirichlet, and logistic_normal can be used to model responses representing rates or probabilities.

- Family xbeta extends the beta family to support [0, 1] responses with exact 0s and / or 1s, when each response takes values 0, 1, and (0, 1) according to a single process. If there is merit in assuming that 0 and 1 values arise from different processes than (0, 1) values, then the zero_inflated_beta, zero_one_inflated_beta families provide more flexibility. For details see Kosmidis & Zeileis (2024).
- 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.
- Family wiener 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_binomial, zero_inflated_beta, zero_one_inflated_beta, and hurdle_cumulative 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.

Below, we list all possible links for each family. The first link mentioned for each family is the default.

- Families gaussian, student, skew_normal, exgaussian, asym_laplace, and gen_extreme_value support the links (as names) identity, log, inverse, and softplus.
- Families poisson, negbinomial, geometric, zero_inflated_poisson, zero_inflated_negbinomial, hurdle_poisson, and hurdle_negbinomial support log, identity, sqrt, and softplus.
- Families binomial, bernoulli, beta_binomial, zero_inflated_binomial, zero_inflated_beta_binomial, Beta, zero_inflated_beta, zero_one_inflated_beta, and xbeta support logit, probit, probit_approx, cloglog, cauchit, identity, and log.
- Families cumulative, cratio, sratio, acat, and hurdle_cumulative support logit, probit, probit_approx, cloglog, and cauchit.
- Families categorical, multinomial, dirichlet_multinomial and dirichlet support logit.
- Families Gamma, weibull, exponential, frechet, and hurdle_gamma support log, identity, inverse, and softplus.
- Families lognormal and hurdle_lognormal support identity and inverse.
- Family logistic_normal supports identity.
- Family inverse.gaussian supports 1/mu^2, inverse, identity, log, and softplus.
- Family von_mises supports tan_half and identity.
- Family cox supports log, identity, and softplus for the proportional hazards parameter.
- Family wiener supports identity, log, and softplus for the main parameter which represents the drift rate.

Please note that when calling the Gamma family function of the **stats** package, the default link will be inverse instead of log although the latter is the default in **brms**. Also, when using the family functions gaussian, binomial, poisson, and Gamma of the **stats** package (see family), special link functions such as softplus or cauchit won't work. In this case, you have to use brmsfamily to specify the family with corresponding link function.

References

Kosmidis I, Zeileis A (2024). Extended-Support Beta Regression for [0, 1] Responses. *arXiv Preprint*. doi:10.48550/arXiv.2409.07233

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 draws (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.

data A data.frame containing all variables used in the model.

data2 A list of data objects which cannot be passed via data.

prior A brmsprior object containing information on the priors used in the model.

stanvars A stanvars object.

model The model code in **Stan** language.

exclude The names of the parameters for which draws are not saved.

algorithm The name of the algorithm used to fit the model.

backend The name of the backend used to fit the model.

threads An object of class 'brmsthreads' created by threading.

opencl An object of class 'brmsopencl' created by opencl.

`stan_args` Named list of additional control arguments that were passed to the Stan backend directly.

`fit` An object of class `stanfit` among others containing the posterior draws.

`basis` An object that contains a small subset of the Stan data created at fitting time, which is needed to process new data correctly.

`criteria` An empty `list` for adding model fit criteria after estimation of the model.

`file` Optional name of a file in which the model object was stored in or loaded from.

`version` The versions of **brms** and **rstan** with which the model was fitted.

`family` (Deprecated) A `brmsfamily` object.

`autocor` (Deprecated) An `cor_brms` object containing the autocorrelation structure if specified.

`ranef` (Deprecated) A `data.frame` containing the group-level structure.

`cov_ranef` (Deprecated) A `list` of customized group-level covariance matrices.

`stan_funs` (Deprecated) A character string of length one or `NULL`.

`data.name` (Deprecated) The name of `data` as specified by the user.

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,
  loop = NULL,
  center = NULL,
  cmc = NULL,
  sparse = NULL,
  decomp = NULL,
  unused = 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. Alternatively, it is possible to fix parameters to certain values by passing numbers or character strings in which case arguments have to be named to provide the parameter names. See 'Details' for more information.
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 other functions.
autocor	An optional formula which contains autocorrelation terms as described in autocor-terms or alternatively a cor_brms object (deprecated). If autocor is specified in brmsformula, it will overwrite the value specified in other functions.
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.
loop	Logical; Only used in non-linear models. Indicates if the computation of the non-linear formula should be done inside (TRUE) or outside (FALSE) a loop over observations. Defaults to TRUE.
center	Logical; Indicates if the population-level design matrix should be centered, which usually increases sampling efficiency. See the 'Details' section for more information. Defaults to TRUE for distributional parameters and to FALSE for non-linear parameters.
cmc	Logical; Indicates whether automatic cell-mean coding should be enabled when removing the intercept by adding 0 to the right-hand of model formulas. Defaults to TRUE to mirror the behavior of standard R formula parsing.
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.
decomp	Optional name of the decomposition used for the population-level design matrix. Defaults to NULL that is no decomposition. Other options currently available are "QR" for the QR decomposition that helps in fitting models with highly correlated predictors.
unused	An optional formula which contains variables that are unused in the model but should still be stored in the model's data frame. This can be useful, for example, if those variables are required for post-processing the model.

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' or 'overall' 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' or 'varying' 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. Equivalently, the `cor` argument of the `gr` function can be used for this purpose, for example, `(1 + x || g)` is equivalent to `(1 + x | gr(g, cor = FALSE))`.

It is also 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|i|g)` and `(1+z|i|g)` somewhere in the formulas passed to `brmsformula`, correlations between the corresponding group-level effects will be estimated. In the above example, `i` is not a variable in the data but just a symbol to indicate correlations between multiple group-level terms. Equivalently, the `id` argument of the `gr` function can be used as well, for example, `(1 + x | gr(g, id = "i"))`.

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 ~ x + (1 | gr(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

Flexible non-linear smooth terms can 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 or approximations need to be used.

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 `sdx`, we can simply include it on the right-hand side of the model formula via `y ~ me(x, sdx)`. This can easily be extended to more general formulas. If `x2` is another measured predictor with corresponding error `sdx2` 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 ~ me(x, sdx) * me(x2, sdx2) * z`. The `me` function is soft deprecated in favor of the more flexible and consistent `mi` function (see below).

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 ~ 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() ~ z`. Measurement error may be included via the `sdy` argument, say, `x | mi(sdy = se) ~ z`. See `mi` for examples with real data.

Autocorrelation terms

Autocorrelation terms can be directly specified inside the `pterms` part as well. Details can be found in `autocor-terms`.

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`, `subset`, `cens`, `trunc`, `trials`, `cat`, `dec`, `rate`, `vreal`, or `vint`. 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 `yi` 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 `yi | se(sei) ~ 1` and `yi | se(sei) ~ 1`

+ (1|study), respectively, where study is a variable uniquely identifying every study. If desired, meta-regression can be performed via yi | se(sei) ~ 1 + mod1 + mod2 + (1|study) or yi | se(sei) ~ 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, yi | se(sei, sigma = TRUE) ~ 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 wei contains the weights and that yi is the response variable. Then, formula yi | weights(wei) ~ predictors implements a weighted regression.

For multivariate models, subset may be used in the aterms part, to use different subsets of the data in different univariate models. For instance, if sub is a logical variable and y is the response of one of the univariate models, we may write y | subset(sub) ~ predictors so that y is predicted only for those observations for which sub evaluates to TRUE.

For log-linear models such as Poisson models, rate may be used in the aterms part to specify the denominator of a response that is expressed as a rate. The numerator is given by the actual response variable and has a distribution according to the family as usual. In Poisson models, using rate(denom) is equivalent to adding offset(log(denom)) to the linear predictor of the main parameter. For negative-binomial models, this equivalence no longer holds and rate(denom) remains the statistically correct approach.

With the exception of categorical and ordinal families, left, right, and interval censoring can be modeled through y | cens(censored) ~ 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 y2) has to be passed to cens. In this case, the formula has the structure y | cens(censored, y2) ~ 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 and ordinal 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). **Please note that the** cbind() **syntax will not work in brms in the expected way because this syntax is reserved for other purposes.**

For all ordinal families, aterms may contain a term thres(number) to specify the number thresholds (e.g, thres(6)), which should be equal to the total number of response categories - 1. If not given, the number of thresholds is calculated from the data. If different threshold vectors should be used for different subsets of the data, the gr argument can be used to provide the grouping variable

(e.g, `thres(6, gr = item)`, if `item` is the grouping variable). In this case, the number of thresholds can also be a variable in the data with different values per group.

A deprecated quasi alias of `thres()` is `cat()` with which the total number of response categories (i.e., number of thresholds + 1) can be specified.

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'`.

All families support the `index` addition term to uniquely identify each observation of the corresponding response variable. Currently, `index` is primarily useful in combination with the `subset` addition and `mi` terms.

For custom families, it is possible to pass an arbitrary number of real and integer vectors via the addition terms `vreal` and `vint`, respectively. An example is provided in `vignette('brms_customfamilies')`. To pass multiple vectors of the same data type, provide them separated by commas inside a single `vreal` or `vint` statement.

Multiple addition terms of different types may be specified at the same time using the + operator. For example, the formula `formula = yi | se(sei) + cens(censored) ~ 1` implies 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

By default, 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 `<X_means, b>`, where `<,>` 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. You can turn off this special handling of the intercept by setting argument `center` to `FALSE`. 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 `alpha`, `beta`, and `lambda`. This is certainly a non-linear model being defined via `formula = y ~ 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 `nl` 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 ~ 1` or equivalently (and more flexible) `alpha ~ 1, beta ~ 1, lambda ~ 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 ~ 1, beta ~ 1 + z + (1|g), lambda ~ 1`. The formula syntax described above applies here as well. In this example, we are using `z` and `g` only for the prediction of `beta`, but we might also use them for the other non-linear parameters (provided that the resulting model is still scientifically reasonable).

By default, non-linear covariates are treated as real vectors in Stan. However, if the data of the covariates is of type 'integer' in R (which can be enforced by the 'as.integer' function), the Stan type will be changed to an integer array. That way, covariates can also be used for indexing purposes in Stan.

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 `sigma` in gaussian models or the hurdle probability `hu` in hurdle models. The syntax closely resembles that of a non-linear parameter, for instance `sigma ~ x + s(z) + (1+x|g)`. For some examples of distributional models, see `vignette("brms_distreg")`.

Parameter `mu` exists for every family and can be used as an alternative to specifying terms in `formula`. If both `mu` and `formula` are given, the right-hand side of `formula` is ignored. Accordingly, specifying terms on the right-hand side of both `formula` and `mu` at the same time is deprecated. In future versions, `formula` might be updated by `mu`.

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`, `zero_inflated_beta`, and `xbeta` 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 the can only be within the unit interval.

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 so-called 'softmax' transformation 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 mvbind notation or with help of the mvbf function. Suppose that y1 and y2 are response variables and x is a predictor. Then mvbind(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, mvbind(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: mvbind(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)
# alternative but equivalent way to specify the above model
bf(y ~ a1 - a2^x, a1 ~ 1 + (1 | gr(g, id = 2)),
   a2 ~ x + (x | gr(g, id = 2)), nl = TRUE)

# define a multivariate model
bf(mvbind(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()

# specify correlated residuals of a gaussian and a poisson model
form1 <- bf(y1 ~ 1 + x + (1|c|obs), sigma = 1) + gaussian()
form2 <- bf(y2 ~ 1 + x + (1|c|obs)) + poisson()

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

# model sigma as a function of the mean
bf(y ~ eta, nl = TRUE) +
  lf(eta ~ 1 + x) +
  nlf(sigma ~ tau * sqrt(eta)) +
  lf(tau ~ 1)
```

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, loop = NULL)

lf(
  ...,
  flist = NULL,
  dpar = NULL,
  resp = NULL,
  center = NULL,
  cmc = NULL,
  sparse = NULL,
  decomp = NULL
)

acformula(autocor, 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. Alternatively, it is possible to fix parameters to certain values by passing numbers or character strings in which case arguments have to be named to provide the parameter names. See 'Details' for more information.
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.

loop	Logical; Only used in non-linear models. Indicates if the computation of the non-linear formula should be done inside (TRUE) or outside (FALSE) a loop over observations. Defaults to TRUE.
center	Logical; Indicates if the population-level design matrix should be centered, which usually increases sampling efficiency. See the 'Details' section for more information. Defaults to TRUE for distributional parameters and to FALSE for non-linear parameters.
cmc	Logical; Indicates whether automatic cell-mean coding should be enabled when removing the intercept by adding 0 to the right-hand of model formulas. Defaults to TRUE to mirror the behavior of standard R formula parsing.
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.
decomp	Optional name of the decomposition used for the population-level design matrix. Defaults to NULL that is no decomposition. Other options currently available are "QR" for the QR decomposition that helps in fitting models with highly correlated predictors.
autocor	A one sided formula containing autocorrelation terms. All none autocorrelation terms in autocor will be silently ignored.
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 me 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)) +
  lf(a ~ x, b ~ z + (1|g)) +
  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)

# add autocorrelation terms
bf(y ~ x) + acformula(~ arma(p = 1, q = 1) + car(W))
```

---

brmshypothesis *Descriptions of* brmshypothesis *Objects*

Description

A brmshypothesis object contains posterior draws 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,
  short = getOption("brms.short_summary", FALSE),
  ...
)

## S3 method for class 'brmshypothesis'
plot(
  x,
  nvariables = 5,
  N = NULL,
  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.
short	A flag indicating whether to provide a shorter summary with less informational text. Defaults to FALSE. Can be set globally for the current session via the brms.short_summary option.
...	Currently ignored.
nvariables	The number of variables (parameters) plotted per page.
N	Deprecated alias of nvariables.
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 plot 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 draws.

See Also

hypothesis

---

brmsterms *Parse Formulas of* **brms** *Models*

Description

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

Usage

```
brmsterms(formula, ...)

## Default S3 method:
brmsterms(formula, ...)

## S3 method for class 'brmsformula'
brmsterms(formula, check_response = TRUE, resp_rhs_all = TRUE, ...)

## S3 method for class 'mvbrmsformula'
brmsterms(formula, ...)
```

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.
...	Further arguments passed to or from other methods.
check_response	Logical; Indicates whether the left-hand side of formula (i.e. response variables and addition arguments) should be parsed. If FALSE, 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 brmsterms.

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

brm_multiple *Run the same* **brms** *model on multiple datasets*

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. Models can be run in parallel using the **future** package.

Usage

```
brm_multiple(
  formula,
  data,
  family = gaussian(),
  prior = NULL,
  data2 = NULL,
  autocor = NULL,
  cov_ranef = NULL,
  sample_prior = c("no", "yes", "only"),
  sparse = NULL,
  knots = NULL,
  stanvars = NULL,
  stan_funs = NULL,
  silent = 1,
  recompile = FALSE,
  combine = TRUE,
  fit = NA,
  algorithm = getOption("brms.algorithm", "sampling"),
  seed = NA,
  file = NULL,
  file_compress = TRUE,
  file_refit = getOption("brms.file_refit", "never"),
  ...
)
```

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 data.frames each of which will be used to fit a separate model. Alternatively, a mids object from the **mice** package.
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 link 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 brmsprior objects created by set_prior or related functions and combined using the c method or the + operator. See also default_prior for more help.

data2 — A *list* of named lists each of which will be used to fit a separate model. Each of the named lists contains objects representing data which cannot be passed via argument data (see brm for examples). The length of the outer list should match the length of the list passed to the data argument.

autocor — (Deprecated) 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 NULL, corresponding to no correlations. In multivariate models, autocor might also be a list of autocorrelation structures. It is now recommend to specify autocorrelation terms directly within formula. See brmsformula for more details.

cov_ranef — (Deprecated) 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. It is now recommended to specify those matrices in the formula interface using the gr and related functions. See vignette("brms_phylogenetics") for more details.

sample_prior — Indicate if draws from priors should be drawn additionally to the posterior draws. Options are "no" (the default), "yes", and "only". Among others, these draws can be used to calculate Bayes factors for point hypotheses via hypothesis. Please note that improper priors are not sampled, including the default improper priors used by brm. See set_prior on how to set (proper) priors. Please also note that prior draws for the overall intercept are not obtained by default for technical reasons. See brmsformula how to obtain prior draws for the intercept. If sample_prior is set to "only", draws are drawn solely from the priors ignoring the likelihood, which allows among others to generate draws from the prior predictive distribution. In this case, all parameters must have proper priors.

sparse — (Deprecated) 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. It is now recommended to use the sparse argument of brmsformula and related functions.

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 for use in **Stan**'s program blocks.

stan_funs — (Deprecated) An optional character string containing self-defined **Stan** functions, which will be included in the functions block of the generated **Stan** code. It is now recommended to use the stanvars argument for this purpose instead.

silent — Verbosity level between 0 and 2. If 1 (the default), most of the informational messages of compiler and sampler are suppressed. If 2, even more messages are suppressed. The actual sampling progress is still printed. Set refresh = 0 to turn this off as well. If using backend = "rstan" you can also set open_progress = FALSE to prevent opening additional progress bars.

recompile — Logical, indicating whether the Stan model should be recompiled for every imputed data set. Defaults to FALSE. If NULL, brm_multiple tries to figure out internally, if recompilation is necessary, for example because data-dependent priors have changed. Using the default of no recompilation should be fine in most cases.

combine — Logical; Indicates if the fitted models should be combined into a single fitted model object via combine_models. Defaults to TRUE.

fit — An instance of S3 class brmsfit_multiple derived from a previous fit; defaults to NA. If fit is of class brmsfit_multiple, 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.

algorithm — Character string naming the estimation approach to use. Options are "sampling" for MCMC (the default), "meanfield" for variational inference with independent normal distributions, "fullrank" for variational inference with a multivariate normal distribution, "pathfinder" for the pathfinder algorithm, "laplace" for the laplace approximation, or "fixed_param" for sampling from fixed parameter values. Can be set globally for the current R session via the "brms.algorithm" option (see options).

seed — The seed for random number generation to make results reproducible. If NA (the default), **Stan** will set the seed randomly.

file — Either NULL or a character string. In the latter case, the fitted model object is saved via saveRDS in a file named after the string supplied in file. The .rds extension is added automatically. If the file already exists, brm will load and return the saved model object instead of refitting the model. Unless you specify the file_refit argument as well, the existing files won't be overwritten, you have to manually remove the file in order to refit and save the model under an existing file name. The file name is stored in the brmsfit object for later usage.

file_compress — Logical or a character string, specifying one of the compression algorithms supported by saveRDS. If the file argument is provided, this compression will be used when saving the fitted model object.

file_refit — Modifies when the fit stored via the file argument is re-used. Can be set globally for the current R session via the "brms.file_refit" option (see options). For "never" (default) the fit is always loaded if it exists and fitting is skipped. For "always" the model is always refitted. If set to "on_change", brms will refit the model if model, data or algorithm as passed to Stan differ from what is stored in the file. This also covers changes in priors, sample_prior, stanvars, covariance structure, etc. If you believe there was a false positive, you can use brmsfit_needs_refit to see why refit is deemed necessary. Refit will not be triggered for changes in additional parameters of the fit (e.g., initial values, number of iterations, control arguments, ...). A known limitation is that a refit will be triggered if within-chain parallelization is switched on/off.

... 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, subset the draws belonging to the individual models and then run convergence diagnostics. See Examples below for details.

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.

Parallelization with multiple CPU cores

brms can make use of multiple CPU cores in parallel to speed up computations in various ways. For efficient use of the available resources it is recommended to only use parallelism to an extend such that the available physical CPUs are not oversubscribed. For example, when you have 8 CPU cores locally available, then you may consider to run 4 chains with 2 threads per chain for best performance if you happen to just run a single model. In case you run a simulation study which requires to run many times a given model, then neither chain nor within-chain parallelization is advisable as the computational resources are already exhausted by the simulation study and any further parallelization beyond the simulation study itself will in fact slow down the overall runtime. Please be aware that for historical reasons the nomenclature of the arguments is possibly confusing. The cores argument refers to running different chains in parallel and the within-chain parallelization will allocate for each chain as many threads as requested. The requested threads therefore increase the use of overall CPUs in a multiplicative way.

For more advanced parallelization (including beyond single model fits), **brms** also integrates with the **future** package. Importantly, this enables seamless integration with the **mirai** parallelization framework through the use of the **future.mirai** adapter. With **mirai** local and remote machines can be used in a fully transparent manner to the user. This includes the possibility to use large number of remote machines running in the context of a computer cluster, which are managed with queuing systems. Please refer to the section on distributed computing of mirai::daemons.

Examples

```
## Not run:
library(mice)
m <- 5
imp <- mice(nhanes2, m = m)

# 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, variable = "^b_", regex = TRUE)
```

```
# investigate convergence of the original models
library(posterior)
draws <- as_draws_array(fit_imp2)
# every dataset has just one chain here
draws_per_dat <- lapply(1:m, \(i) subset_draws(draws, chain = i))
lapply(draws_per_dat, summarise_draws, default_convergence_measures())

# use the future package for parallelization
library(future)
plan(multisession, workers = 4)
fit_imp3 <- brm_multiple(bmi ~ age + hyp + chl, data = imp, chains = 1)
summary(fit_imp3)

## End(Not run)
```

`car` *Spatial conditional autoregressive (CAR) structures*

Description

Set up an spatial conditional autoregressive (CAR) term in **brms**. The function does not evaluate its arguments – it exists purely to help set up a model with CAR terms.

Usage

```
car(M, gr = NA, type = "escar")
```

Arguments

`M`	Adjacency matrix of locations. All non-zero entries are treated as if the two locations are adjacent. If `gr` is specified, the row names of `M` have to match the levels of the grouping factor.
`gr`	An optional 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.
`type`	Type of the CAR structure. Currently implemented are `"escar"` (exact sparse CAR), `"esicar"` (exact sparse intrinsic CAR), `"icar"` (intrinsic CAR), and `"bym2"`. 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` and `bym2` type is implemented based on the case study of Mitzi Morris (https://mc-stan.org/users/documentation/case-studies/icar_stan.html).

Value

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

See Also

autocor-terms

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
rownames(W) <- 1:nrow(W)

# 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)
g <- 1:length(y)
dat <- data.frame(y, size, x1, x2, g)

# fit a CAR model
fit <- brm(y | trials(size) ~ x1 + x2 + car(W, gr = g),
           data = dat, data2 = list(W = W),
           family = binomial())
summary(fit)

## 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

object	An object of class brmsfit.
summary	Should summary statistics 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.
...	Further arguments passed to fixef.brmsfit and ranef.brmsfit.

Value

A list of 3D arrays (one per grouping factor). If summary is TRUE, the 1st dimension contains the factor levels, the 2nd dimension contains the summary statistics (see posterior_summary), and the 3rd dimension contains the group-level effects. If summary is FALSE, the 1st dimension contains the posterior draws, the 2nd dimension contains the factor levels, and the 3rd dimension contains the group-level effects.

Examples

```
## Not run:
fit <- brm(count ~ zAge + zBase * Trt + (1+Trt|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 usefully for instance when having manually run models in parallel.

Usage

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

Arguments

...	One or more brmsfit objects.
mlist	Optional list of one or more brmsfit objects.
check_data	Logical; indicates if the data should be checked for being the same across models (defaults to TRUE). Setting it to FALSE 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. Deprecated and will be removed in the future. Please use loo_compare instead.

Usage

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

Arguments

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

Details

See loo_compare for the recommended way of comparing models with the **loo** package.

Value

An object of class iclist.

See Also

loo, loo_compare add_criterion

Examples

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

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

# compare both models
compare_ic(waic1, waic2)

## End(Not run)
```

---

conditional_effects.brmsfit

Display Conditional Effects of Predictors

Description

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

Usage

```
## S3 method for class 'brmsfit'
conditional_effects(
  x,
  effects = NULL,
  conditions = NULL,
  int_conditions = NULL,
  re_formula = NA,
  prob = 0.95,
  robust = TRUE,
  method = "posterior_epred",
  spaghetti = FALSE,
  surface = FALSE,
  categorical = FALSE,
  ordinal = FALSE,
  transform = NULL,
  resolution = 100,
  select_points = 0,
  too_far = 0,
  probs = NULL,
  ...
)

conditional_effects(x, ...)

## S3 method for class 'brms_conditional_effects'
plot(
  x,
  ncol = NULL,
  points = getOption("brms.plot_points", FALSE),
  rug = getOption("brms.plot_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(),
  facet_args = list(),
  theme = NULL,
  ask = TRUE,
  plot = TRUE,
  ...
)
```

Arguments

x	An object of class brmsfit.
effects	An optional character vector naming effects (main effects or interactions) for which to compute conditional plots. Interactions are specified by a : between variable names. If NULL (the default), plots are generated for all main effects and two-way interactions estimated in the model. When specifying effects manually, *all* two-way interactions (including grouping variables) may be plotted even if not originally modeled.
conditions	An optional data.frame containing variable values to condition on. Each effect defined in effects will be plotted separately for each row of conditions. Values in the cond__ column will be used as titles of the subplots. If cond__ 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 NULL (the default), numeric variables will be conditionalized by using their means and factors will get their first level assigned. NA values within factors 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.
int_conditions	An optional named list whose elements are vectors of values of the variables specified in effects. At these values, predictions are evaluated. The names of int_conditions have to match the variable names exactly. Additionally, the elements of the 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 $mean$ and at $mean +/- sd$ for numeric predictors and at all categories for factor-like predictors.
re_formula	A formula containing group-level effects to be considered in the conditional predictions. If NULL, include all group-level effects; if NA (default), include no group-level effects.
prob	A value between 0 and 1 indicating the desired probability to be covered by the uncertainty intervals. The default is 0.95.
robust	If TRUE (the default) the median is used as the measure of central tendency. If FALSE the mean is used instead.
method	Method used to obtain predictions. Can be set to "posterior_epred" (the default), "posterior_predict", or "posterior_linpred". For more details, see the respective function documentations.
spaghetti	Logical. Indicates if predictions should be visualized via spaghetti plots. Only applied for numeric predictors. If TRUE, it is recommended to set argument ndraws 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.
categorical	Logical. Indicates if effects of categorical or ordinal models should be shown in terms of probabilities of response categories. Defaults to FALSE.

ordinal	(Deprecated) Please use argument categorical. Logical. Indicates if effects in ordinal models should be visualized as a raster with the response categories on the y-axis. Defaults to FALSE.
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 = "posterior_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.
probs	(Deprecated) The quantiles to be used in the computation of uncertainty intervals. Please use argument prob instead.
...	Further arguments such as draw_ids or ndraws passed to posterior_predict or posterior_epred.
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. Indicates if the original data points should be added via geom_jitter. Default is FALSE. Can be controlled globally via the brms.plot_points option. 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. Indicates if 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. Can be controlled globally via the brms.plot_rug option.
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.
facet_args	Only used if if multiple conditions are provided: A named list of arguments passed to facet_wrap.
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 conditional_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.

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 'brms_conditional_effects' which is a named list with one data.frame per effect containing all information required to generate conditional 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 ~ zAge + zBase * Trt + (1 | patient),
           data = epilepsy, family = poisson())

## plot all conditional effects
```

```
plot(conditional_effects(fit), ask = FALSE)

## change colours to grey scale
library(ggplot2)
ce <- conditional_effects(fit, "zBase:Trt")
plot(ce, plot = FALSE)[[1]] +
  scale_color_grey() +
  scale_fill_grey()

## only plot the conditional interaction effect of 'zBase:Trt'
## for different values for 'zAge'
conditions <- data.frame(zAge = c(-1, 0, 1))
plot(conditional_effects(fit, effects = "zBase:Trt",
                         conditions = conditions))

## also incorporate group-level effects variance over patients
## also add data points and a rug representation of predictor values
plot(conditional_effects(fit, effects = "zBase:Trt",
                         conditions = conditions, re_formula = NULL),
     points = TRUE, rug = TRUE)

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

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

## End(Not run)
```

`conditional_smooths.brmsfit` *Display Smooth Terms*

Description

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

Usage

```
## S3 method for class 'brmsfit'
conditional_smooths(
  x,
  smooths = NULL,
  int_conditions = NULL,
  prob = 0.95,
  spaghetti = FALSE,
  surface = TRUE,
  resolution = 100,
  too_far = 0,
  ndraws = NULL,
  draw_ids = NULL,
  nsamples = NULL,
  subset = NULL,
  probs = NULL,
  ...
)

conditional_smooths(x, ...)
```

Arguments

x	An object of class brmsfit.
smooths	Optional character vector of smooth terms to display. If NULL (the default) all smooth terms are shown.
int_conditions	An optional named list whose elements are vectors of values of the variables specified in effects. At these values, predictions are evaluated. The names of int_conditions have to match the variable names exactly. Additionally, the elements of the 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 $mean$ and at $mean +/- sd$ for numeric predictors and at all categories for factor-like predictors.
prob	A value between 0 and 1 indicating the desired probability to be covered by the uncertainty intervals. The default is 0.95.
spaghetti	Logical. Indicates if predictions should be visualized via spaghetti plots. Only applied for numeric predictors. If TRUE, it is recommended to set argument ndraws 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 TRUE. The surface type can be controlled via argument stype of the related plotting method.
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.
`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.
`ndraws`	Positive integer indicating how many posterior draws should be used. If `NULL` (the default) all draws are used. Ignored if `draw_ids` is not `NULL`.
`draw_ids`	An integer vector specifying the posterior draws to be used. If `NULL` (the default), all draws are used.
`nsamples`	Deprecated alias of `ndraws`.
`subset`	Deprecated alias of `draw_ids`.
`probs`	(Deprecated) The quantiles to be used in the computation of uncertainty intervals. Please use argument `prob` instead.
`...`	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 `brms_conditional_effects`. See `conditional_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(conditional_smooths(fit), rug = TRUE, ask = FALSE)
# show only the smooth term s(x2)
plot(conditional_smooths(fit, smooths = "s(x2)"), ask = FALSE)

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

## End(Not run)
```

constant *Constant priors in* **brms**

Description

Function used to set up constant priors in **brms**. The function does not evaluate its arguments – it exists purely to help set up the model.

Usage

```
constant(const, broadcast = TRUE)
```

Arguments

const	Numeric value, vector, matrix of values to which the parameters should be fixed to. Can also be a valid Stan variable in the model.
broadcast	Should const be automatically broadcasted to the correct size of the parameter? Defaults to TRUE. If you supply vectors or matrices in const or vector/matrix valued Stan variables, you need to set broadcast to TRUE (see Examples).

Value

A named list with elements const and broadcast.

See Also

set_prior

Examples

```
stancode(count ~ Base + Age, data = epilepsy,
         prior = prior(constant(1), class = "b"))

# will fail parsing because brms will try to broadcast a vector into a vector
stancode(count ~ Base + Age, data = epilepsy,
         prior = prior(constant(alpha), class = "b"),
         stanvars = stanvar(c(1, 0), name = "alpha"))

stancode(count ~ Base + Age, data = epilepsy,
         prior = prior(constant(alpha, broadcast = FALSE), class = "b"),
         stanvars = stanvar(c(1, 0), name = "alpha"))
```

control_params *Extract Control Parameters of the NUTS Sampler*

Description

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

Usage

```
control_params(x, ...)

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

Arguments

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

Value

A named `list` with control parameter values.

cor_ar *(Deprecated) AR(p) correlation structure*

Description

This function is deprecated. Please see `ar` for the new syntax. 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 `~ t`, or `~ 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 `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.

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. This is currently only possible for stationary ARMA effects of order 1. If the model family does not have natural residuals, latent residuals are added automatically. If FALSE (the default) a regression formulation is used that is considerably faster and allows for ARMA effects of order higher than 1 but is only available for gaussian models and some of its generalizations.

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.

See Also

cor_arma

Examples

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

---

cor_arma *(Deprecated) ARMA(p,q) correlation structure*

Description

This function is deprecated. Please see arma for the new syntax. This functions 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 ~ t, or ~ 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 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.
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	No longer supported.
cov	A flag indicating whether ARMA effects should be estimated by means of residual covariance matrices. This is currently only possible for stationary ARMA effects of order 1. If the model family does not have natural residuals, latent residuals are added automatically. If FALSE (the default) a regression formulation is used that is considerably faster and allows for ARMA effects of order higher than 1 but is only available for gaussian models and some of its generalizations.

Value

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

See Also

cor_ar, cor_ma

Examples

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

---

cor_brms *(Deprecated) 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_car Spatial conditional autoregressive (CAR) structure

cor_sar Spatial simultaneous autoregressive (SAR) structure

cor_fixed fixed user-defined covariance structure

See Also

cor_arma, cor_ar, cor_ma, cor_car, cor_sar, cor_fixed

---

cor_car *(Deprecated) Spatial conditional autoregressive (CAR) structures*

Description

These function are deprecated. Please see car for the new syntax. These functions are constructors for the cor_car class implementing spatial conditional autoregressive structures.

Usage

```
cor_car(W, formula = ~1, type = "escar")

cor_icar(W, formula = ~1)
```

Arguments

W	Adjacency matrix of locations. All non-zero entries are treated as if the two locations are adjacent. If formula contains a grouping factor, the row names of W have to match the levels of the grouping factor.
formula	An optional one-sided formula of the form ~ 1 | g, where g 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.
type	Type of the CAR structure. Currently implemented are "escar" (exact sparse CAR), "esicar" (exact sparse intrinsic CAR), "icar" (intrinsic CAR), and "bym2". 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 and bym2 type is implemented based on the case study of Mitzi Morris (https://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_cosy` *(Deprecated) Compound Symmetry (COSY) Correlation Structure*

Description

This function is deprecated. Please see `cosy` for the new syntax. This functions is a constructor for the `cor_cosy` class, representing a compound symmetry structure corresponding to uniform correlation.

Usage

```
cor_cosy(formula = ~1)
```

Arguments

formula	A one sided formula of the form ~ t, or ~ 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 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.

Value

An object of class cor_cosy, representing a compound symmetry correlation structure.

Examples

```
cor_cosy(~ visit | patient)
```

---

cor_fixed *(Deprecated) Fixed user-defined covariance matrices*

Description

This function is deprecated. Please see fcor for the new syntax. 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 *(Deprecated) MA(q) correlation structure*

Description

This function is deprecated. Please see ma for the new syntax. 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 ~ t, or ~ 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 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.
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. This is currently only possible for stationary ARMA effects of order 1. If the model family does not have natural residuals, latent residuals are added automatically. If FALSE (the default) a regression formulation is used that is considerably faster and allows for ARMA effects of order higher than 1 but is only available for gaussian models and some of its generalizations.

Value

An object of class cor_arma containing solely moving average terms.

See Also

cor_arma

Examples

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

cor_sar *(Deprecated) Spatial simultaneous autoregressive (SAR) structures*

Description

Thse functions are deprecated. Please see sar for the new syntax. 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)
```

`cosy` *Set up COSY correlation structures*

Description

Set up a compounds symmetry (COSY) term in **brms**. The function does not evaluate its arguments – it exists purely to help set up a model with COSY terms.

Usage

```
cosy(time = NA, gr = NA)
```

Arguments

`time`	An optional time variable specifying the time ordering of the observations. By default, the existing order of the observations in the data is used.
`gr`	An optional grouping variable. If specified, the correlation structure is assumed to apply only to observations within the same grouping level.

Value

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

See Also

`autocor-terms`

Examples

```
## Not run:
data("lh")
lh <- as.data.frame(lh)
fit <- brm(x ~ cosy(), data = lh)
summary(fit)

## End(Not run)
```

`create_priorsense_data.brmsfit` *Prior sensitivity: Create priorsense data*

Description

The `create_priorsense_data.brmsfit` method can be used to create the data structure needed by the **priorsense** package for performing power-scaling sensitivity analysis. This method is called automatically when performing powerscaling via `powerscale` or other related functions, so you will rarely need to call it manually yourself.

Usage

```
create_priorsense_data.brmsfit(x, ...)
```

Arguments

`x`	A `brmsfit` object.
`...`	Additional arguments passed to `log_lik`, for example `newdata`.

Value

A `priorsense_data` object to be used in conjunction with the **priorsense** package.

Examples

```
## Not run:
# fit a model with non-uniform priors
fit <- brm(rating ~ treat + period + carry,
           data = inhaler, family = sratio(),
           prior = set_prior("normal(0, 0.5)"))
summary(fit)

# The following code requires the 'priorsense' package to be installed:
library(priorsense)

# perform power-scaling of the prior
powerscale(fit, alpha = 1.5, component = "prior")

# perform power-scaling sensitivity checks
powerscale_sensitivity(fit)

# create power-scaling sensitivity plots (for one variable)
powerscale_plot_dens(fit, variable = "b_treat")

## End(Not run)
```

cs *Category Specific Predictors in* **brms** *Models*

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)
```

custom_family *Custom Families in* **brms** *Models*

Description

Define custom families (i.e. response distribution) 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,
  loop = TRUE,
  specials = NULL,
  threshold = "flexible",
  log_lik = NULL,
  posterior_predict = NULL,
  posterior_epred = NULL,
  predict = NULL,
  fitted = 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"). This controls if the corresponding density function will be named with <name>_lpdf or <name>_lpmf.
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 that are part of the likelihood function without being distributional parameters. That is, vars can be used to pass data to the likelihood. Such arguments will be added to the list of function arguments at the end, after the distributional parameters. See stanvar for details about adding self-defined data to the generated **Stan** model. Addition arguments vreal and vint may be used for this purpose as well (see Examples below). See also brmsformula and addition-terms for more details.
loop	Logical; Should the likelihood be evaluated via a loop (TRUE; the default) over observations in Stan? If FALSE, the Stan code will be written in a vectorized manner over observations if possible.
specials	A character vector of special options to enable for this custom family. Currently for internal use only.

threshold — Optional threshold type for custom ordinal families. Ignored for non-ordinal families.

log_lik — Optional function to compute log-likelihood values of the model in R. This is only relevant if one wants to ensure compatibility with method log_lik.

posterior_predict — Optional function to compute posterior prediction of the model in R. This is only relevant if one wants to ensure compatibility with method posterior_predict.

posterior_epred — Optional function to compute expected values of the posterior predictive distribution of the model in R. This is only relevant if one wants to ensure compatibility with method posterior_epred.

predict — Deprecated alias of 'posterior_predict'.

fitted — Deprecated alias of 'posterior_epred'.

env — An environment in which certain post-processing functions related to the custom family can be found, if there were not directly passed to custom_family. This is only relevant if one wants to ensure compatibility with the methods log_lik, posterior_predict, or posterior_epred. By default, env is the environment from which custom_family is called.

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 from class brmsfamily.

See Also

brmsfamily, brmsformula, 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 = "vint1[n]"
)

# define the corresponding Stan density function
stan_density <- "
  real beta_binomial2_lpmf(int y, real mu, real phi, int N) {
    return beta_binomial_lpmf(y | N, mu * phi, (1 - mu) * phi);
  }
"
stanvars <- stanvar(scode = stan_density, block = "functions")

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


# define a *vectorized* custom family (no loop over observations)
# notice also that 'vint' no longer has an observation index
beta_binomial2_vec <- custom_family(
  "beta_binomial2", dpars = c("mu", "phi"),
  links = c("logit", "log"), lb = c(NA, 0),
  type = "int", vars = "vint1", loop = FALSE
)

# define the corresponding Stan density function
stan_density_vec <- "
  real beta_binomial2_lpmf(array[] int y, vector mu, real phi, array[] int N) {
    return beta_binomial_lpmf(y | N, mu * phi, (1 - mu) * phi);
  }
"
stanvars_vec <- stanvar(scode = stan_density_vec, block = "functions")

# fit the model
fit_vec <- brm(y | vint(ntrials) ~ z, data = dat,
           family = beta_binomial2_vec,
           stanvars = stanvars_vec)
summary(fit_vec)

## End(Not run)
```

`default_prior` *Default priors for Bayesian models*

Description

default_prior is a generic function that can be used to get default priors for Bayesian models. Its original use is within the **brms** package, but new methods for use with objects from other packages can be registered to the same generic.

Usage

```
default_prior(object, ...)

get_prior(formula, ...)
```

Arguments

object	An object whose class will determine which method will be used. A symbolic description of the model to be fitted.
...	Further arguments passed to the specific method.
formula	Synonym of object for use in get_prior.

Details

See default_prior.default for the default method applied for **brms** models. You can view the available methods by typing methods(default_prior).

Value

Usually, a brmsprior object. See default_prior.default for more details.

See Also

set_prior, default_prior.default

Examples

```
## get all parameters and parameters classes to define priors on
(prior <- default_prior(count ~ zAge + zBase * Trt + (1|patient) + (1|obs),
                        data = epilepsy, family = poisson()))
```

---

default_prior.default *Default Priors for* **brms** *Models*

Description

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

Usage

```
## Default S3 method:
default_prior(
  object,
  data,
  family = gaussian(),
  autocor = NULL,
  data2 = NULL,
  knots = NULL,
  drop_unused_levels = TRUE,
  sparse = NULL,
  ...
)
```

Arguments

object	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 data.frame (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 link 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	(Deprecated) 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 NULL, corresponding to no correlations. In multivariate models, autocor might also be a list of autocorrelation structures. It is now recommend to specify autocorrelation terms directly within formula. See brmsformula for more details.
data2	A named list of objects containing data, which cannot be passed via argument data. Required for some objects used in autocorrelation structures to specify dependency structures as well as for within-group covariance matrices.
knots	Optional list containing user specified knot values to be used for basis construction of smoothing terms. See gamm for more details.
drop_unused_levels	Should unused factors levels in the data be dropped? Defaults to TRUE.
sparse	(Deprecated) 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. It is now recommended to use the sparse argument of brmsformula and related functions.
...	Other arguments for internal usage only.

Value

A brmsprior object. That is, a data.frame with specific columns including 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

default_prior, set_prior

Examples

```
# get all parameters and parameters classes to define priors on
(prior <- default_prior(count ~ zAge + zBase * Trt + (1|patient) + (1|obs),
                        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
prior$prior[5] <- "student_t(10, 0, 5)"

# verify that the priors indeed found their way into Stan's model code
stancode(count ~ zAge + zBase * Trt + (1|patient) + (1|obs),
         data = epilepsy, family = poisson(),
         prior = prior)
```

---

density_ratio *Compute Density Ratios*

Description

Compute the ratio of two densities at given points based on draws of the corresponding distributions.

Usage

```
density_ratio(x, y = NULL, point = 0, n = 4096, ...)
```

Arguments

x	Vector of draws from the first distribution, usually the posterior distribution of the quantity of interest.
y	Optional vector of draws from the second distribution, usually the prior distribution of the quantity of interest. If NULL (the default), only the density of x will be evaluated.
point	Numeric values at which to evaluate and compare the densities. Defaults to 0.
n	Single numeric value. Influences the accuracy of the density estimation. See density for details.
...	Further arguments passed to density.

Details

In order to achieve sufficient accuracy in the density estimation, more draws than usual are required. That is you may need an effective sample size of 10,000 or more to reliably estimate the densities.

Value

A vector of length equal to `length(point)`. If `y` is provided, the density ratio of `x` against `y` is returned. Else, only the density of `x` is returned.

Examples

```
x <- rnorm(10000)
y <- rnorm(10000, mean = 1)
density_ratio(x, y, point = c(0, 1))
```

---

diagnostic-quantities *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(x, pars = NULL, ...)

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

Arguments

`object`, `x`	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)
```

Dirichlet *The Dirichlet Distribution*

Description

Density function and random number generation for the dirichlet distribution with shape parameter vector alpha.

Usage

```
ddirichlet(x, alpha, log = FALSE)

rdirichlet(n, alpha)
```

Arguments

x	Matrix of quantiles. Each row corresponds to one probability vector.
alpha	Matrix of positive shape parameters. Each row corresponds to one probability vector.
log	Logical; If TRUE, values are returned on the log scale.
n	Number of draws to sample from the distribution.

Details

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

`draws-brms` *Transform* `brmsfit` *to* `draws` *objects*

Description

Transform a `brmsfit` object to a format supported by the **posterior** package.

Usage

```
## S3 method for class 'brmsfit'
as_draws(x, variable = NULL, regex = FALSE, inc_warmup = FALSE, ...)

## S3 method for class 'brmsfit'
as_draws_matrix(x, variable = NULL, regex = FALSE, inc_warmup = FALSE, ...)

## S3 method for class 'brmsfit'
as_draws_array(x, variable = NULL, regex = FALSE, inc_warmup = FALSE, ...)

## S3 method for class 'brmsfit'
as_draws_df(x, variable = NULL, regex = FALSE, inc_warmup = FALSE, ...)

## S3 method for class 'brmsfit'
as_draws_list(x, variable = NULL, regex = FALSE, inc_warmup = FALSE, ...)

## S3 method for class 'brmsfit'
as_draws_rvars(x, variable = NULL, regex = FALSE, inc_warmup = FALSE, ...)
```

Arguments

`x`	A `brmsfit` object or another R object for which the methods are defined.
`variable`	A character vector providing the variables to extract. By default, all variables are extracted.
`regex`	Logical; Should variable should be treated as a (vector of) regular expressions? Any variable in `x` matching at least one of the regular expressions will be selected. Defaults to `FALSE`.
`inc_warmup`	Should warmup draws be included? Defaults to `FALSE`.
`...`	Arguments passed to individual methods (if applicable).

Details

To subset iterations, chains, or draws, use the `subset_draws` method after transforming the `brmsfit` to a `draws` object.

See Also

draws subset_draws

Examples

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

# extract posterior draws in an array format
(draws_fit <- as_draws_array(fit))
posterior::summarize_draws(draws_fit)

# extract only certain variables
as_draws_array(fit, variable = "r_patient")
as_draws_array(fit, variable = "^b_", regex = TRUE)

# extract posterior draws in a random variables format
as_draws_rvars(fit)

## End(Not run)
```

---

draws-index-brms *Index* brmsfit *objects*

Description

Index brmsfit objects

Usage

```
## S3 method for class 'brmsfit'
variables(x, ...)

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

## S3 method for class 'brmsfit'
niterations(x)

## S3 method for class 'brmsfit'
nchains(x)

## S3 method for class 'brmsfit'
ndraws(x)
```

Arguments

x	A brmsfit object or another R object for which the methods are defined.
...	Arguments passed to individual methods (if applicable).

---

emmeans-brms-helpers *Support Functions for* **emmeans**

Description

Functions required for compatibility of **brms** with **emmeans**. Users are not required to call these functions themselves. Instead, they will be called automatically by the emmeans function of the **emmeans** package.

Usage

```
recover_data.brmsfit(
  object,
  data,
  resp = NULL,
  dpar = NULL,
  nlpar = NULL,
  re_formula = NA,
  epred = FALSE,
  ...
)

emm_basis.brmsfit(
  object,
  trms,
  xlev,
  grid,
  vcov.,
  resp = NULL,
  dpar = NULL,
  nlpar = NULL,
  re_formula = NA,
  epred = FALSE,
  ...
)
```

Arguments

object	An object of class brmsfit.
data, trms, xlev, grid, vcov.	Arguments required by **emmeans**.

resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
dpar	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
nlpar	Optional name of a predicted non-linear parameter. If specified, expected predictions of this parameters are returned.
re_formula	Optional formula containing group-level effects to be considered in the prediction. If NULL, include all group-level effects; if NA (default), include no group-level effects.
epred	Logical. If TRUE compute predictions of the posterior predictive distribution's mean (see posterior_epred.brmsfit) while ignoring arguments dpar and nlpar. Defaults to FALSE. If you have specified a response transformation within the formula, you need to set epred to TRUE for **emmeans** to detect this transformation.
...	Additional arguments passed to **emmeans**.

Details

In order to ensure compatibility of most **brms** models with **emmeans**, predictions are not generated 'manually' via a design matrix and coefficient vector, but rather via posterior_linpred.brmsfit. This appears to generally work well, but note that it produces an '.@linfct' slot that contains the computed predictions as columns instead of the coefficients.

Examples

```
## Not run:
fit1 <- brm(time | cens(censored) ~ age * sex + disease + (1|patient),
            data = kidney, family = lognormal())
summary(fit1)

# summarize via 'emmeans'
library(emmeans)
rg <- ref_grid(fit1)
em <- emmeans(rg, "disease")
summary(em, point.est = mean)

# obtain estimates for the posterior predictive distribution's mean
epred <- emmeans(fit1, "disease", epred = TRUE)
summary(epred, point.est = mean)


# model with transformed response variable
fit2 <- brm(log(mpg) ~ factor(cyl), data = mtcars)
summary(fit2)

# results will be on the log scale by default
emmeans(fit2, ~ cyl)
# log transform is detected and can be adjusted automatically
emmeans(fit2, ~ cyl, epred = TRUE, type = "response")
```

```
## 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 data frame 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

patient The patient number

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

count The seizure count between two visits

obs The observation number, that is a unique identifier for each observation

zAge Standardized `Age`

zBase Standardized `Base`

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 ~ zAge + zBase * Trt,
            data = epilepsy, family = poisson())
summary(fit1)
plot(fit1)
```

```
## poisson regression with varying intercepts of patients
## as well as normal priors for overall effects parameters.
fit2 <- brm(count ~ zAge + zBase * Trt + (1|patient),
            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 `mu` and standard deviation `sigma` of the gaussian component, as well as scale `beta` 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

`x`, `q`	Vector of quantiles.
`mu`	Vector of means of the combined distribution.
`sigma`	Vector of standard deviations of the gaussian component.
`beta`	Vector of scales of the exponential component.
`log`	Logical; If `TRUE`, values are returned on the log scale.
`lower.tail`	Logical; If `TRUE` (default), return P(X <= x). Else, return P(X > x) .
`log.p`	Logical; If `TRUE`, values are returned on the log scale.
`n`	Number of draws to sample 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 object of class `brmsfit`.
`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`.

expp1 *Exponential function plus one.*

Description

Computes `exp(x) + 1`.

Usage

```
expp1(x)
```

Arguments

`x`	A numeric or complex vector.

family.brmsfit *Extract Model Family Objects*

Description

Extract Model Family Objects

Usage

```
## S3 method for class 'brmsfit'
family(object, resp = NULL, ...)
```

Arguments

object	An object of class brmsfit.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
...	Currently unused.

Value

A brmsfamily object or a list of such objects for multivariate models.

fcor *Fixed residual correlation (FCOR) structures*

Description

Set up a fixed residual correlation (FCOR) term in **brms**. The function does not evaluate its arguments – it exists purely to help set up a model with FCOR terms.

Usage

```
fcor(M)
```

Arguments

M	Known correlation/covariance matrix of the response variable. If a vector is passed, it will be used as diagonal entries (variances) and correlations/covariances will be set to zero. The actual covariance matrix used in the likelihood is obtained by multiplying M by the square of the residual standard deviation parameter sigma estimated as part of the model.

Value

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

See Also

autocor-terms

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 + fcor(V), data = dat, data2 = list(V = V))

## End(Not run)
```

---

fitted.brmsfit *Expected Values of the Posterior Predictive Distribution*

Description

This method is an alias of posterior_epred.brmsfit with additional arguments for obtaining summaries of the computed draws.

Usage

```
## S3 method for class 'brmsfit'
fitted(
  object,
  newdata = NULL,
  re_formula = NULL,
  scale = c("response", "linear"),
  resp = NULL,
  dpar = NULL,
  nlpar = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  sort = FALSE,
  summary = TRUE,
  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. NA values within factors (excluding grouping variables) 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. NA values within grouping variables are treated as a new level.
re_formula	formula containing group-level effects to be considered in the prediction. If NULL (default), include all group-level effects; if NA or ~0, include no group-level effects.
scale	Either "response" or "linear". If "response", results are returned on the scale of the response variable. If "linear", results are returned on the scale of the linear predictor term, that is without applying the inverse link function or other transformations.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
dpar	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
nlpar	Optional name of a predicted non-linear parameter. If specified, expected predictions of this parameters are returned.
ndraws	Positive integer indicating how many posterior draws should be used. If NULL (the default) all draws are used. Ignored if draw_ids is not NULL.
draw_ids	An integer vector specifying the posterior draws to be used. If NULL (the default), all draws are used.
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).
summary	Should summary statistics 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.
...	Further arguments passed to prepare_predictions that control several aspects of data validation and prediction.

Value

An array of predicted *mean* response values. If summary = FALSE the output resembles those of posterior_epred.brmsfit.

If summary = TRUE the output depends on the family: For categorical and ordinal families, the output is an N x E x C array, where N is the number of observations, E is the number of summary statistics,

and C is the number of categories. For all other families, the output is an N x E matrix. The number of summary statistics E is equal to `2 + length(probs)`: The `Estimate` column contains point estimates (either mean or median depending on argument `robust`), while the `Est.Error` column contains uncertainty estimates (either standard deviation or median absolute deviation depending on argument `robust`). The remaining columns starting with `Q` contain quantile estimates as specified via argument `probs`.

In multivariate models, an additional dimension is added to the output which indexes along the different response variables.

See Also

`posterior_epred.brmsfit`

Examples

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

## compute expected predictions
fitted_values <- fitted(fit)
head(fitted_values)

## plot expected predictions 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),
  pars = NULL,
  ...
)
```

Arguments

object	An object of class brmsfit.
summary	Should summary statistics 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.
pars	Optional names of coefficients to extract. By default, all coefficients are extracted.
...	Currently ignored.

Value

If summary is TRUE, a matrix returned by posterior_summary for the population-level effects. If summary is FALSE, a matrix with one row per posterior draw and one column per population-level effect.

Examples

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

## 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

x, q	Vector of quantiles.
loc	Vector of locations.
scale	Vector of scales.
shape	Vector of shapes.
log	Logical; If TRUE, values are returned on the log scale.
lower.tail	Logical; If TRUE (default), return P(X <= x). Else, return P(X > x) .
log.p	Logical; If TRUE, values are returned on the log scale.
p	Vector of probabilities.
n	Number of draws to sample 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 `mu`, scale `sigma` and shape `xi`.

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
)

qgen_extreme_value(
  p,
  mu = 0,
  sigma = 1,
```

```
  xi = 0,
  lower.tail = TRUE,
  log.p = FALSE
)

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

Arguments

x, q	Vector of quantiles.
mu	Vector of locations.
sigma	Vector of scales.
xi	Vector of shapes.
log	Logical; If TRUE, values are returned on the log scale.
lower.tail	Logical; If TRUE (default), return P(X <= x). Else, return P(X > x) .
log.p	Logical; If TRUE, values are returned on the log scale.
p	Vector of probabilities.
n	Number of draws to sample from the distribution.

Details

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

---

get_dpar *Draws of a Distributional Parameter*

Description

Get draws of a distributional parameter from a `brmsprep` or `mvbrmsprep` object. This function is primarily useful when developing custom families or packages depending on **brms**. This function lets callers easily handle both the case when the distributional parameter is predicted directly, via a (non-)linear predictor or fixed to a constant. See the vignette `vignette("brms_customfamilies")` for an example use case.

Usage

```
get_dpar(prep, dpar, i = NULL, inv_link = NULL)
```

Arguments

prep	A 'brmsprep' or 'mvbrmsprep' object created by prepare_predictions.
dpar	Name of the distributional parameter.
i	The observation numbers for which predictions shall be extracted. If NULL (the default), all observation will be extracted. Ignored if dpar is not predicted.
inv_link	Should the inverse link function be applied? If NULL (the default), the value is chosen internally. In particular, inv_link is TRUE by default for custom families.

Value

If the parameter is predicted and `i` is `NULL` or `length(i) > 1`, an `S` x `N` matrix. If the parameter it not predicted or `length(i) == 1`, a vector of length `S`. Here `S` is the number of draws and `N` is the number of observations or length of `i` if specified.

Examples

```
## Not run:
posterior_predict_my_dist <- function(i, prep, ...) {
  mu <- brms::get_dpar(prep, "mu", i = i)
  mypar <- brms::get_dpar(prep, "mypar", i = i)
  my_rng(mu, mypar)
}

## End(Not run)
```

`get_refmodel.brmsfit` *Projection Predictive Variable Selection: Get Reference Model*

Description

The `get_refmodel.brmsfit` method can be used to create the reference model structure which is needed by the **projpred** package for performing a projection predictive variable selection. This method is called automatically when performing variable selection via `varsel` or `cv_varsel`, so you will rarely need to call it manually yourself.

Usage

```
get_refmodel.brmsfit(
  object,
  newdata = NULL,
  resp = NULL,
  cvfun = NULL,
  dis = NULL,
  latent = FALSE,
  brms_seed = NULL,
  ...
)
```

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. `NA` values within factors (excluding grouping variables) 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. `NA` values within grouping variables are treated as a new level.

resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
cvfun	Optional cross-validation function (see get_refmodel for details). If NULL (the default), cvfun is defined internally based on kfold.brmsfit.
dis	Passed to argument dis of init_refmodel, but leave this at NULL unless **projpred** complains about it.
latent	See argument latent of extend_family. Setting this to TRUE requires a **projpred** version >= 2.4.0.
brms_seed	A seed used to infer seeds for kfold.brmsfit and for sampling group-level effects for new levels (in multilevel models). If NULL, then set.seed is not called at all. If not NULL, then the pseudorandom number generator (PRNG) state is reset (to the state before calling this function) upon exiting this function.
...	Further arguments passed to init_refmodel.

Details

The extract_model_data function used internally by get_refmodel.brmsfit ignores arguments wrhs and orhs (a warning is thrown if these are non-NULL). For example, arguments weightsnew and offsetnew of proj_linpred, proj_predict, and predict.refmodel are passed to wrhs and orhs, respectively.

Value

A refmodel object to be used in conjunction with the **projpred** package.

Examples

```
## Not run:
# fit a simple model
fit <- brm(count ~ zAge + zBase * Trt,
           data = epilepsy, family = poisson())
summary(fit)

# The following code requires the 'projpred' package to be installed:
library(projpred)

# perform variable selection without cross-validation
vs <- varsel(fit)
summary(vs)
plot(vs)

# perform variable selection with cross-validation
cv_vs <- cv_varsel(fit)
summary(cv_vs)
plot(cv_vs)

## End(Not run)
```

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

Description

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

Usage

```
gp(
  ...,
  by = NA,
  k = NA,
  cov = "exp_quad",
  iso = TRUE,
  gr = TRUE,
  cmc = TRUE,
  scale = TRUE,
  c = 5/4
)
```

Arguments

`...`	One or more predictors for the GP.
`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 GP. In the factor variable case, a separate GP is fitted for each factor level.
`k`	Optional number of basis functions for computing Hilbert-space approximate GPs. If `NA` (the default), exact GPs are computed.
`cov`	Name of the covariance kernel. Currently supported are `"exp_quad"` (exponentiated-quadratic kernel; default), `"matern32"` (Matern 3/2 kernel), `"matern52"` (Matern 5/2 kernel), and `"exponential"` (exponential kernel; alias: `"matern12"`).
`iso`	A flag to indicate whether an isotropic (`TRUE`; the default) or a non-isotropic GP should be used. In the former case, the same amount of smoothing is applied to all predictors. In the latter case, predictors may have different smoothing. Ignored if only a single predictor is supplied.
`gr`	Logical; Indicates if auto-grouping should be used (defaults to `TRUE`). If enabled, observations sharing the same predictor values will be represented by the same latent variable in the GP. This will improve sampling efficiency drastically if the number of unique predictor combinations is small relative to the number of observations.
`cmc`	Logical; Only relevant if `by` is a factor. If `TRUE` (the default), cell-mean coding is used for the `by`-factor, that is one GP per level is estimated. If `FALSE`, contrast GPs are estimated according to the contrasts set for the `by`-factor.

`scale` Logical; If `TRUE` (the default), predictors are scaled so that the maximum Euclidean distance between two points is 1. This often improves sampling speed and convergence. Scaling also affects the estimated length-scale parameters in that they resemble those of scaled predictors (not of the original predictors) if `scale` is `TRUE`.

`c` Numeric value only used in approximate GPs. Defines the multiplicative constant of the predictors' range over which predictions should be computed. A good default could be `c = 5/4` but we are still working on providing better recommendations.

Details

A GP is a stochastic process, which describes the relation between one or more predictors $x = (x_1, ..., x_d)$ and a response $f(x)$, where d is the number of predictors. A GP is the generalization of the multivariate normal distribution to an infinite number of dimensions. Thus, it can be interpreted as a prior over functions. The values of $f()$ at any finite set of locations are jointly multivariate normal, with a covariance matrix defined by the covariance kernel $k_p(x_i, x_j)$, where p is the vector of parameters of the GP:

$$(f(x_1), \dots f(x_n) \sim MVN(0, (k_p(x_i, x_j))_{i,j=1}^n).$$

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.

For mathematical details on the supported kernels, please see the Stan manual: https://mc-stan.org/docs/functions-reference/matrix_operations.html under "Gaussian Process Covariance Functions".

There are several parameters estimated for GPs, the most important of which are as follows:

Parameter	Notation	Support	Meaning
`lscale`	ℓ	$\mathbb{R}^+$	length-scale of the GP's covariance kernel
`sdgp`	σ	$\mathbb{R}^+$	marginal standard deviation of the GP's covariance kernel
`zgp`	z	$\mathbb{R}$	latent variable values of the training data

Assuming an exponentiated quadratic covariance structure, the parameters can be broadly interpreted as follows (see also https://mc-stan.org/docs/stan-users-guide/gaussian-processes.html#gaussian-process-regression). Note that in the above documentation the parameter σ is denoted as α instead, and the parameter ℓ as ρ. The length-scale ℓ controls the frequency of the functions represented by the GP prior i.e., values of $\ell \gg 0$ lead to lower-frequency functions, while values of $\ell \approx 0$ lead to higher-frequency functions. In slightly simpler terms, ℓ sets the distance over which observations in the input space are strongly correlated. The marginal standard deviation σ controls the magnitude of the range of the function represented by the GP i.e., it represents how much the values of the function tend to deviate from the mean level. Lastly, the parameter z represents latent variable values per observation (or unique input point).

Value

An object of class `'gp_term'`, 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 GP model
fit1 <- brm(y ~ gp(x2), dat, chains = 2)
summary(fit1)
me1 <- conditional_effects(fit1, ndraws = 200, spaghetti = TRUE)
plot(me1, ask = FALSE, points = TRUE)

# fit a more complicated GP model and use an approximate GP for x2
fit2 <- brm(y ~ gp(x0) + x1 + gp(x2, k = 10) + x3, dat, chains = 2)
summary(fit2)
me2 <- conditional_effects(fit2, ndraws = 200, spaghetti = TRUE)
plot(me2, ask = FALSE, points = TRUE)

# fit a multivariate GP model with Matern 3/2 kernel
fit3 <- brm(y ~ gp(x1, x2, cov = "matern32"), dat, chains = 2)
summary(fit3)
me3 <- conditional_effects(fit3, ndraws = 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(conditional_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,
  cor = TRUE,
  id = NA,
  pw = NULL,
  cov = NULL,
  dist = "gaussian"
)
```

Arguments

...	One or more terms containing grouping factors.
by	An optional factor variable, specifying sub-populations of the groups. For each level of the by variable, a separate variance-covariance matrix will be fitted. Levels of the grouping factor must be nested in levels of the by variable.
cor	Logical. If TRUE (the default), group-level terms will be modelled as correlated.
id	Optional character string. All group-level terms across the model with the same id will be modeled as correlated (if cor is TRUE). See brmsformula for more details.
pw	Optional numeric variable specifying prior weights. They weight the contribution of each group to the log-prior of the group-level coefficients. Should have one distinct value for each level of the grouping variable.
cov	An optional matrix which is proportional to the within-group covariance matrix of the group-level effects. 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 vignette("brms_phylogenetics") for more details. By default, levels of the same grouping factor are modeled as independent of each other.
dist	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)

# include a group-level weight variable
epilepsy[['patient_samp_wgt']] <- c(1, rep(c(0.9, 1.1), each = 29))
fit4 <- brm(count ~ Trt + (1|gr(patient, pw = patient_samp_wgt)),
            data = epilepsy)
summary(fit4)

## End(Not run)
```

`horseshoe` *Regularized horseshoe priors in* **brms**

Description

Function used to set up regularized horseshoe priors and related hierarchical shrinkage priors 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,
  main = FALSE
)
```

Arguments

`df`	Degrees of freedom of student-t prior of the local shrinkage parameters. Defaults to 1.
`scale_global`	Scale of the student-t prior of the global shrinkage parameter. Defaults to 1. In linear models, `scale_global` will internally be multiplied by the residual standard deviation parameter `sigma`.
`df_global`	Degrees of freedom of student-t prior of the global shrinkage parameter. Defaults to 1. If `df_global` is greater 1, the shape of the prior will no longer resemble a horseshoe and it may be more appropriately called an hierarchical shrinkage prior in this case.

scale_slab	Scale of the Student-t slab. Defaults to 2. The original unregularized horseshoe prior is obtained by setting scale_slab to infinite, which we can approximate in practice by setting it to a very large real value.
df_slab	Degrees of freedom of the student-t slab. Defaults to 4.
par_ratio	Ratio of the expected number of non-zero coefficients to the expected number of zero coefficients. If specified, scale_global is ignored and internally computed as 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.
main	Logical (defaults to FALSE); only relevant if the horseshoe prior spans multiple parameter classes. In this case, only arguments given in the single instance where main is TRUE will be used. Arguments given in other instances of the prior will be ignored. See the Examples section below.

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 one the same scale. Generally, models with horseshoe priors a 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.

The prior does not account for scale differences of the terms it is applied on. Accordingly, please make sure that all these terms have a comparable scale to ensure that shrinkage is applied properly.

Currently, the following classes support the horseshoe prior: b (overall regression coefficients), sds (SDs of smoothing splines), sdgp (SDs of Gaussian processes), ar (autoregressive coefficients), ma

(moving average coefficients), sderr (SD of latent residuals), sdcar (SD of spatial CAR structures), sd (SD of varying coefficients).

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. Artificial Intelligence and Statistics. http://proceedings.mlr.press/v5/carvalho09a

Piironen J. & Vehtari A. (2017). On the Hyperprior Choice for the Global Shrinkage Parameter in the Horseshoe Prior. Artificial Intelligence and Statistics. https://arxiv.org/pdf/1610.05559v1

Piironen, J., and Vehtari, A. (2017). Sparsity information and regularization in the horseshoe and other shrinkage priors. Electronic Journal of Statistics. https://arxiv.org/abs/1707.01694

See Also

set_prior

Examples

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

# specify the horseshoe prior across multiple parameter classes
set_prior(horseshoe(df = 3, par_ratio = 0.1, main = TRUE), class = "b") +
  set_prior(horseshoe(), class = "sd")
```

Hurdle *Hurdle Distributions*

Description

Density and distribution functions for hurdle distributions.

Usage

```
dhurdle_poisson(x, lambda, hu, log = FALSE)

phurdle_poisson(q, lambda, hu, lower.tail = TRUE, log.p = FALSE)

dhurdle_negbinomial(x, mu, shape, hu, log = FALSE)

phurdle_negbinomial(q, mu, shape, hu, lower.tail = TRUE, log.p = FALSE)

dhurdle_gamma(x, shape, scale, hu, log = FALSE)
```

```
phurdle_gamma(q, shape, scale, hu, lower.tail = TRUE, log.p = FALSE)

dhurdle_lognormal(x, mu, sigma, hu, log = FALSE)

phurdle_lognormal(q, mu, sigma, hu, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x	Vector of quantiles.
hu	hurdle probability
log	Logical; If TRUE, values are returned on the log scale.
q	Vector of quantiles.
lower.tail	Logical; If TRUE (default), return P(X <= x). Else, return P(X > x) .
log.p	Logical; If TRUE, values are returned on the log scale.
mu, lambda	location parameter
shape	shape parameter
sigma, scale	scale parameter

Details

The density of a hurdle distribution can be specified as follows. If $x = 0$ set $f(x) = \theta$. Else set $f(x) = (1 - \theta) * g(x)/(1 - G(0))$ where $g(x)$ and $G(x)$ are the density and distribution function of the non-hurdle part, respectively.

---

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,
  robust = FALSE,
  seed = NULL,
  ...
```

```
)

hypothesis(x, ...)

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

Arguments

x	An R object. If it is no brmsfit object, it must be coercible to a data.frame. In the latter case, the variables used in the hypothesis argument need to correspond to column names of x, while the rows are treated as representing posterior draws of the variables.
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 variables)
group	Name of a grouping factor to evaluate only group-level effects parameters related to this grouping factor.
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 and group). 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).
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.
seed	A single numeric value passed to set.seed to make results reproducible. This is currently only relevant for point hypotheses that scope over at least two parameters (see Details).
...	Currently ignored.

Details

Among others, hypothesis computes an evidence ratio (Evid.Ratio) for each hypothesis. For a one-sided hypothesis, this is just the posterior probability (Post.Prob) 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 a two-sided (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 `"yes"`. Otherwise `Evid.Ratio` (and `Post.Prob`) will be `NA`.

Please note that the Savage-Dickey density ratio as implemented here provides only a very basic test of point hypotheses. It is recommended that you use bridge sampling instead (via `bayes_factor` which relies on the **bridgesampling** package). When interpreting Bayes factors for point hypotheses, 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. Additionally, note that point hypotheses that scope over more than one parameter (e.g., when testing equality between two parameters) involve random sampling of the priors over those parameters (to accommodate the the assumption that priors for different parameters are independent of each other). This introduces an element of randomness into such hypothesis tests. Consider repeating the test to assure results are sufficiently stable, and use the argument `seed` for reproducibility. Finally, note that, for technical reasons, we cannot sample from priors of certain parameters classes. Most notably, these include overall intercept parameters (prior class `"Intercept"`) as well as group-level coefficients.

For one-sided hypotheses, the `Evid.Ratio` may sometimes be `0` or `Inf` implying very small or large evidence, respectively, in favor of the tested hypothesis. For one-sided hypotheses pairs, this basically means that all posterior draws are on the same side of the value dividing the two hypotheses. In that sense, instead of `0` or `Inf`, you may rather read it as `Evid.Ratio` smaller `1 / S` or greater `S`, respectively, where `S` denotes the number of posterior draws used in the computations.

The argument `alpha` specifies the size of the credible interval (i.e., Bayesian confidence interval). For instance, if we tested a two-sided hypothesis and set `alpha = 0.05` (5%) an, 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.

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 = "yes",
           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)
str(dat)
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 data frame 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)
```

---

inits.brmsfit *Extract Initial Values Used for Each Chain*

Description

Extract the initial values used by Stan for each chain. This is currently only available if the model was fitted with the `rstan` backend.

Usage

```
## S3 method for class 'brmsfit'
inits(x, ...)

inits(x, ...)
```

Arguments

x	A brmsfit object.
...	Currently ignored.

Value

The initial values (either user-specified or generated randomly) for all chains. This is a list with one component per chain. Each component is a named list containing the initial values for each parameter for the corresponding chain.

InvGaussian *The Inverse Gaussian Distribution*

Description

Density, distribution function, and random generation for the inverse Gaussian distribution with location mu, 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

x, q	Vector of quantiles.
mu	Vector of locations.
shape	Vector of shapes.
log	Logical; If TRUE, values are returned on the log scale.
lower.tail	Logical; If TRUE (default), return P(X <= x). Else, return P(X > x) .
log.p	Logical; If TRUE, values are returned on the log scale.
n	Number of draws to sample from the distribution.

Details

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

inv_logit_scaled *Scaled inverse logit-link*

Description

Computes `inv_logit(x) * (ub - lb) + lb`

Usage

```
inv_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 between `lb` and `ub`.

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

Description

Checks if argument is a `brmsfit` object

Usage

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

Arguments

x	An R object

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

Description

Checks if argument is a brmsfit_multiple object

Usage

```
is.brmsfit_multiple(x)
```

Arguments

x	An R object

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

Description

Checks if argument is a brmsformula object

Usage

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

Arguments

x	An R object

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

Description

Checks if argument is a brmsprior object

Usage

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

Arguments

x	An R object

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

Description

Checks if argument is a brmsterms object

Usage

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

Arguments

x	An R object

See Also

brmsterms

is.cor_brms *Check if argument is a correlation structure*

Description

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

Usage

```
is.cor_brms(x)

is.cor_arma(x)

is.cor_cosy(x)

is.cor_sar(x)

is.cor_car(x)

is.cor_fixed(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

brmsterms

kfold.brmsfit *K-Fold Cross-Validation*

Description

Perform exact K-fold cross-validation by refitting the model K times each leaving out one-Kth of the original data. Folds can be run in parallel using the **future** package.

Usage

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

Arguments

x	A brmsfit object.
...	Further arguments passed to brm and log_lik.
K	The number of subsets of equal (if possible) size into which the data will be 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 or a one-dimensional array (created via as.array) potentially of length one, the corresponding subsets will be used. This argument is primarily useful, if evaluation of all subsets is infeasible for some reason.
folds	Determines how the subsets are being constructed. Possible values are NULL (the default), "stratified", "grouped", or "loo". May also be a vector of length equal to the number of observations in the data. Alters the way group is handled. More information is provided in the 'Details' section.

group	Optional name of a grouping variable or factor in the model. What exactly is done with this variable depends on argument folds. More information is provided in the 'Details' section.
joint	Indicates which observations' log likelihoods shall be considered jointly in the ELPD computation. If "obs" or FALSE (the default), each observation is considered separately. This enables comparability of kfold with loo. If "fold" or TRUE, the joint log likelihoods per fold are used. If "group", the joint log likelihoods per group within folds are used (only available if argument group is specified).
compare	A flag indicating if the information criteria of the models should be compared to each other via loo_compare.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
model_names	If NULL (the default) will use model names derived from deparsing 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.
recompile	Logical, indicating whether the Stan model should be recompiled. This may be necessary if you are running reloo on another machine than the one used to fit the model.
future_args	A list of further arguments passed to future for additional control over parallel execution if activated.

Details

The kfold function performs exact K-fold cross-validation. First the data are partitioned into K folds (i.e. subsets) of equal (or as close to equal as possible) size by default. 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.

The subsets can be constructed in multiple different ways:

- If both folds and group are NULL, the subsets are randomly chosen so that they have equal (or as close to equal as possible) size.
- If folds is NULL but group is specified, the data is split up into subsets, each time omitting all observations of one of the factor levels, while ignoring argument K.
- If folds = "stratified" the subsets are stratified after group using loo::kfold_split_stratified.
- If folds = "grouped" the subsets are split by group using loo::kfold_split_grouped.
- If folds = "loo" exact leave-one-out cross-validation will be performed and K will be ignored. Further, if group is specified, 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.

- If `folds` is a numeric vector, it must contain one element per observation in the data. Each element of the vector is an integer in `1:K` indicating to which of the `K` folds the corresponding observation belongs. There are some convenience functions available in the **loo** package that create integer vectors to use for this purpose (see the Examples section below and also the kfold-helpers page).

When running `kfold` on a `brmsfit` created with the **cmdstanr** backend in a different R session, several recompilations will be triggered because by default, **cmdstanr** writes the model executable to a temporary directory. To avoid that, set option `"cmdstanr_write_stan_file_dir"` to a non-temporary path of your choice before creating the original `brmsfit` (see section 'Examples' below).

By default, this method uses `sample_new_levels = "gaussian"` to sample parameter values for new grouping-factor levels (see also `prepare_predictions`). This default will fail for models with non-Gaussian group-level effects. In this case, we recommend setting `sample_new_levels = "uncertainty"`.

Value

`kfold` returns an object that has a similar structure as the objects returned by the `loo` and `waic` methods and can be used with the same post-processing functions.

Parallelization with multiple CPU cores

brms can make use of multiple CPU cores in parallel to speed up computations in various ways. For efficient use of the available resources it is recommended to only use parallelism to an extend such that the available physical CPUs are not oversubscribed. For example, when you have 8 CPU cores locally available, then you may consider to run 4 chains with 2 threads per chain for best performance if you happen to just run a single model. In case you run a simulation study which requires to run many times a given model, then neither chain nor within-chain parallelization is advisable as the computational resources are already exhausted by the simulation study and any further parallelization beyond the simulation study itself will in fact slow down the overall runtime. Please be aware that for historical reasons the nomenclature of the arguments is possibly confusing. The `cores` argument refers to running different chains in parallel and the within-chain parallelization will allocate for each chain as many threads as requested. The requested threads therefore increase the use of overall CPUs in a multiplicative way.

For more advanced parallelization (including beyond single model fits), **brms** also integrates with the **future** package. Importantly, this enables seamless integration with the **mirai** parallelization framework through the use of the **future.mirai** adapter. With **mirai** local and remote machines can be used in a fully transparent manner to the user. This includes the possibility to use large number of remote machines running in the context of a computer cluster, which are managed with queuing systems. Please refer to the section on distributed computing of `mirai::daemons`.

See Also

`loo`, `reloo`

Examples

```
## Not run:
fit1 <- brm(count ~ zAge + zBase * 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 = 1))

# use joint likelihoods per fold for ELPD evaluation
kfold(fit1, chains = 1, joint = "fold")

# use the future package for parallelization of models
# that is to fit models belonging to different folds in parallel
library(future)
plan(multisession, workers = 4)
kfold(fit1, chains = 1)
plan(sequential)

## to avoid recompilations when running kfold() on a 'cmdstanr'-backend fit
## in a fresh R session, set option 'cmdstanr_write_stan_file_dir' before
## creating the initial 'brmsfit'
## CAUTION: the following code creates some files in the current working
## directory: two 'model_<hash>.stan' files, one 'model_<hash>(.exe)'
## executable, and one 'fit_cmdstanr_<some_number>.rds' file
set.seed(7)
fname <- paste0("fit_cmdstanr_", sample.int(.Machine$integer.max, 1))
options(cmdstanr_write_stan_file_dir = getwd())
fit_cmdstanr <- brm(rate ~ conc + state, data = Puromycin,
                    backend = "cmdstanr", file = fname)

# now restart the R session and run the following (after attaching 'brms')
set.seed(7)
fname <- paste0("fit_cmdstanr_", sample.int(.Machine$integer.max, 1))
fit_cmdstanr <- brm(rate ~ conc + state,
                    data = Puromycin,
                    backend = "cmdstanr",
                    file = fname)
kfold_cmdstanr <- kfold(fit_cmdstanr, K = 2)

## End(Not run)
```

`kfold_predict` *Predictions from K-Fold Cross-Validation*

Description

Compute and evaluate predictions after performing K-fold cross-validation via `kfold`.

Usage

```
kfold_predict(x, method = "posterior_predict", resp = NULL, ...)
```

Arguments

x	Object of class 'kfold' computed by kfold. For kfold_predict to work, the fitted model objects need to have been stored via argument save_fits of kfold.
method	Method used to obtain predictions. Can be set to "posterior_predict" (the default), "posterior_epred", or "posterior_linpred". For more details, see the respective function documentations.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
...	Further arguments passed to prepare_predictions that control several aspects of data validation and prediction.

Value

A list with two slots named 'y' and 'yrep'. Slot y contains the vector of observed responses. Slot yrep contains the matrix of predicted responses, with rows being posterior draws and columns being observations.

See Also

kfold

Examples

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

# perform k-fold cross validation
(kf <- kfold(fit, save_fits = TRUE, chains = 1))

# define a loss function
rmse <- function(y, yrep) {
  yrep_mean <- colMeans(yrep)
  sqrt(mean((yrep_mean - y)^2))
}

# predict responses and evaluate the loss
kfp <- kfold_predict(kf)
rmse(y = kfp$y, yrep = kfp$yrep)

## 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 data frame 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 surivival analysis using the "weibull" family
fit1 <- brm(time | cens(censored) ~ age + sex + disease,
            data = kidney, family = weibull, init = "0")
summary(fit1)
plot(fit1)

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

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

`lasso` *(Defunct) Set up a lasso prior in* **brms**

Description

This functionality is no longer supported as of brms version 2.19.2. Please use the `horseshoe` or `R2D2` shrinkage priors instead.

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`.

Value

An error indicating that the lasso prior is no longer supported.

References

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

See Also

`set_prior`, `horseshoe`, `R2D2`

launch_shinystan.brmsfit *Interface to* **shinystan**

Description

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

Usage

```
launch_shinystan.brmsfit(object, rstudio = getOption("shinystan.rstudio"), ...)
```

Arguments

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

LogisticNormal *The (Multivariate) Logistic Normal Distribution*

Description

Density function and random generation for the (multivariate) logistic normal distribution with latent mean vector `mu` and covariance matrix `Sigma`.

Usage

```
dlogistic_normal(x, mu, Sigma, refcat = 1, log = FALSE, check = FALSE)

rlogistic_normal(n, mu, Sigma, refcat = 1, 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.
`refcat`	A single integer indicating the reference category. Defaults to `1`.
`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 draws to sample from the distribution.

logit_scaled *Scaled logit-link*

Description

Computes `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 *Logarithm with a minus one offset.*

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,
  resp = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  pointwise = FALSE,
  combine = TRUE,
  add_point_estimate = FALSE,
  cores = NULL,
  ...
)
```

Arguments

object	A fitted model 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. NA values within factors (excluding grouping variables) 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. NA values within grouping variables are treated as a new level.
re_formula	formula containing group-level effects to be considered in the prediction. If NULL (default), include all group-level effects; if NA or ~0, include no group-level effects.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
ndraws	Positive integer indicating how many posterior draws should be used. If NULL (the default) all draws are used. Ignored if draw_ids is not NULL.
draw_ids	An integer vector specifying the posterior draws to be used. If NULL (the default), all draws are used.
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 draws 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.
combine	Only relevant in multivariate models. Indicates if the log-likelihoods of the submodels should be combined per observation (i.e. added together; the default) or if the log-likelihoods should be returned separately.
add_point_estimate	For internal use only. Ensures compatibility with the loo_subsample method.
cores	Number of cores (defaults to 1). On non-Windows systems, this argument can be set globally via the mc.cores option.
...	Further arguments passed to prepare_predictions that control several aspects of data validation and prediction.

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.

In multilevel models, it is possible to allow new levels of grouping factors to be used in the predictions. This can be controlled via argument allow_new_levels. New levels can be sampled in multiple ways, which can be controlled via argument sample_new_levels. Both of these arguments are documented in prepare_predictions along with several other useful arguments to control specific aspects of the predictions.

Value

Usually, an S x N matrix containing the pointwise log-likelihood draws, where S is the number of draws 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 draws.

---

loo.brmsfit *Efficient approximate leave-one-out cross-validation (LOO)*

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,
  moment_match = FALSE,
  reloo = FALSE,
  k_threshold = 0.7,
  save_psis = FALSE,
  moment_match_args = list(),
  reloo_args = list(),
  model_names = NULL
)
```

Arguments

x	A brmsfit object.
...	More brmsfit objects or further arguments passed to the underlying post-processing functions. In particular, see prepare_predictions for further supported arguments.
compare	A flag indicating if the information criteria of the models should be compared to each other via loo_compare.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
pointwise	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, pointwise = TRUE is the way to go.

moment_match — Logical; Indicate whether loo_moment_match should be applied on problematic observations. Defaults to FALSE. For most models, moment matching will only work if you have set save_pars = save_pars(all = TRUE) when fitting the model with brm. See loo_moment_match.brmsfit for more details.

reloo — Logical; Indicate whether reloo should be applied on problematic observations. Defaults to FALSE.

k_threshold — The Pareto k threshold for which observations loo_moment_match or reloo is applied if argument moment_match or reloo is TRUE. Defaults to 0.7. See pareto_k_ids for more details.

save_psis — Should the "psis" object created internally be saved in the returned object? For more details see loo.

moment_match_args — Optional named list of additional arguments passed to loo_moment_match.

reloo_args — Optional named list of additional arguments passed to reloo. This can be useful, among others, to control how many chains, iterations, etc. to use for the fitted sub-models.

model_names — If NULL (the default) will use model names derived from deparsing the call. Otherwise will use the passed values as model names.

Details

See loo_compare for details on model comparisons. For brmsfit objects, LOO is an alias of loo. Use method add_criterion to store information criteria in the fitted model object for later usage.

Value

If just one object is provided, an object of class loo. If multiple objects are provided, an object of class loolist.

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)
(loo1 <- loo(fit1))
```

```
# model with an additional varying intercept for subjects
fit2 <- brm(rating ~ treat + period + carry + (1|subject),
            data = inhaler)
(loo2 <- loo(fit2))

# compare both models
loo_compare(loo1, loo2)

## End(Not run)
```

`loo_compare.brmsfit` *Model comparison with the* **loo** *package*

Description

For more details see `loo_compare`.

Usage

```
## S3 method for class 'brmsfit'
loo_compare(x, ..., criterion = c("loo", "waic", "kfold"), model_names = NULL)
```

Arguments

`x`	A `brmsfit` object.
`...`	More `brmsfit` objects.
`criterion`	The name of the criterion to be extracted from `brmsfit` objects.
`model_names`	If `NULL` (the default) will use model names derived from deparsing the call. Otherwise will use the passed values as model names.

Details

All `brmsfit` objects should contain precomputed criterion objects. See `add_criterion` for more help.

Value

An object of class "`compare.loo`".

Examples

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

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

# compare both models
loo_compare(fit1, fit2, criterion = "waic")

## 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 `brmsfit` object.
`...`	More `brmsfit` objects or further arguments passed to the underlying post-processing functions. In particular, see `prepare_predictions` for further supported arguments.
`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_moment_match.brmsfit` *Moment matching for efficient approximate leave-one-out cross-validation*

Description

Moment matching for efficient approximate leave-one-out cross-validation (LOO-CV). See `loo_moment_match` for more details.

Usage

```
## S3 method for class 'brmsfit'
loo_moment_match(
  x,
  loo = NULL,
  k_threshold = 0.7,
  newdata = NULL,
  resp = NULL,
  check = TRUE,
  recompile = FALSE,
  ...
)

## S3 method for class 'loo'
loo_moment_match(x, fit, ...)
```

Arguments

`x`	An R object of class `brmsfit` or `loo` depending on the method.
`loo`	An R object of class `loo`. If `NULL`, brms will try to extract a precomputed `loo` object from the fitted model, added there via `add_criterion`.
`k_threshold`	The Pareto k threshold for which observations moment matching is applied. Defaults to `0.7`. See `pareto_k_ids` for more details.
`newdata`	An optional data.frame for which to evaluate predictions. If `NULL` (default), the original data of the model is used. `NA` values within factors (excluding grouping variables) 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. `NA` values within grouping variables are treated as a new level.
`resp`	Optional names of response variables. If specified, predictions are performed only for the specified response variables.

check	Logical; If TRUE (the default), some checks check are performed if the loo object was generated from the brmsfit object passed to argument fit.
recompile	Logical, indicating whether the Stan model should be recompiled. This may be necessary if you are running moment matching on another machine than the one used to fit the model. No recompilation is done by default.
...	Further arguments passed to the underlying methods. Additional arguments initially passed to loo, for example, newdata or resp need to be passed again to loo_moment_match in order for the latter to work correctly.
fit	An R object of class brmsfit.

Details

The moment matching algorithm requires draws of all variables defined in Stan's parameters block to be saved. Otherwise loo_moment_match cannot be computed. Thus, please set save_pars = save_pars(all = TRUE) in the call to brm, if you are planning to apply loo_moment_match to your models.

Value

An updated object of class loo.

References

Paananen, T., Piironen, J., Buerkner, P.-C., Vehtari, A. (2021). Implicitly Adaptive Importance Sampling. Statistics and Computing.

Examples

```
## Not run:
fit1 <- brm(count ~ zAge + zBase * Trt + (1|patient),
            data = epilepsy, family = poisson(),
            save_pars = save_pars(all = TRUE))

# throws warning about some pareto k estimates being too high
(loo1 <- loo(fit1))

# no more warnings after moment matching
(mmloo1 <- loo_moment_match(fit1, loo = loo1))

## 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_epred(
  object,
  type = c("mean", "var", "quantile"),
  probs = 0.5,
  psis_object = NULL,
  resp = NULL,
  ...
)

loo_epred(object, ...)

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

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

Arguments

`object`	An object of class `brmsfit`.

type	The statistic to be computed on the results. Can by either "mean" (default), "var", or "quantile".
probs	A vector of quantiles to compute. Only used if type = quantile.
psis_object	An optional object returned by psis. If psis_object 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, predictions are performed only for the specified response variables.
...	Optional arguments passed to the underlying methods that is log_lik, as well as posterior_predict, posterior_epred or posterior_linpred.
prob	For loo_predictive_interval, a scalar in $(0, 1)$ indicating the desired probability mass to include in the intervals. The default is prob = 0.9 (90% intervals).

Value

loo_predict, loo_epred, loo_linpred, and loo_predictive_interval all return a matrix with one row per observation and one column per summary statistic as specified by arguments type and probs. In multivariate or categorical models a third dimension is added to represent the response variables or categories, respectively.

loo_predictive_interval(..., prob = p) is equivalent to loo_predict(..., type = "quantile", probs = c(a, 1-a)) with a = (1 - p)/2.

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)
loo_epred(fit, type = "var", psis_object = psis)

## End(Not run)
```

`loo_R2.brmsfit` *Compute a LOO-adjusted R-squared for regression models*

Description

Compute a LOO-adjusted R-squared for regression models

Usage

```
## S3 method for class 'brmsfit'
loo_R2(
  object,
  resp = NULL,
  summary = TRUE,
  robust = FALSE,
  probs = c(0.025, 0.975),
  seed = NULL,
  args_epred = list(),
  args_loglik = list(),
  ...
)
```

Arguments

`object`	An object of class `brmsfit`.
`resp`	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
`summary`	Should summary statistics 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`.
`seed`	Optional integer used to initialize the random number generator.
`args_epred`	A named list of further arguments passed only to `posterior_epred`.
`args_loglik`	A named list of further arguments passed only to `log_lik`.
`...`	Further arguments passed to both `posterior_epred` and `log_lik`.

Value

If `summary = TRUE`, an M x C matrix is returned (M = number of response variables and c = `length(probs) + 2`) containing Bayesian bootstrap based summary statistics of the LOO-adjusted

R-squared values. If `summary = FALSE`, the Bayesian bootstrap draws of the LOO-adjusted R-squared values are returned in an S x M matrix (S is the number of draws).

@details LOO-R2 uses LOO residuals and is defined as $1 - Var_{loo-res}/Var_y$, with

$$Var_y = V_{n=1}^{N} y_n, and Var_{loo-res} = V_{n=1}^{N} \hat{e}_{loo,n},$$

where $\hat{e}_{loo,n} = y_n - \hat{y}_{loo,n}$. Bayesian bootstrap is used to draw from the approximated uncertainty distribution as described by Vehtari and Lampinen (2002).

References

Vehtari and Lampinen (2002). Bayesian model assessment and comparison using cross-validation predictive densities. Neural Computation, 14(10):2439-2468.

Examples

```
## Not run:
fit <- brm(mpg ~ wt + cyl, data = mtcars)
summary(fit)
loo_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))
loo_R2(fit, newdata = nd)

## End(Not run)
```

---

loo_subsample.brmsfit *Efficient approximate leave-one-out cross-validation (LOO) using subsampling*

Description

Efficient approximate leave-one-out cross-validation (LOO) using subsampling

Usage

```
## S3 method for class 'brmsfit'
loo_subsample(x, ..., compare = TRUE, resp = NULL, model_names = NULL)
```

Arguments

`x`	A `brmsfit` object.
`...`	More `brmsfit` objects or further arguments passed to the underlying post-processing functions. In particular, see `prepare_predictions` for further supported arguments.

compare	A flag indicating if the information criteria of the models should be compared to each other via loo_compare.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
model_names	If NULL (the default) will use model names derived from deparsing the call. Otherwise will use the passed values as model names.

Details

More details can be found on loo_subsample.

Examples

```
## Not run:
# model with population-level effects only
fit1 <- brm(rating ~ treat + period + carry,
            data = inhaler)
(loo1 <- loo_subsample(fit1))

# model with an additional varying intercept for subjects
fit2 <- brm(rating ~ treat + period + carry + (1|subject),
            data = inhaler)
(loo2 <- loo_subsample(fit2))

# compare both models
loo_compare(loo1, loo2)

## End(Not run)
```

---

loss *Cumulative Insurance Loss Payments*

Description

This dataset, discussed in Gesmann & Morris (2020), contains cumulative insurance loss payments over the course of ten years.

Usage

```
loss
```

Format

A data frame of 55 observations containing information on the following 4 variables.

AY Origin year of the insurance (1991 to 2000)

dev Deviation from the origin year in months

cum Cumulative loss payments

premium Achieved premiums for the given origin year

Source

Gesmann M. & Morris J. (2020). Hierarchical Compartmental Reserving Models. *CAS Research Papers*.

Examples

```
## Not run:
# non-linear model to predict cumulative loss payments
fit_loss <- brm(
  bf(cum ~ ult * (1 - exp(-(dev/theta)^omega)),
     ult ~ 1 + (1|AY), omega ~ 1, theta ~ 1,
     nl = TRUE),
  data = loss, family = gaussian(),
  prior = c(
    prior(normal(5000, 1000), nlpar = "ult"),
    prior(normal(1, 2), nlpar = "omega"),
    prior(normal(45, 10), nlpar = "theta")
  ),
  control = list(adapt_delta = 0.9)
)

# basic summaries
summary(fit_loss)
conditional_effects(fit_loss)

# plot predictions per origin year
conditions <- data.frame(AY = unique(loss$AY))
rownames(conditions) <- unique(loss$AY)
me_loss <- conditional_effects(
  fit_loss, conditions = conditions,
  re_formula = NULL, method = "predict"
)
plot(me_loss, ncol = 5, points = TRUE)

## End(Not run)
```

---

ma *Set up MA(q) correlation structures*

Description

Set up a moving average (MA) term of order q in **brms**. The function does not evaluate its arguments – it exists purely to help set up a model with MA terms.

Usage

```
ma(time = NA, gr = NA, q = 1, cov = FALSE)
```

Arguments

time	An optional time variable specifying the time ordering of the observations. By default, the existing order of the observations in the data is used.
gr	An optional grouping variable. If specified, the correlation structure is assumed to apply only to observations within the same grouping level.
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. This is currently only possible for stationary ARMA effects of order 1. If the model family does not have natural residuals, latent residuals are added automatically. If FALSE (the default), a regression formulation is used that is considerably faster and allows for ARMA effects of order higher than 1 but is only available for gaussian models and some of its generalizations.

Value

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

See Also

autocor-terms, arma, ar

Examples

```
## Not run:
data("LakeHuron")
LakeHuron <- as.data.frame(LakeHuron)
fit <- brm(x ~ ma(p = 2), data = LakeHuron)
summary(fit)

## 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 conditional_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

conditional_effects, rows2labels

Examples

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

---

mcmc_plot.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'
mcmc_plot(
  object,
  pars = NA,
  type = "intervals",
  variable = NULL,
  regex = FALSE,
  fixed = FALSE,
  ...
)

mcmc_plot(object, ...)
```

Arguments

object	An R object typically of class brmsfit
pars	Deprecated alias of variable. Names of the parameters to plot, as given by a character vector or a regular expression.
type	The type of the plot. Supported types are (as names) hist, dens, hist_by_chain, dens_overlay, violin, intervals, areas, acf, acf_bar,trace, trace_highlight, scatter, rhat, rhat_hist, neff, neff_hist nuts_acceptance, nuts_divergence, nuts_stepsize, nuts_treedepth, and nuts_energy. For an overview on the various plot types see MCMC-overview.
variable	Names of the variables (parameters) to plot, as given by a character vector or a regular expression (if regex = TRUE). By default, a hopefully not too large selection of variables is plotted.
regex	Logical; Indicates whether variable should be treated as regular expressions. Defaults to FALSE.
fixed	(Deprecated) Indicates whether parameter names should be matched exactly (TRUE) or treated as regular expressions (FALSE). Default is FALSE and only works with argument pars.
...	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 ~ zAge + zBase * Trt + (1|patient),
             data = epilepsy, family = "poisson")

# plot posterior intervals
mcmc_plot(model)

# only show population-level effects in the plots
mcmc_plot(model, variable = "^b_", regex = TRUE)

# show histograms of the posterior distributions
mcmc_plot(model, type = "hist")

# plot some diagnostics of the sampler
mcmc_plot(model, type = "neff")
mcmc_plot(model, type = "rhat")
```

```
# plot some diagnostics specific to the NUTS sampler
mcmc_plot(model, type = "nuts_acceptance")
mcmc_plot(model, type = "nuts_divergence")

## End(Not run)
```

---

me *Predictors with Measurement Error in* **brms** *Models*

Description

(Soft deprecated) Specify predictors with measurement error. The function does not evaluate its arguments – it exists purely to help set up a model.

Usage

```
me(x, sdx, 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). me terms are soft deprecated in favor of the more general and consistent mi terms. 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_pars = save_pars(latent = 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_pars = save_pars(latent = TRUE))
summary(fit2)

## End(Not run)
```

---

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

Description

Specify predictor term with missing values in **brms**. The function does not evaluate its arguments – it exists purely to help set up a model. For documentation on how to specify missing values in response variables, see `resp_mi`.

Usage

```
mi(x, idx = NA)
```

Arguments

`x`	The variable containing missing values.
`idx`	An optional variable containing indices of observations in 'x' that are to be used in the model. This is mostly relevant in partially subsetted models (via `resp_subset`) but may also have other applications that I haven't thought of.

Details

For detailed documentation see `help(brmsformula)`.

See Also

`brmsformula`

Examples

```
## Not run:
data("nhanes", package = "mice")
N <- nrow(nhanes)

# simple model with missing data
bform1 <- bf(bmi | mi() ~ age * mi(chl)) +
  bf(chl | mi() ~ age) +
  set_rescor(FALSE)
```

```
fit1 <- brm(bform1, data = nhanes)

summary(fit1)
plot(conditional_effects(fit1, resp = "bmi"), ask = FALSE)
loo(fit1, newdata = na.omit(fit1$data))

# simulate some measurement noise
nhanes$se <- rexp(N, 2)

# measurement noise can be handled within 'mi' terms
# with or without the presence of missing values
bform2 <- bf(bmi | mi() ~ age * mi(chl)) +
  bf(chl | mi(se) ~ age) +
  set_rescor(FALSE)

fit2 <- brm(bform2, data = nhanes)

summary(fit2)
plot(conditional_effects(fit2, resp = "bmi"), ask = FALSE)

# 'mi' terms can also be used when some responses are subsetted
nhanes$sub <- TRUE
nhanes$sub[1:2] <- FALSE
nhanes$id <- 1:N
nhanes$idx <- sample(3:N, N, TRUE)

# this requires the addition term 'index' being specified
# in the subsetted part of the model
bform3 <- bf(bmi | mi() ~ age * mi(chl, idx)) +
  bf(chl | mi(se) + subset(sub) + index(id) ~ age) +
  set_rescor(FALSE)

fit3 <- brm(bform3, data = nhanes)

summary(fit3)
plot(conditional_effects(fit3, resp = "bmi"), ask = FALSE)

## End(Not run)
```

`mixture` *Finite Mixture Families in* **brms**

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 ... and flist.
order	Ordering constraint to identify mixture components. If 'mu' or TRUE, population-level intercepts of the mean parameters are ordered in non-ordinal models and fixed to the same value in ordinal models (see details). 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.

Ordering intercepts in mixtures of ordinal families is not possible as each family has itself a set of vector of intercepts (i.e. ordinal thresholds). Instead, **brms** will fix the vector of intercepts across components in ordinal mixtures, if desired, so that users can try to identify the mixture model via selective inclusion of predictors.

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 init = 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,
            init = 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,
            init = 0, chains = 2)
summary(fit4)
pp_check(fit4)

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

## End(Not run)
```

`mm` *Set up multi-membership grouping terms in* **brms**

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,
  by = NULL,
```

```
  cor = TRUE,
  id = NA,
  pw = NULL,
  cov = NULL,
  dist = "gaussian"
)
```

Arguments

`...`	One or more terms containing grouping factors.
`weights`	A matrix specifying the membership 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), membership weights are standardized in order to sum to one per row. If negative weights are specified, `scale` needs to be set to `FALSE`.
`by`	An optional factor matrix, specifying sub-populations of the groups. It should have as many columns as grouping terms specified in `...`. For each level of the `by` variable, a separate variance-covariance matrix will be fitted. Levels of the grouping factor must be nested in levels of the `by` variable matrix.
`cor`	Logical. If `TRUE` (the default), group-level terms will be modelled as correlated.
`id`	Optional character string. All group-level terms across the model with the same `id` will be modeled as correlated (if `cor` is `TRUE`). See `brmsformula` for more details.
`pw`	Optional numeric matrix specifying prior weights. They weight the contribution of each group to the log-prior of the group-level coefficients. Should have as many columns as grouping terms specified in `...` and one distinct value for each group level.
`cov`	An optional matrix which is proportional to the within-group covariance matrix of the group-level effects. 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 `vignette("brms_phylogenetics")` for more details. By default, levels of the same grouping factor are modeled as independent of each other.
`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)
```

---

mo *Monotonic Predictors in* **brms** *Models*

Description

Specify a monotonic predictor term in **brms**. The function does not evaluate its arguments – it exists purely to help set up a model.

Usage

```
mo(x, id = NA)
```

Arguments

x	An integer variable or an ordered factor to be modeled as monotonic.
id	Optional character string. All monotonic terms with the same id within one formula will be modeled as having the same simplex (shape) parameter vector. If all monotonic terms of the same predictor have the same id, the resulting predictions will be conditionally monotonic for all values of interacting covariates (Bürkner & Charpentier, 2020).

Details

See Bürkner and Charpentier (2020) for the underlying theory. For detailed documentation of the formula syntax used for monotonic terms, see help(brmsformula) as well as vignette("brms_monotonic").

References

Bürkner P. C. & Charpentier E. (2020). Modeling Monotonic Effects of Ordinal Predictors in Regression Models. British Journal of Mathematical and Statistical Psychology. doi:10.1111/bmsp.12195

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)
summary(fit1)
plot(fit1, N = 6)
plot(conditional_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)
summary(fit2)
plot(conditional_effects(fit2), points = TRUE)

# ensure conditional monotonicity
fit3 <- brm(ls ~ mo(income, id = "i")*x, data = dat)
summary(fit3)
plot(conditional_effects(fit3), points = TRUE)

## End(Not run)
```

---

model_weights.brmsfit *Model Weighting Methods*

Description

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

Usage

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

model_weights(x, ...)
```

Arguments

x	A brmsfit object.
...	More brmsfit objects or further arguments passed to the underlying post-processing functions. In particular, see prepare_predictions for further supported arguments.
weights	Name of the criterion to compute weights from. Should be one of "loo", "waic", "kfold", "stacking" (current default), "bma", or "pseudobma". For the former three options, Akaike weights will be computed based on the information criterion values returned by the respective methods. For "stacking" and "pseudobma", method loo_model_weights will be used to obtain weights. For "bma", method post_prob will be used to compute Bayesian model averaging weights based on log marginal likelihood values (make sure to specify reasonable priors in this case). For some methods, weights may also 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 mu and covariance matrix Sigma.

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 draws to sample from the distribution.

Details

See the Stan user's manual https://mc-stan.org/docs/ 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 mu, covariance matrix Sigma, 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

x	Vector or matrix of quantiles. If x is a matrix, each row is taken to be a quantile.
df	Vector of degrees of freedom.
mu	Location 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 draws to sample from the distribution.

Details

See the Stan user's manual https://mc-stan.org/docs/ for details on the parameterization

mvbind *Bind response variables in multivariate models*

Description

Can be used to specify a multivariate **brms** model within a single formula. Outside of `brmsformula`, it just behaves like `cbind`.

Usage

```
mvbind(...)
```

Arguments

`...`	Same as in `cbind`

See Also

`brmsformula`, `mvbrmsformula`

Examples

```
bf(mvbind(y1, y2) ~ x)
```

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 `formula` or `brmsformula`, each specifying a univariate model. See `brmsformula` for details on how to specify univariate models.
`flist`	Optional list of formulas, which are treated in the same way as formulas passed via the `...` argument.
`rescor`	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), `rescor` 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)
```

---

ngrps.brmsfit *Number of Grouping Factor Levels*

Description

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

Usage

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

ngrps(object, ...)
```

Arguments

`object`	An R object.
`...`	Currently ignored.

Value

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

nsamples.brmsfit *(Deprecated) Number of Posterior Samples*

Description

Extract the number of posterior samples (draws) stored in a fitted Bayesian model. Method `nsamples` is deprecated. Please use `ndraws` instead.

Usage

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

Arguments

`object`	An object of class `brmsfit`.
`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.
`...`	Currently ignored.

opencl *GPU support in Stan via OpenCL*

Description

Use OpenCL for GPU support in **Stan** via the **brms** interface. Only some **Stan** functions can be run on a GPU at this point and so a lot of **brms** models won't benefit from OpenCL for now.

Usage

```
opencl(ids = NULL)
```

Arguments

`ids`	(integer vector of length 2) The platform and device IDs of the OpenCL device to use for fitting. If you don't know the IDs of your OpenCL device, `c(0,0)` is most likely what you need.

Details

For more details on OpenCL in **Stan**, check out https://mc-stan.org/docs/2_26/cmdstan-guide/parallelization.html#opencl as well as https://mc-stan.org/docs/2_26/stan-users-guide/opencl.html.

Value

A brmsopencl object which can be passed to the opencl argument of brm and related functions.

Examples

```
## Not run:
# this model just serves as an illustration
# OpenCL may not actually speed things up here
fit <- brm(count ~ zAge + zBase * Trt + (1|patient),
           data = epilepsy, family = poisson(),
           chains = 2, cores = 2, opencl = opencl(c(0, 0)),
           backend = "cmdstanr")
summary(fit)

## End(Not run)
```

---

pairs.brmsfit *Create a matrix of output plots from a* brmsfit *object*

Description

A pairs method that is customized for MCMC output.

Usage

```
## S3 method for class 'brmsfit'
pairs(x, pars = NA, variable = NULL, regex = FALSE, fixed = FALSE, ...)
```

Arguments

x	An object of class brmsfit
pars	Deprecated alias of variable. Names of the parameters to plot, as given by a character vector or a regular expression.
variable	Names of the variables (parameters) to plot, as given by a character vector or a regular expression (if regex = TRUE). By default, a hopefully not too large selection of variables is plotted.
regex	Logical; Indicates whether variable should be treated as regular expressions. Defaults to FALSE.
fixed	(Deprecated) Indicates whether parameter names should be matched exactly (TRUE) or treated as regular expressions (FALSE). Default is FALSE and only works with argument pars.
...	Further arguments to be passed to mcmc_pairs.

Details

For a detailed description see mcmc_pairs.

Examples

```
## Not run:
fit <- brm(count ~ zAge + zBase * Trt
           + (1|patient) + (1|visit),
           data = epilepsy, family = "poisson")
pairs(fit, variable = variables(fit)[1:3])
pairs(fit, variable = "^sd_", regex = TRUE)

## End(Not run)
```

`parnames` *Extract Parameter Names*

Description

Extract all parameter names of a given model.

Usage

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

Arguments

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

Value

A character vector containing the parameter names of the model.

`plot.brmsfit` *Trace and Density Plots for MCMC Draws*

Description

Trace and Density Plots for MCMC Draws

Usage

```
## S3 method for class 'brmsfit'
plot(
  x,
  pars = NA,
  combo = c("hist", "trace"),
  nvariables = 5,
  N = NULL,
  variable = NULL,
  regex = FALSE,
  fixed = FALSE,
  bins = 30,
  theme = NULL,
  plot = TRUE,
  ask = TRUE,
  newpage = TRUE,
  ...
)
```

Arguments

x	An object of class brmsfit.
pars	Deprecated alias of variable. Names of the parameters to plot, as given by a character vector or a regular expression.
combo	A character vector with at least two elements. Each element of combo corresponds to a column in the resulting graphic and should be the name of one of the available MCMC functions (omitting the mcmc_ prefix).
nvariables	The number of variables (parameters) plotted per page.
N	Deprecated alias of nvariables.
variable	Names of the variables (parameters) to plot, as given by a character vector or a regular expression (if regex = TRUE). By default, a hopefully not too large selection of variables is plotted.
regex	Logical; Indicates whether variable should be treated as regular expressions. Defaults to FALSE.
fixed	(Deprecated) Indicates whether parameter names should be matched exactly (TRUE) or treated as regular expressions (FALSE). Default is FALSE and only works with argument pars.
bins	Number of bins used for posterior histograms (defaults to 30).
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.

Examples

```
## Not run:
fit <- brm(count ~ zAge + zBase * Trt
           + (1|patient) + (1|visit),
           data = epilepsy, family = "poisson")
plot(fit)
## plot population-level effects only
plot(fit, variable = "^b_", regex = TRUE)

## End(Not run)
```

---

posterior_average.brmsfit

Posterior draws of parameters averaged across models

Description

Extract posterior draws 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,
  ...,
  variable = NULL,
  pars = NULL,
  weights = "stacking",
  ndraws = NULL,
  nsamples = NULL,
  missing = NULL,
  model_names = NULL,
  control = list(),
  seed = NULL
)

posterior_average(x, ...)
```

Arguments

x	A brmsfit object.
...	More brmsfit objects or further arguments passed to the underlying post-processing functions. In particular, see prepare_predictions for further supported arguments.
variable	Names of variables (parameters) for which to average across models. Only those variables can be averaged that appear in every model. Defaults to all overlapping variables.
pars	Deprecated alias of variable.
weights	Name of the criterion to compute weights from. Should be one of "loo", "waic", "kfold", "stacking" (current default), "bma", or "pseudobma". For the former three options, Akaike weights will be computed based on the information criterion values returned by the respective methods. For "stacking" and "pseudobma", method loo_model_weights will be used to obtain weights. For "bma", method post_prob will be used to compute Bayesian model averaging weights based on log marginal likelihood values (make sure to specify reasonable priors in this case). For some methods, weights may also be a numeric vector of pre-specified weights.
ndraws	Total number of posterior draws to use.
nsamples	Deprecated alias of ndraws.
missing	An optional numeric value or a named list of numeric values to use if a model does not contain a variable for which posterior draws should be averaged. Defaults to NULL, in which case only those variables can be averaged that are present in all of the models.
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 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 draws.

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_epred.brmsfit` *Draws from the Expected Value of the Posterior Predictive Distribution*

Description

Compute posterior draws of the expected value of the posterior predictive distribution. 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 posterior predictions performed by the `posterior_predict.brmsfit` method. This is because only the uncertainty in the expected value of the posterior predictive distribution is incorporated in the draws computed by `posterior_epred` while the residual error is ignored there. However, the estimated means of both methods averaged across draws should be very similar.

Usage

```
## S3 method for class 'brmsfit'
posterior_epred(
  object,
  newdata = NULL,
  re_formula = NULL,
  re.form = NULL,
  resp = NULL,
  dpar = NULL,
  nlpar = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  sort = FALSE,
  ...
)
```

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. NA values within factors (excluding grouping variables) 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. NA values within grouping variables are treated as a new level.
re_formula	formula containing group-level effects to be considered in the prediction. If NULL (default), include all group-level effects; if NA or ~0, include no group-level effects.
re.form	Alias of re_formula.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
dpar	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
nlpar	Optional name of a predicted non-linear parameter. If specified, expected predictions of this parameters are returned.
ndraws	Positive integer indicating how many posterior draws should be used. If NULL (the default) all draws are used. Ignored if draw_ids is not NULL.
draw_ids	An integer vector specifying the posterior draws to be used. If NULL (the default), all draws are used.
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).
...	Further arguments passed to prepare_predictions that control several aspects of data validation and prediction.

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.

In multilevel models, it is possible to allow new levels of grouping factors to be used in the predictions. This can be controlled via argument allow_new_levels. New levels can be sampled in multiple ways, which can be controlled via argument sample_new_levels. Both of these arguments are documented in prepare_predictions along with several other useful arguments to control specific aspects of the predictions.

Value

An array of draws. For categorical and ordinal models, the output is an S x N x C array. Otherwise, the output is an S x N matrix, where S is the number of posterior draws, N is the number of observations, and C is the number of categories. In multivariate models, an additional dimension is added to the output which indexes along the different response variables.

Examples

```
## Not run:
## fit a model
```

```
fit <- brm(rating ~ treat + period + carry + (1|subject),
           data = inhaler)

## compute expected predictions
ppe <- posterior_epred(fit)
str(ppe)

## 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, variable = NULL, prob = 0.95, ...)
```

Arguments

`object`	An object of class `brmsfit`.
`pars`	Deprecated alias of `variable`. For reasons of backwards compatibility, `pars` is interpreted as a vector of regular expressions by default unless `fixed = TRUE` is specified.
`variable`	A character vector providing the variables to extract. By default, all variables 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 variables.

Examples

```
## Not run:
fit <- brm(count ~ zAge + zBase * Trt,
           data = epilepsy, family = negbinomial())
posterior_interval(fit)

## End(Not run)
```

posterior_linpred.brmsfit
Posterior Draws of the Linear Predictor

Description

Compute posterior draws of the linear predictor, that is draws before applying any link functions or other transformations. Can be performed for the data used to fit the model (posterior predictive checks) or for new data.

Usage

```
## S3 method for class 'brmsfit'
posterior_linpred(
  object,
  transform = FALSE,
  newdata = NULL,
  re_formula = NULL,
  re.form = NULL,
  resp = NULL,
  dpar = NULL,
  nlpar = NULL,
  incl_thres = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  sort = FALSE,
  ...
)
```

Arguments

object	An object of class brmsfit.
transform	Logical; if FALSE (the default), draws of the linear predictor are returned. If TRUE, draws of the transformed linear predictor, that is, after applying the inverse link function are returned.
newdata	An optional data.frame for which to evaluate predictions. If NULL (default), the original data of the model is used. NA values within factors (excluding grouping variables) 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. NA values within grouping variables are treated as a new level.
re_formula	formula containing group-level effects to be considered in the prediction. If NULL (default), include all group-level effects; if NA or ~0, include no group-level effects.
re.form	Alias of re_formula.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.

dpar	Name of a predicted distributional parameter for which draws are to be returned. By default, draws of the main distributional parameter(s) "mu" are returned.
nlpar	Optional name of a predicted non-linear parameter. If specified, expected predictions of this parameters are returned.
incl_thres	Logical; only relevant for ordinal models when transform is FALSE, and ignored otherwise. Shall the thresholds and category-specific effects be included in the linear predictor? For backwards compatibility, the default is to not include them.
ndraws	Positive integer indicating how many posterior draws should be used. If NULL (the default) all draws are used. Ignored if draw_ids is not NULL.
draw_ids	An integer vector specifying the posterior draws to be used. If NULL (the default), all draws are used.
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).
...	Further arguments passed to prepare_predictions that control several aspects of data validation and prediction.

See Also

posterior_epred.brmsfit

Examples

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

## extract linear predictor values
pl <- posterior_linpred(fit)
str(pl)

## End(Not run)
```

---

posterior_predict.brmsfit

Draws from the Posterior Predictive Distribution

Description

Compute posterior draws of the posterior predictive distribution. Can be performed for the data used to fit the model (posterior predictive checks) or for new data. By definition, these draws have higher variance than draws of the expected value of the posterior predictive distribution computed by posterior_epred.brmsfit. This is because the residual error is incorporated in posterior_predict. However, the estimated means of both methods averaged across draws should be very similar.

Usage

```
## S3 method for class 'brmsfit'
posterior_predict(
  object,
  newdata = NULL,
  re_formula = NULL,
  re.form = NULL,
  transform = NULL,
  resp = NULL,
  negative_rt = FALSE,
  ndraws = NULL,
  draw_ids = NULL,
  sort = FALSE,
  ntrys = 5,
  cores = NULL,
  ...
)
```

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. NA values within factors (excluding grouping variables) 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. NA values within grouping variables are treated as a new level.
re_formula	formula containing group-level effects to be considered in the prediction. If NULL (default), include all group-level effects; if NA or ~0, include no group-level effects.
re.form	Alias of re_formula.
transform	(Deprecated) A function or a character string naming a function to be applied on the predicted responses before summary statistics are computed.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
negative_rt	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.
ndraws	Positive integer indicating how many posterior draws should be used. If NULL (the default) all draws are used. Ignored if draw_ids is not NULL.
draw_ids	An integer vector specifying the posterior draws to be used. If NULL (the default), all draws are used.
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).

ntrys	Parameter used in rejection sampling for truncated discrete models only (defaults to 5). See Details for more information.
cores	Number of cores (defaults to 1). On non-Windows systems, this argument can be set globally via the mc.cores option.
...	Further arguments passed to prepare_predictions that control several aspects of data validation and prediction.

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.

In multilevel models, it is possible to allow new levels of grouping factors to be used in the predictions. This can be controlled via argument allow_new_levels. New levels can be sampled in multiple ways, which can be controlled via argument sample_new_levels. Both of these arguments are documented in prepare_predictions along with several other useful arguments to control specific aspects of the predictions.

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

An array of draws. In univariate models, the output is as an S x N matrix, where S is the number of posterior draws and N is the number of observations. In multivariate models, an additional dimension is added to the output which indexes along the different response variables.

Examples

```
## Not run:
## fit a model
fit <- brm(time | cens(censored) ~ age + sex + (1 + age || patient),
           data = kidney, family = "exponential", init = "0")

## predicted responses
pp <- posterior_predict(fit)
str(pp)

## predicted responses excluding the group-level effect of age
pp <- posterior_predict(fit, re_formula = ~ (1 | patient))
str(pp)

## predicted responses of patient 1 for new data
newdata <- data.frame(
  sex = factor(c("male", "female")),
  age = c(20, 50),
```

```
  patient = c(1, 1)
)
pp <- posterior_predict(fit, newdata = newdata)
str(pp)

## End(Not run)
```

`posterior_samples.brmsfit` *(Deprecated) Extract Posterior Samples*

Description

Extract posterior samples of specified parameters. The `posterior_samples` method is deprecated. We recommend using the more modern and consistent `as_draws_*` extractor functions of the **posterior** package instead.

Usage

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

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

Arguments

`x`	An `R` object typically 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.
`fixed`	Indicates whether parameter names should be matched exactly (`TRUE`) or treated as regular expressions (`FALSE`). Default is `FALSE`.
`add_chain`	A flag indicating if the returned `data.frame` should contain two additional columns. The `chain` column indicates the chain in which each sample was generated, the `iter` column indicates the iteration number within each chain.
`subset`	A numeric vector indicating the rows (i.e., posterior samples) to be returned. If `NULL` (the default), all posterior samples are returned.

as.matrix	Should the output be a matrix instead of a data.frame? Defaults to FALSE.
as.array	Should the output be an array instead of a data.frame? Defaults to FALSE.
...	Arguments passed to individual methods (if applicable).

Value

A data.frame (matrix or array) containing the posterior samples.

See Also

as_draws, as.data.frame

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, pars = "^b")
head(samples1)

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

## End(Not run)
```

---

posterior_smooths.brmsfit

Posterior Predictions of Smooth Terms

Description

Compute posterior predictions of smooth s and t2 terms of models fitted with **brms**.

Usage

```
## S3 method for class 'brmsfit'
posterior_smooths(
  object,
  smooth,
  newdata = NULL,
  resp = NULL,
  dpar = NULL,
  nlpar = NULL,
  ndraws = NULL,
```

```
  draw_ids = NULL,
  ...
)

posterior_smooths(object, ...)
```

Arguments

object	An object of class brmsfit.
smooth	Name of a single smooth term for which predictions should be computed.
newdata	An optional data.frame for which to evaluate predictions. If NULL (default), the original data of the model is used. Only those variables appearing in the chosen smooth term are required.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
dpar	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
nlpar	Optional name of a predicted non-linear parameter. If specified, expected predictions of this parameters are returned.
ndraws	Positive integer indicating how many posterior draws should be used. If NULL (the default) all draws are used. Ignored if draw_ids is not NULL.
draw_ids	An integer vector specifying the posterior draws to be used. If NULL (the default), all draws are used.
...	Currently ignored.

Value

An S x N matrix, where S is the number of posterior draws and N is the number of observations.

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)
summary(fit)

newdata <- data.frame(x2 = seq(0, 1, 10))
str(posterior_smooths(fit, smooth = "s(x2)", newdata = newdata))

## End(Not run)
```

posterior_summary *Summarize Posterior draws*

Description

Summarizes posterior draws based on point estimates (mean or median), estimation errors (SD or MAD) and quantiles. This function mainly exists to retain backwards compatibility. It will eventually be replaced by functions of the **posterior** package (see examples below).

Usage

```
posterior_summary(x, ...)

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

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

Arguments

x	An R object.
...	More arguments passed to or from other methods.
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.
pars	Deprecated alias of variable. For reasons of backwards compatibility, pars is interpreted as a vector of regular expressions by default unless fixed = TRUE is specified.
variable	A character vector providing the variables to extract. By default, all variables are extracted.

Value

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

See Also

summarize_draws

Examples

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

# recommended workflow using posterior
library(posterior)
draws <- as_draws_array(fit)
summarise_draws(draws, default_summary_measures())

## End(Not run)
```

`posterior_table` *Table Creation for Posterior Draws*

Description

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

Usage

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

Arguments

`x`	A matrix of posterior draws where rows indicate draws 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 draws.

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

x	A brmsfit object.
...	More brmsfit objects or further arguments passed to the underlying post-processing functions. In particular, see prepare_predictions for further supported arguments.
prior_prob	Numeric vector with prior model probabilities. If omitted, a uniform prior is used (i.e., all models are equally likely a priori). The default NULL corresponds to equal prior model 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.

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.

The computation of model probabilities based on bridge sampling requires a lot more posterior samples than usual. A good conservative rule of thump is perhaps 10-fold more samples (read: the default of 4000 samples may not be enough in many cases). If not enough posterior samples are provided, the bridge sampling algorithm tends to be unstable leading to considerably different results each time it is run. We thus recommend running post_prob multiple times to check the stability of the results.

More details are provided under bridgesampling::post_prob.

See Also

bridge_sampler, bayes_factor

Examples

```
## Not run:
# model with the treatment effect
fit1 <- brm(
  count ~ zAge + zBase + Trt,
  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 ~ zAge + zBase,
  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 draws averaged across models*

Description

Compute posterior predictive draws 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 = "stacking",
  method = "posterior_predict",
  ndraws = NULL,
  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 brmsfit object.
...	More brmsfit objects or further arguments passed to the underlying post-processing functions. In particular, see prepare_predictions for further supported arguments.
weights	Name of the criterion to compute weights from. Should be one of "loo", "waic", "kfold", "stacking" (current default), "bma", or "pseudobma". For the former three options, Akaike weights will be computed based on the information criterion values returned by the respective methods. For "stacking" and "pseudobma", method loo_model_weights will be used to obtain weights. For "bma", method post_prob will be used to compute Bayesian model averaging weights based on log marginal likelihood values (make sure to specify reasonable priors in this case). For some methods, weights may also be a numeric vector of pre-specified weights.
method	Method used to obtain predictions to average over. Should be one of "posterior_predict" (default), "posterior_epred", "posterior_linpred" or "predictive_error".
ndraws	Total number of posterior draws to use.
nsamples	Deprecated alias of ndraws.
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 quantile function. Only used if summary 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.
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 weights.
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,
  ndraws = NULL,
  prefix = c("ppc", "ppd"),
  group = NULL,
  x = NULL,
  newdata = NULL,
  resp = NULL,
  draw_ids = NULL,
  nsamples = NULL,
  subset = NULL,
  ...
)
```

Arguments

object	An object of class brmsfit.
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. type = "xyz") to get a list of supported types in the resulting error message.
ndraws	Positive integer indicating how many posterior draws should be used. If NULL all draws are used. If not specified, the number of posterior draws is chosen automatically. Ignored if draw_ids is not NULL.
prefix	The prefix of the **bayesplot** function to be applied. Either ‘"ppc"‘ (posterior predictive check; the default) or ‘"ppd"‘ (posterior predictive distribution), the latter being the same as the former except that the observed data is not shown for ‘"ppd"‘.
group	Optional name of a factor variable in the model by which to stratify the ppc plot. This argument is required for ppc *_grouped types and ignored otherwise.
x	Optional name of a variable in the model. Only used for ppc types having an x argument and ignored otherwise.
newdata	An optional data.frame for which to evaluate predictions. If NULL (default), the original data of the model is used. NA values within factors (excluding grouping variables) 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. NA values within grouping variables are treated as a new level.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
draw_ids	An integer vector specifying the posterior draws to be used. If NULL (the default), all draws are used.
nsamples	Deprecated alias of ndraws.
subset	Deprecated alias of draw_ids.
...	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 ~ zAge + zBase * Trt
            + (1|patient) + (1|obs),
            data = epilepsy, family = poisson())
```

```
pp_check(fit)  # shows dens_overlay plot by default
pp_check(fit, type = "error_hist", ndraws = 11)
pp_check(fit, type = "scatter_avg", ndraws = 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")

## get a plot without the observed data
pp_check(fit, prefix = "ppd")

## 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,
  resp = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  log = FALSE,
  summary = TRUE,
  robust = FALSE,
  probs = c(0.025, 0.975),
  ...
)

pp_mixture(x, ...)
```

Arguments

`x`	An R object usually of class `brmsfit`.

newdata	An optional data.frame for which to evaluate predictions. If NULL (default), the original data of the model is used. NA values within factors (excluding grouping variables) 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. NA values within grouping variables are treated as a new level.
re_formula	formula containing group-level effects to be considered in the prediction. If NULL (default), include all group-level effects; if NA or ~0, include no group-level effects.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
ndraws	Positive integer indicating how many posterior draws should be used. If NULL (the default) all draws are used. Ignored if draw_ids is not NULL.
draw_ids	An integer vector specifying the posterior draws to be used. If NULL (the default), all draws are used.
log	Logical; Indicates whether to return probabilities on the log-scale.
summary	Should summary statistics 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.
...	Further arguments passed to prepare_predictions that control several aspects of data validation and prediction.

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

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

is a normalizing constant.

Value

If summary = TRUE, an N x E x 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 x N x K array, where S is the number of posterior draws.

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, init = 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 *Draws from the Posterior Predictive Distribution*

Description

This method is an alias of posterior_predict.brmsfit with additional arguments for obtaining summaries of the computed draws.

Usage

```
## S3 method for class 'brmsfit'
predict(
  object,
  newdata = NULL,
  re_formula = NULL,
  transform = NULL,
```

```
  resp = NULL,
  negative_rt = FALSE,
  ndraws = NULL,
  draw_ids = NULL,
  sort = FALSE,
  ntrys = 5,
  cores = NULL,
  summary = TRUE,
  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. NA values within factors (excluding grouping variables) 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. NA values within grouping variables are treated as a new level.
re_formula	formula containing group-level effects to be considered in the prediction. If NULL (default), include all group-level effects; if NA or ~0, include no group-level effects.
transform	(Deprecated) A function or a character string naming a function to be applied on the predicted responses before summary statistics are computed.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
negative_rt	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.
ndraws	Positive integer indicating how many posterior draws should be used. If NULL (the default) all draws are used. Ignored if draw_ids is not NULL.
draw_ids	An integer vector specifying the posterior draws to be used. If NULL (the default), all draws are used.
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).
ntrys	Parameter used in rejection sampling for truncated discrete models only (defaults to 5). See Details for more information.
cores	Number of cores (defaults to 1). On non-Windows systems, this argument can be set globally via the mc.cores option.
summary	Should summary statistics 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.
...	Further arguments passed to prepare_predictions that control several aspects of data validation and prediction.

Value

An array of predicted response values. If summary = FALSE the output resembles those of posterior_predict.brmsfit.

If summary = TRUE the output depends on the family: For categorical and ordinal families, the output is an N x C matrix, where N is the number of observations, C is the number of categories, and the values are predicted category probabilities. For all other families, the output is a N x E matrix where E = 2 + length(probs) is the number of summary statistics: The Estimate column contains point estimates (either mean or median depending on argument robust), while the Est.Error column contains uncertainty estimates (either standard deviation or median absolute deviation depending on argument robust). The remaining columns starting with Q contain quantile estimates as specified via argument probs.

See Also

posterior_predict.brmsfit

Examples

```
## Not run:
## fit a model
fit <- brm(time | cens(censored) ~ age + sex + (1 + age || patient),
           data = kidney, family = "exponential", init = "0")

## predicted responses
pp <- predict(fit)
head(pp)

## predicted responses excluding the group-level effect of age
pp <- predict(fit, re_formula = ~ (1 | patient))
head(pp)

## 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)
```

predictive_error.brmsfit *Posterior Draws of Predictive Errors*

Description

Compute posterior draws of predictive errors, that is, observed minus predicted responses. Can be performed for the data used to fit the model (posterior predictive checks) or for new data.

Usage

```
## S3 method for class 'brmsfit'
predictive_error(
  object,
  newdata = NULL,
  re_formula = NULL,
  re.form = NULL,
  method = "posterior_predict",
  resp = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  sort = FALSE,
  ...
)
```

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. NA values within factors (excluding grouping variables) 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. NA values within grouping variables are treated as a new level.
re_formula	formula containing group-level effects to be considered in the prediction. If NULL (default), include all group-level effects; if NA or ~0, include no group-level effects.
re.form	Alias of re_formula.
method	Method used to obtain predictions. Can be set to "posterior_predict" (the default), "posterior_epred", or "posterior_linpred". For more details, see the respective function documentations.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
ndraws	Positive integer indicating how many posterior draws should be used. If NULL (the default) all draws are used. Ignored if draw_ids is not NULL.

draw_ids	An integer vector specifying the posterior draws to be used. If NULL (the default), all draws are used.
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).
...	Further arguments passed to prepare_predictions that control several aspects of data validation and prediction.

Value

An S x N array of predictive error draws, where S is the number of posterior draws and N is the number of observations.

Examples

```
## Not run:
## fit a model
fit <- brm(rating ~ treat + period + carry + (1|subject),
           data = inhaler, cores = 2)

## extract predictive errors
pe <- predictive_error(fit)
str(pe)

## End(Not run)
```

---

predictive_interval.brmsfit

Predictive Intervals

Description

Compute intervals from the posterior predictive distribution.

Usage

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

Arguments

object	An R object of class brmsfit.
prob	A number p (0 < p < 1) indicating the desired probability mass to include in the intervals. Defaults to 0.9.
...	Further arguments passed to posterior_predict.

Value

A matrix with 2 columns for the lower and upper bounds of the intervals, respectively, and as many rows as observations being predicted.

Examples

```
## Not run:
fit <- brm(count ~ zBase, data = epilepsy, family = poisson())
predictive_interval(fit)

## End(Not run)
```

---

prepare_predictions.brmsfit

Prepare Predictions

Description

This method helps in preparing **brms** models for certain post-processing tasks most notably various forms of predictions. Unless you are a package developer, you will rarely need to call `prepare_predictions` directly.

Usage

```
## S3 method for class 'brmsfit'
prepare_predictions(
  x,
  newdata = NULL,
  re_formula = NULL,
  allow_new_levels = FALSE,
  sample_new_levels = "uncertainty",
  incl_autocor = TRUE,
  oos = NULL,
  resp = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  nsamples = NULL,
  subset = NULL,
  nug = NULL,
  smooths_only = FALSE,
  offset = TRUE,
  newdata2 = NULL,
  new_objects = NULL,
  point_estimate = NULL,
  ndraws_point_estimate = 1,
  ...
```

```
)

prepare_predictions(x, ...)
```

Arguments

x	An R object typically of class 'brmsfit'.
newdata	An optional data.frame for which to evaluate predictions. If NULL (default), the original data of the model is used. NA values within factors (excluding grouping variables) 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. NA values within grouping variables are treated as a new level.
re_formula	formula containing group-level effects to be considered in the prediction. If NULL (default), include all group-level effects; if NA or ~0, include no group-level effects.
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), each posterior sample for a new level is drawn from the posterior draws of a randomly chosen existing level. Each posterior sample for a new level may be drawn from a different existing level such that the resulting set of new posterior draws represents the variation across 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 or predicting new levels in situations where relatively few levels where observed in the old_data. If "old_levels", directly sample new levels from the existing levels, where a new level is assigned all of the posterior draws of the same (randomly chosen) existing level.
incl_autocor	A flag indicating if correlation structures originally specified via autocor should be included in the predictions. Defaults to TRUE.
oos	Optional indices of observations for which to compute out-of-sample rather than in-sample predictions. Only required in models that make use of response values to make predictions, that is, currently only ARMA models.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
ndraws	Positive integer indicating how many posterior draws should be used. If NULL (the default) all draws are used. Ignored if draw_ids is not NULL.
draw_ids	An integer vector specifying the posterior draws to be used. If NULL (the default), all draws are used.
nsamples	Deprecated alias of ndraws.
subset	Deprecated alias of draw_ids.

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.
smooths_only	Logical; If TRUE only predictions related to smoothing splines (i.e., s or t2) will be computed. Defaults to FALSE.
offset	Logical; Indicates if offsets should be included in the predictions. Defaults to TRUE.
newdata2	A named list of objects containing new data, which cannot be passed via argument newdata. Required for some objects used in autocorrelation structures, or stanvars.
new_objects	Deprecated alias of newdata2.
point_estimate	Shall the returned object contain only point estimates of the parameters instead of their posterior draws? Defaults to NULL in which case no point estimate is computed. Alternatively, may be set to "mean" or "median". This argument is primarily implemented to ensure compatibility with the loo_subsample method.
ndraws_point_estimate	Only used if point_estimate is not NULL. How often shall the point estimate's value be repeated? Defaults to 1.
...	Further arguments passed to validate_newdata.

Value

An object of class 'brmsprep' or 'mvbrmsprep', depending on whether a univariate or multivariate model is passed.

`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, short = getOption("brms.short_summary", FALSE), ...)
```

Arguments

x	An object of class brmsfit
digits	The number of significant digits for printing out the summary; defaults to 2. The effective sample size is always rounded to integers.

short	A flag indicating whether to provide a shorter summary with less informational text. Defaults to FALSE. Can be set globally for the current session via the brms.short_summary option.
...	Additional arguments that would be passed to method summary of brmsfit.

See Also

summary.brmsfit

---

print.brmsprior *Print method for* brmsprior *objects*

Description

Print method for brmsprior objects

Usage

```
## S3 method for class 'brmsprior'
print(x, show_df = NULL, ...)
```

Arguments

x	An object of class brmsprior.
show_df	Logical; Print priors as a single data.frame (TRUE) or as a sequence of sampling statements (FALSE)?
...	Currently ignored.

---

prior_draws.brmsfit *Extract Prior Draws*

Description

Extract prior draws of specified parameters

Usage

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

prior_draws(x, ...)

prior_samples(x, ...)
```

Arguments

x	An R object typically of class brmsfit.
variable	A character vector providing the variables to extract. By default, all variables are extracted.
pars	Deprecated alias of variable. For reasons of backwards compatibility, pars is interpreted as a vector of regular expressions by default unless fixed = TRUE is specified.
...	Arguments passed to individual methods (if applicable).

Details

To make use of this function, the model must contain draws of prior distributions. This can be ensured by setting sample_prior = TRUE in function brm. Priors of certain parameters cannot be saved for technical reasons. For instance, this is the case for the population-level intercept, which is only computed after fitting the model by default. If you want to treat the intercept as part of all the other regression coefficients, so that sampling from its prior becomes possible, use ... ~ 0 + Intercept + ... in the formulas.

Value

A data.frame containing the prior draws.

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 draws
draws1 <- prior_draws(fit)
head(draws1)

# extract prior draws for the coefficient of 'treat'
draws2 <- prior_draws(fit, "b_treat")
head(draws2)

## End(Not run)
```

---

prior_summary.brmsfit *Priors of* brms *models*

Description

Extract priors of models fitted with **brms**.

Usage

```
## S3 method for class 'brmsfit'
prior_summary(object, all = TRUE, ...)
```

Arguments

object	An object of class brmsfit.
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

An brmsprior object.

Examples

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

---

psis.brmsfit *Pareto smoothed importance sampling (PSIS)*

Description

Implementation of Pareto smoothed importance sampling (PSIS), a method for stabilizing importance ratios. The version of PSIS implemented here corresponds to the algorithm presented in Vehtari, Simpson, Gelman, Yao, and Gabry (2024). For PSIS diagnostics see the pareto-k-diagnostic page.

Usage

```
## S3 method for class 'brmsfit'
psis(log_ratios, newdata = NULL, resp = NULL, model_name = NULL, ...)
```

Arguments

log_ratios	A fitted model object of class brmsfit. Argument is named "log_ratios" to match the argument name of the loo::psis generic function.
newdata	An optional data.frame for which to evaluate predictions. If NULL (default), the original data of the model is used. NA values within factors (excluding grouping variables) 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. NA values within grouping variables are treated as a new level.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
model_name	Currently ignored.
...	Further arguments passed to log_lik and loo::psis.

Value

The psis() methods return an object of class "psis", which is a named list with the following components:

log_weights Vector or matrix of smoothed (and truncated) but *unnormalized* log weights. To get normalized weights use the weights() method provided for objects of class "psis".

diagnostics A named list containing two vectors:

- pareto_k: Estimates of the shape parameter k of the generalized Pareto distribution. See the pareto-k-diagnostic page for details.
- n_eff: PSIS effective sample size estimates.

Objects of class "psis" also have the following attributes:

norm_const_log Vector of precomputed values of colLogSumExps(log_weights) that are used internally by the weights method to normalize the log weights.

tail_len Vector of tail lengths used for fitting the generalized Pareto distribution.

r_eff If specified, the user's r_eff argument.

dims Integer vector of length 2 containing S (posterior sample size) and N (number of observations).

method Method used for importance sampling, here psis.

References

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

Vehtari, A., Simpson, D., Gelman, A., Yao, Y., and Gabry, J. (2024). Pareto smoothed importance sampling. *Journal of Machine Learning Research*, 25(72):1-58. PDF

Examples

```
## Not run:
fit <- brm(rating ~ treat + period + carry, data = inhaler)
psis(fit)

## End(Not run)
```

R2D2 *R2D2 Priors in* **brms**

Description

Function used to set up R2D2(M2) priors in **brms**. The function does not evaluate its arguments – it exists purely to help set up the model.

Usage

```
R2D2(mean_R2 = 0.5, prec_R2 = 2, cons_D2 = 0.5, autoscale = TRUE, main = FALSE)
```

Arguments

mean_R2	Mean of the Beta prior on the coefficient of determination R^2.
prec_R2	Precision of the Beta prior on the coefficient of determination R^2.
cons_D2	Concentration vector of the Dirichlet prior on the variance decomposition parameters. Lower values imply more shrinkage.
autoscale	Logical; indicating whether the R2D2 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.
main	Logical (defaults to FALSE); only relevant if the R2D2 prior spans multiple parameter classes. In this case, only arguments given in the single instance where main is TRUE will be used. Arguments given in other instances of the prior will be ignored. See the Examples section below.

Details

The prior does not account for scale differences of the terms it is applied on. Accordingly, please make sure that all these terms have a comparable scale to ensure that shrinkage is applied properly.

Currently, the following classes support the R2D2(M2) prior: b (overall regression coefficients), sds (SDs of smoothing splines), sdgp (SDs of Gaussian processes), ar (autoregressive coefficients), ma (moving average coefficients), sderr (SD of latent residuals), sdcar (SD of spatial CAR structures), sd (SD of varying coefficients).

When the prior is only applied to parameter class b, it is equivalent to the original R2D2 prior (with Gaussian kernel). When the prior is also applied to other parameter classes, it is equivalent to the R2D2M2 prior.

Even when the R2D2(M2) prior is applied to multiple parameter classes at once, the concentration vector (argument cons_D2) has to be provided jointly in the the one instance of the prior where main = TRUE. The order in which the elements of concentration vector correspond to the classes' coefficients is the same as the order of the classes provided above.

References

Zhang, Y. D., Naughton, B. P., Bondell, H. D., & Reich, B. J. (2020). Bayesian regression using a prior on the model fit: The R2-D2 shrinkage prior. Journal of the American Statistical Association. https://arxiv.org/pdf/1609.00046

Aguilar J. E. & Bürkner P. C. (2022). Intuitive Joint Priors for Bayesian Linear Multilevel Models: The R2D2M2 prior. ArXiv preprint. https://arxiv.org/pdf/2208.07132

See Also

set_prior

Examples

```
set_prior(R2D2(mean_R2 = 0.8, prec_R2 = 10))

# specify the R2D2 prior across multiple parameter classes
set_prior(R2D2(mean_R2 = 0.8, prec_R2 = 10, main = TRUE), class = "b") +
  set_prior(R2D2(), class = "sd")
```

---

ranef.brmsfit *Extract Group-Level Estimates*

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),
  pars = NULL,
  groups = NULL,
  ...
)
```

Arguments

object	An object of class brmsfit.
summary	Should summary statistics 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.
pars	Optional names of coefficients to extract. By default, all coefficients are extracted.
groups	Optional names of grouping variables for which to extract effects.
...	Currently ignored.

Value

A list of 3D arrays (one per grouping factor). If summary is TRUE, the 1st dimension contains the factor levels, the 2nd dimension contains the summary statistics (see posterior_summary), and the 3rd dimension contains the group-level effects. If summary is FALSE, the 1st dimension contains the posterior draws, the 2nd dimension contains the factor levels, and the 3rd dimension contains the group-level effects.

Examples

```
## Not run:
fit <- brm(count ~ zAge + zBase * Trt + (1+Trt|visit),
           data = epilepsy, family = gaussian(), chains = 2)
ranef(fit)

## End(Not run)
```

---

read_csv_as_stanfit *Read CmdStan CSV files as a brms-formatted stanfit object*

Description

read_csv_as_stanfit is used internally to read CmdStan CSV files into a stanfit object that is consistent with the structure of the fit slot of a brmsfit object.

Usage

```
read_csv_as_stanfit(
  files,
  variables = NULL,
  sampler_diagnostics = NULL,
  model = NULL,
  exclude = "",
  algorithm = "sampling"
)
```

Arguments

files	Character vector of CSV files names where draws are stored.
variables	Character vector of variables to extract from the CSV files.
sampler_diagnostics	Character vector of sampler diagnostics to extract.
model	A compiled cmdstanr model object (optional). Provide this argument if you want to allow updating the model without recompilation.
exclude	Character vector of variables to exclude from the stanfit. Only used when `variables` is also specified.
algorithm	The algorithm with which the model was fitted. See `brm` for details.

Value

A stanfit object consistent with the structure of the `fit` slot of a brmsfit object.

Examples

```
## Not run:
# fit a model manually via cmdstanr
scode <- stancode(count ~ Trt, data = epilepsy)
sdata <- standata(count ~ Trt, data = epilepsy)
mod <- cmdstanr::cmdstan_model(cmdstanr::write_stan_file(scode))
stanfit <- mod$sample(data = sdata)

# feed the Stan model back into brms
fit <- brm(count ~ Trt, data = epilepsy, empty = TRUE, backend = 'cmdstanr')
fit$fit <- read_csv_as_stanfit(stanfit$output_files(), model = mod)
fit <- rename_pars(fit)
summary(fit)

## End(Not run)
```

recompile_model *Recompile Stan models in* brmsfit *objects*

Description

Recompile the Stan model inside a brmsfit object, if necessary. This does not change the model, it simply recreates the executable so that sampling is possible again.

Usage

```
recompile_model(x, recompile = NULL)
```

Arguments

x	An object of class brmsfit.
recompile	Logical, indicating whether the Stan model should be recompiled. If NULL (the default), recompile_model tries to figure out internally, if recompilation is necessary. Setting it to FALSE will cause recompile_model to always return the brmsfit object unchanged.

Value

A (possibly updated) brmsfit object.

reloo.brmsfit *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. Models for problematic observations can be run in parallel using the **future** package.

Usage

```
## S3 method for class 'brmsfit'
reloo(
  x,
  loo = NULL,
  k_threshold = 0.7,
  newdata = NULL,
  resp = NULL,
  check = TRUE,
  recompile = NULL,
  future_args = list(),
```

```
  ...
)

## S3 method for class 'loo'
reloo(x, fit, ...)

reloo(x, ...)
```

Arguments

x	An R object of class brmsfit or loo depending on the method.
loo	An R object of class loo. If NULL, brms will try to extract a precomputed loo object from the fitted model, added there via add_criterion.
k_threshold	The threshold at which Pareto k estimates are treated as problematic. Defaults to 0.7. See pareto_k_ids for more details.
newdata	An optional data.frame for which to evaluate predictions. If NULL (default), the original data of the model is used. NA values within factors (excluding grouping variables) 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. NA values within grouping variables are treated as a new level.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
check	Logical; If TRUE (the default), some checks check are performed if the loo object was generated from the brmsfit object passed to argument fit.
recompile	Logical, indicating whether the Stan model should be recompiled. This may be necessary if you are running reloo on another machine than the one used to fit the model.
future_args	A list of further arguments passed to future for additional control over parallel execution if activated.
...	Further arguments passed to update.brmsfit and log_lik.brmsfit.
fit	An R object of class brmsfit.

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.

By default, this method uses sample_new_levels = "gaussian" to sample parameter values for new grouping-factor levels (see also prepare_predictions). This default will fail for models with non-Gaussian group-level effects. In this case, we recommend setting sample_new_levels = "uncertainty".

Value

An object of the class `loo`.

Parallelization with multiple CPU cores

brms can make use of multiple CPU cores in parallel to speed up computations in various ways. For efficient use of the available resources it is recommended to only use parallelism to an extend such that the available physical CPUs are not oversubscribed. For example, when you have 8 CPU cores locally available, then you may consider to run 4 chains with 2 threads per chain for best performance if you happen to just run a single model. In case you run a simulation study which requires to run many times a given model, then neither chain nor within-chain parallelization is advisable as the computational resources are already exhausted by the simulation study and any further parallelization beyond the simulation study itself will in fact slow down the overall runtime. Please be aware that for historical reasons the nomenclature of the arguments is possibly confusing. The `cores` argument refers to running different chains in parallel and the within-chain parallelization will allocate for each chain as many threads as requested. The requested threads therefore increase the use of overall CPUs in a multiplicative way.

For more advanced parallelization (including beyond single model fits), **brms** also integrates with the **future** package. Importantly, this enables seamless integration with the **mirai** parallelization framework through the use of the **future.mirai** adapter. With **mirai** local and remote machines can be used in a fully transparent manner to the user. This includes the possibility to use large number of remote machines running in the context of a computer cluster, which are managed with queuing systems. Please refer to the section on distributed computing of `mirai::daemons`.

See Also

`loo`, `kfold`

Examples

```
## Not run:
fit1 <- brm(count ~ zAge + zBase * Trt + (1|patient),
            data = epilepsy, family = poisson())

# throws warning about some pareto k estimates being too high
(loo1 <- loo(fit1))

# no more warnings after reloo
(reloo1 <- reloo(fit1, loo = loo1, chains = 1))

## End(Not run)
```

`rename_pars` *Rename parameters in brmsfit objects*

Description

Rename parameters within the `stanfit` object after model fitting to ensure reasonable parameter names. This function is usually called automatically by `brm` and users will rarely be required to call it themselves.

Usage

```
rename_pars(x)
```

Arguments

`x`	A `brmsfit` object.

Details

Function `rename_pars` is a deprecated alias of `rename_pars`.

Value

A `brmsfit` object with adjusted parameter names.

Examples

```
## Not run:
# fit a model manually via rstan
scode <- stancode(count ~ Trt, data = epilepsy)
sdata <- standata(count ~ Trt, data = epilepsy)
stanfit <- rstan::stan(model_code = scode, data = sdata)

# feed the Stan model back into brms
fit <- brm(count ~ Trt, data = epilepsy, empty = TRUE)
fit$fit <- stanfit
fit <- rename_pars(fit)
summary(fit)

## End(Not run)
```

`residuals.brmsfit` *Posterior Draws of Residuals/Predictive Errors*

Description

This method is an alias of `predictive_error.brmsfit` with additional arguments for obtaining summaries of the computed draws.

Usage

```
## S3 method for class 'brmsfit'
residuals(
  object,
  newdata = NULL,
  re_formula = NULL,
  method = "posterior_predict",
  type = c("ordinary", "pearson"),
  resp = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  sort = FALSE,
  summary = TRUE,
  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. NA values within factors (excluding grouping variables) 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. NA values within grouping variables are treated as a new level.
re_formula	formula containing group-level effects to be considered in the prediction. If NULL (default), include all group-level effects; if NA or ~0, include no group-level effects.
method	Method used to obtain predictions. Can be set to "posterior_predict" (the default), "posterior_epred", or "posterior_linpred". For more details, see the respective function documentations.
type	The type of the residuals, either "ordinary" or "pearson". More information is provided under 'Details'.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
ndraws	Positive integer indicating how many posterior draws should be used. If NULL (the default) all draws are used. Ignored if draw_ids is not NULL.
draw_ids	An integer vector specifying the posterior draws to be used. If NULL (the default), all draws are used.
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).
summary	Should summary statistics 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.
...	Further arguments passed to prepare_predictions that control several aspects of data validation and prediction.

Details

Residuals of type 'ordinary' are of the form $R = Y - Yrep$, where Y is the observed and $Yrep$ is the predicted response. Residuals of type pearson are of the form $R = (Y - Yrep)/SD(Yrep)$, where $SD(Yrep)$ is an estimate of the standard deviation of $Yrep$.

Value

An array of predictive error/residual draws. If summary = FALSE the output resembles those of predictive_error.brmsfit. If summary = TRUE the output is an N x E matrix, where N is the number of observations and E denotes the summary statistics computed from the draws.

Examples

```
## Not run:
## fit a model
fit <- brm(rating ~ treat + period + carry + (1|subject),
           data = inhaler, cores = 2)

## extract residuals/predictive errors
res <- residuals(fit)
head(res)

## End(Not run)
```

---

restructure *Restructure Old R Objects*

Description

restructure is a generic function used to restructure old R objects to work with newer versions of the package that generated them. Its original use is within the **brms** package, but new methods for use with objects from other packages can be registered to the same generic.

Usage

```
restructure(x, ...)
```

Arguments

x	An object to be restructured. The object's class will determine which method to apply
...	Additional arguments to pass to the specific methods

Details

Usually the version of the package that generated the object will be stored somewhere in the object and this information will be used by the specific method to determine what transformations to apply. See `restructure.brmsfit` for the default method applied for **brms** models. You can view the available methods by typing: `methods(restructure)`

Value

An object of the same class as `x` compatible with the latest version of the package that generated it.

See Also

`restructure.brmsfit`

`restructure.brmsfit` *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

```
## S3 method for class 'brmsfit'
restructure(x, ...)
```

Arguments

x	An object of class `brmsfit`.
...	Currently ignored.

Details

If you are restructuring an old spline model (fitted with brms < 2.19.3) to avoid prediction inconsistencies between machines (see GitHub issue #1465), please make sure to `restructure` your model on the machine on which it was originally fitted.

Value

A brmsfit object compatible with the latest version of **brms**.

---

rows2labels *Convert Rows to Labels*

Description

Convert information in rows to labels for each row.

Usage

```
rows2labels(x, digits = 2, sep = " & ", incl_vars = TRUE, ...)
```

Arguments

x	A data.frame for which to extract labels.
digits	Minimal number of decimal places shown in the labels of numeric variables.
sep	A single character string defining the separator between variables used in the labels.
incl_vars	Indicates if variable names should be part of the labels. Defaults to TRUE.
...	Currently unused.

Value

A character vector of the same length as the number of rows of x.

See Also

make_conditions, conditional_effects

---

s *Defining smooths in* **brms** *formulas*

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. When using them, please cite the appropriate references obtained via `citation("mgcv")`.

brms uses the "random effects" parameterization of smoothing splines as explained in mgcv::gamm. A nice tutorial on this topic can be found in Pedersen et al. (2019). The answers provided in this Stan discourse post may also be helpful.

References

Pedersen, E. J., Miller, D. L., Simpson, G. L., & Ross, N. (2019). Hierarchical generalized additive models in ecology: an introduction with mgcv. PeerJ.

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(conditional_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(conditional_smooths(fit2), ask = FALSE)

## End(Not run)
```

---

sar *Spatial simultaneous autoregressive (SAR) structures*

Description

Set up an spatial simultaneous autoregressive (SAR) term in **brms**. The function does not evaluate its arguments – it exists purely to help set up a model with SAR terms.

Usage

```
sar(M, type = "lag")
```

Arguments

M	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). More information is provided in the 'Details' section.

Details

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. Currently, only families gaussian and student support SAR structures.

Value

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

See Also

autocor-terms

Examples

```
## Not run:
data(oldcol, package = "spdep")
fit1 <- brm(CRIME ~ INC + HOVAL + sar(COL.nb, type = "lag"),
            data = COL.OLD, data2 = list(COL.nb = COL.nb),
            chains = 2, cores = 2)
summary(fit1)
plot(fit1)

fit2 <- brm(CRIME ~ INC + HOVAL + sar(COL.nb, type = "error"),
            data = COL.OLD, data2 = list(COL.nb = COL.nb),
            chains = 2, cores = 2)
summary(fit2)
plot(fit2)

## End(Not run)
```

save_pars *Control Saving of Parameter Draws*

Description

Control which (draws of) parameters should be saved in a **brms** model. The output of this function is meant for usage in the save_pars argument of brm.

Usage

```
save_pars(group = TRUE, latent = FALSE, all = FALSE, manual = NULL)
```

Arguments

group	A flag to indicate if group-level coefficients for each level of the grouping factors should be saved (default is TRUE). Set to FALSE to save memory. Alternatively, group may also be a character vector naming the grouping factors for which to save draws of coefficients.
latent	A flag to indicate if draws of latent variables obtained by using me and mi terms should be saved (default is FALSE). Saving these draws allows to better use methods such as posterior_predict with the latent variables but leads to very large R objects even for models of moderate size and complexity. Alternatively, latent may also be a character vector naming the latent variables for which to save draws.
all	A flag to indicate if draws of all variables defined in Stan's parameters block should be saved (default is FALSE). Saving these draws is required in order to apply the certain methods such as bridge_sampler and bayes_factor.
manual	A character vector naming Stan variable names which should be saved. These names should match the variable names inside the Stan code before renaming. This feature is meant for power users only and will rarely be useful outside of very special cases.

Value

A list of class "save_pars".

Examples

```
## Not run:
# don't store group-level coefficients
fit <- brm(count ~ zAge + zBase * Trt + (1|patient),
           data = epilepsy, family = poisson(),
           save_pars = save_pars(group = FALSE))
variables(fit)

## 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,
  tag = "",
  check = TRUE
)

prior(prior, ...)

prior_(prior, ...)

prior_string(prior, ...)

empty_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 coefficient within the parameter class.
group	Grouping factor for group-level parameters.
resp	Name of the response variable. Only used in multivariate models.
dpar	Name of a distributional parameter. Only used in distributional models.
nlpar	Name of a non-linear parameter. Only used in non-linear models.
lb	Lower bound for parameter restriction. Currently only allowed for classes "b". Defaults to NULL, that is no restriction.
ub	Upper bound for parameter restriction. Currently only allowed for classes "b". Defaults to NULL, that is no restriction.

tag	Character to append to the lprior variable in the Stan code. Used for selectively checking sensitivity of priors in priorsense.
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 https://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** trys 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.

Below, we provide details for the individual parameter classes that you can set priors on. 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 default_prior.

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 x1 and x2 (i.e., y ~ x1 + x2 in formula syntax). Then, x1 and x2 have regression parameters b_x1 and b_x2 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 x1, and a unit student-t prior with 10 degrees of freedom for x2, 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 normal(0, 10) prior on the effect of x1 and a 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.

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, alpha 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 alpha at once.

The same goes for specifying priors for specific distributional parameters in the context of distributional regression, for example, set_prior("<prior>", coef = "x", dpar = "sigma"). For most other parameter classes (see below), you need to indicate non-linear and distributional parameters in the same way as shown here.

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. 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 ~ x1 + x2 + (1 + x1 | g). We see that the intercept as well as x1 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 2.5. This prior is used (a) to be only 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>").

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 correlations 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. Smoothing Splines

Smoothing splines are implemented in **brms** using the 'random effects' formulation as explained in gamm). Thus, each spline has its corresponding standard deviations 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), sderr (standard deviation of latent residuals in latent ARMA models), cosy (compound symmetry correlation), car (spatial conditional autoregression), as well as lagsar and errorsar (spatial simultaneous autoregression).

Priors can be defined by set_prior("<prior>", class = "ar") for ar and similar for other autocorrelation parameters. By default, ar and ma are bounded between -1 and 1; cosy, car, lagsar, and errorsar are bounded between 0 and 1. The default priors are flat over the respective definition areas.

7. Parameters of measurement error terms

Latent variables induced via measurement error me terms require both mean and standard deviation parameters, whose prior classes are named "meanme" and "sdme", respectively. If multiple latent variables are induced this way, their correlation matrix will be modeled as well and corresponding priors can be specified via the "corme" class. All of the above parameters have flat priors over their respective definition spaces by default.

8. 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.

9. 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 Student-t distribution. By default, nu has prior gamma(2, 0.1), which is close to a penalized complexity prior (see Stan prior choice Wiki), and a fixed lower bound of 1. Family negbinomial needs a shape parameter that has by default inv_gamma(0.4, 0.3) prior which is close to a penalized complexity prior (see Stan prior choice Wiki). Families gamma, weibull, and inverse.gaussian, 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.

10. Shrinkage priors

To reduce the danger of overfitting in models with many predictor terms fit on comparably sparse data, brms supports special shrinkage priors, namely the (regularized) horseshoe and the R2D2 prior. These priors can be applied on many parameter classes, either directly on the coefficient classes (e.g., class b), if directly setting priors on them is supported, or on the corresponding standard deviation hyperparameters (e.g., class sd) otherwise. Currently, the following classes support shrinkage priors: b (overall regression coefficients), sds (SDs of smoothing splines), sdgp (SDs of Gaussian processes), ar (autoregressive coefficients), ma (moving average coefficients), sderr (SD of latent residuals), sdcar (SD of spatial CAR structures), sd (SD of varying coefficients).

11. Fixing parameters to constants

Fixing parameters to constants is possible by using the constant function, for example, constant(1) to fix a parameter to 1. Broadcasting to vectors and matrices is done automatically.

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.
- empty_prior(): Create an empty brmsprior object.

See Also

default_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
default_prior(rating ~ treat + period + carry + (1|subject),
              data = inhaler, family = cumulative())

# define some priors
bprior <- 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
stancode(rating ~ treat + period + carry + (1|subject),
         data = inhaler, family = cumulative(),
         prior = bprior)

# use the horseshoe prior to model sparsity in regression coefficients
stancode(count ~ zAge + zBase * Trt,
         data = epilepsy, family = poisson(),
         prior = set_prior("horseshoe(3)"))

# fix certain priors to constants
bprior <- prior(constant(1), class = "b") +
  prior(constant(2), class = "b", coef = "zBase") +
  prior(constant(0.5), class = "sd")
stancode(count ~ zAge + zBase + (1 | patient),
              data = epilepsy, prior = bprior)

# pass priors to Stan without checking
prior <- prior_string("target += normal_lpdf(b[1] | 0, 1)", check = FALSE)
stancode(count ~ Trt, data = epilepsy, prior = prior)

# define priors in a vectorized manner
# useful in particular for categorical or multivariate models
set_prior("normal(0, 2)", dpar = c("muX", "muY", "muZ"))

# specify tags for different priors for sensitivity analysis using priorsense
# It is then possible to check the sensitivity when changing the priors with
# the same tag while leaving others the same
prior_cov <- prior(normal(0, 10), class = "b", tag = "covariates")
prior_trt <- prior(normal(0, 1), class = "b", coef = "Trt1", tag = "treatment")
```

```
stancode(count ~ Trt + zAge + zBase + (1 | patient),
         data = epilepsy, prior = c(prior_cov, prior_trt))
```

`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

`x, q`	Vector of quantiles.
`meanlog`	Vector of means.
`sdlog`	Vector of standard deviations.
`shift`	Vector of shifts.
`log`	Logical; If `TRUE`, values are returned on the log scale.
`lower.tail`	Logical; If `TRUE` (default), return P(X <= x). Else, return P(X > x) .
`log.p`	Logical; If `TRUE`, values are returned on the log scale.
`p`	Vector of probabilities.
`n`	Number of draws to sample 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 `mu`, standard deviation `sigma`, and skewness `alpha`.

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

x, q	Vector of quantiles.
mu	Vector of mean values.
sigma	Vector of standard deviation values.
alpha	Vector of skewness values.
xi	Optional vector of location values. If NULL (the default), will be computed internally.
omega	Optional vector of scale values. If NULL (the default), will be computed internally.
log	Logical; If TRUE, values are returned on the log scale.
lower.tail	Logical; If TRUE (default), return P(X <= x). Else, return P(X > x) .
log.p	Logical; If TRUE, values are returned on the log scale.
p	Vector of probabilities.
tol	Tolerance of the approximation used in the computation of quantiles.
n	Number of draws to sample from the distribution.

Details

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

---

stancode *Stan Code for Bayesian models*

Description

stancode is a generic function that can be used to generate Stan code for Bayesian models. Its original use is within the **brms** package, but new methods for use with objects from other packages can be registered to the same generic.

Usage

```
stancode(object, ...)

make_stancode(formula, ...)
```

Arguments

object	An object whose class will determine which method to apply. Usually, it will be some kind of symbolic description of the model form which Stan code should be generated.
...	Further arguments passed to the specific method.
formula	Synonym of object for use in make_stancode.

Details

See stancode.default for the default method applied for **brms** models. You can view the available methods by typing: methods(stancode) The make_stancode function is an alias of stancode.

Value

Usually, a character string containing the generated Stan code. For pretty printing, we recommend the returned object to be of class c("character", "brmsmodel").

See Also

stancode.default, stancode.brmsfit

Examples

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

---

stancode.brmsfit *Extract Stan code from* brmsfit *objects*

Description

Extract Stan code from a fitted **brms** model.

Usage

```
## S3 method for class 'brmsfit'
stancode(
  object,
  version = TRUE,
  regenerate = NULL,
  threads = NULL,
  backend = NULL,
  ...
)
```

Arguments

object	An object of class brmsfit.
version	Logical; indicates if the first line containing the **brms** version number should be included. Defaults to TRUE.
regenerate	Logical; indicates if the Stan code should be regenerated with the current **brms** version. By default, regenerate will be FALSE unless required to be TRUE by other arguments.
threads	Controls whether the Stan code should be threaded. See threading for details.

backend	Controls the Stan backend. See brm for details.
...	Further arguments passed to stancode if the Stan code is regenerated.

Value

Stan code for further processing.

`stancode.default` *Stan Code for* **brms** *Models*

Description

Generate Stan code for **brms** models

Usage

```
## Default S3 method:
stancode(
  object,
  data,
  family = gaussian(),
  prior = NULL,
  autocor = NULL,
  data2 = NULL,
  cov_ranef = NULL,
  sparse = NULL,
  sample_prior = "no",
  stanvars = NULL,
  stan_funs = NULL,
  knots = NULL,
  drop_unused_levels = TRUE,
  threads = getOption("brms.threads", NULL),
  normalize = getOption("brms.normalize", TRUE),
  save_model = NULL,
  ...
)
```

Arguments

object	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 data.frame (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 link 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 brmsprior objects created by set_prior or related functions and combined using the c method or the + operator. See also default_prior for more help.

autocor — (Deprecated) 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 NULL, corresponding to no correlations. In multivariate models, autocor might also be a list of autocorrelation structures. It is now recommend to specify autocorrelation terms directly within formula. See brmsformula for more details.

data2 — A named list of objects containing data, which cannot be passed via argument data. Required for some objects used in autocorrelation structures to specify dependency structures as well as for within-group covariance matrices.

cov_ranef — (Deprecated) 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. It is now recommended to specify those matrices in the formula interface using the gr and related functions. See vignette("brms_phylogenetics") for more details.

sparse — (Deprecated) 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. It is now recommended to use the sparse argument of brmsformula and related functions.

sample_prior — Indicate if draws from priors should be drawn additionally to the posterior draws. Options are "no" (the default), "yes", and "only". Among others, these draws can be used to calculate Bayes factors for point hypotheses via hypothesis. Please note that improper priors are not sampled, including the default improper priors used by brm. See set_prior on how to set (proper) priors. Please also note that prior draws for the overall intercept are not obtained by default for technical reasons. See brmsformula how to obtain prior draws for the intercept. If sample_prior is set to "only", draws are drawn solely from the priors ignoring the likelihood, which allows among others to generate draws from the prior predictive distribution. In this case, all parameters must have proper priors.

stanvars — An optional stanvars object generated by function stanvar to define additional variables for use in **Stan**'s program blocks.

stan_funs — (Deprecated) An optional character string containing self-defined **Stan** functions, which will be included in the functions block of the generated **Stan** code. It is now recommended to use the stanvars argument for this purpose instead.

knots — Optional list containing user specified knot values to be used for basis construction of smoothing terms. See gamm for more details.

drop_unused_levels — Should unused factors levels in the data be dropped? Defaults to TRUE.

threads — Number of threads to use in within-chain parallelization. For more control over the threading process, threads may also be a brmsthreads object created by threading. Within-chain parallelization is experimental! We recommend its use only if you are experienced with Stan's reduce_sum function and have a slow running model that cannot be sped up by any other means. Can be set globally for the current R session via the "brms.threads" option (see options).

normalize — Logical. Indicates whether normalization constants should be included in the Stan code (defaults to TRUE). Setting it to FALSE requires Stan version >= 2.25 to work. If FALSE, sampling efficiency may be increased but some post processing functions such as bridge_sampler will not be available. Can be controlled globally for the current R session via the 'brms.normalize' option.

save_model — Either NULL or a character string. In the latter case, the model's Stan code is saved via cat in a text file named after the string supplied in save_model.

... — Other arguments for internal usage only.

Value

A character string containing the fully commented **Stan** code to fit a **brms** model. It is of class c("character", "brmsmodel") to facilitate pretty printing.

Examples

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

stancode(count ~ zAge + zBase * Trt + (1|patient),
         data = epilepsy, family = "poisson")
```

---

standata *Stan data for Bayesian models*

Description

standata is a generic function that can be used to generate data for Bayesian models to be passed to Stan. Its original use is within the **brms** package, but new methods for use with objects from other packages can be registered to the same generic.

Usage

```
standata(object, ...)

make_standata(formula, ...)
```

Arguments

object	A formula object whose class will determine which method will be used. A symbolic description of the model to be fitted.
...	Further arguments passed to the specific method.
formula	Synonym of object for use in make_standata.

Details

See standata.default for the default method applied for **brms** models. You can view the available methods by typing methods(standata). The make_standata function is an alias of standata.

Value

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

See Also

standata.default, standata.brmsfit

Examples

```
sdata1 <- standata(rating ~ treat + period + carry + (1|subject),
                   data = inhaler, family = "cumulative")
str(sdata1)
```

---

standata.brmsfit *Extract data passed to Stan from* brmsfit *objects*

Description

Extract all data that was used by Stan to fit a **brms** model.

Usage

```
## S3 method for class 'brmsfit'
standata(
  object,
  newdata = NULL,
  re_formula = NULL,
  newdata2 = NULL,
  new_objects = NULL,
  incl_autocor = TRUE,
  ...
)
```

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. NA values within factors (excluding grouping variables) 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. NA values within grouping variables are treated as a new level.
re_formula	formula containing group-level effects to be considered in the prediction. If NULL (default), include all group-level effects; if NA or ~0, include no group-level effects.
newdata2	A named list of objects containing new data, which cannot be passed via argument newdata. Required for some objects used in autocorrelation structures, or stanvars.
new_objects	Deprecated alias of newdata2.
incl_autocor	A flag indicating if correlation structures originally specified via autocor should be included in the predictions. Defaults to TRUE.
...	More arguments passed to standata.default. and validate_newdata.

Value

A named list containing the data passed to Stan.

---

standata.default *Data for* **brms** *Models*

Description

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

Usage

```
## Default S3 method:
standata(
  object,
  data,
  family = gaussian(),
  prior = NULL,
  autocor = NULL,
  data2 = NULL,
  cov_ranef = NULL,
  sample_prior = "no",
  stanvars = NULL,
  threads = getOption("brms.threads", NULL),
  knots = NULL,
  drop_unused_levels = TRUE,
  ...
)
```

Arguments

object	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 data.frame (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 link 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 brmsprior objects created by set_prior or related functions and combined using the c method or the + operator. See also default_prior for more help.
autocor	(Deprecated) 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 NULL, corresponding to no correlations. In multivariate models, autocor might also be a list of autocorrelation structures. It is now recommend to specify autocorrelation terms directly within formula. See brmsformula for more details.
data2	A named list of objects containing data, which cannot be passed via argument data. Required for some objects used in autocorrelation structures to specify dependency structures as well as for within-group covariance matrices.
cov_ranef	(Deprecated) 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. It is now recommended to specify those matrices in the formula interface using the gr and related functions. See vignette("brms_phylogenetics") for more details.
sample_prior	Indicate if draws from priors should be drawn additionally to the posterior draws. Options are "no" (the default), "yes", and "only". Among others, these draws can be used to calculate Bayes factors for point hypotheses via hypothesis. Please note that improper priors are not sampled, including the default improper priors used by brm. See set_prior on how to set (proper) priors. Please also note that prior draws for the overall intercept are not obtained by default for technical reasons. See brmsformula how to obtain prior draws for the intercept. If sample_prior is set to "only", draws are drawn solely from the priors ignoring the likelihood, which allows among others to generate draws from the prior predictive distribution. In this case, all parameters must have proper priors.
stanvars	An optional stanvars object generated by function stanvar to define additional variables for use in **Stan**'s program blocks.

threads — Number of threads to use in within-chain parallelization. For more control over the threading process, threads may also be a brmsthreads object created by threading. Within-chain parallelization is experimental! We recommend its use only if you are experienced with Stan's reduce_sum function and have a slow running model that cannot be sped up by any other means. Can be set globally for the current R session via the "brms.threads" option (see options).

knots — Optional list containing user specified knot values to be used for basis construction of smoothing terms. See gamm for more details.

drop_unused_levels — Should unused factors levels in the data be dropped? Defaults to TRUE.

... — Other arguments for internal use.

Value

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

Examples

```
sdata1 <- standata(rating ~ treat + period + carry + (1|subject),
                   data = inhaler, family = "cumulative")
str(sdata1)

sdata2 <- standata(count ~ zAge + zBase * Trt + (1|patient),
                   data = epilepsy, family = "poisson")
str(sdata2)
```

---

stanvar *User-defined variables passed to Stan*

Description

Prepare user-defined variables to be passed to one of Stan's program blocks. This is primarily useful for defining more complex priors, for refitting models without recompilation despite changing priors, or for defining custom Stan functions.

Usage

```
stanvar(
  x = NULL,
  name = NULL,
  scode = NULL,
  block = "data",
  position = "start",
  pll_args = NULL
)
```

Arguments

x	An R object containing data to be passed to Stan. Only required if block = 'data' and ignored otherwise.
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	Line of Stan code to define the variable in Stan language. If block = 'data', the Stan code is inferred based on the class of x by default.
block	Name of one of Stan's program blocks in which the variable should be defined. Can be 'data', 'tdata' (transformed data), 'parameters', 'tparameters' (transformed parameters), 'model', 'likelihood' (part of the model block where the likelihood is given), 'genquant' (generated quantities) or 'functions'.
position	Name of the position within the block where the Stan code should be placed. Currently allowed are 'start' (the default) and 'end' of the block.
pll_args	Optional Stan code to be put into the header of partial_log_lik functions. This ensures that the variables specified in scode can be used in the likelihood even when within-chain parallelization is activated via threading.

Details

The stanvar function is not vectorized. Instead, multiple stanvars objects can be added together via + (see Examples).

Special attention is necessary when using stanvars to inject code into the 'likelihood' block while having threading activated. In this case, your custom Stan code may need adjustments to ensure correct observation indexing. Please investigate the generated Stan code via stancode to see which adjustments are necessary in your case.

Value

An object of class stanvars.

Examples

```
bprior <- prior(normal(mean_intercept, 10), class = "Intercept")
stanvars <- stanvar(5, name = "mean_intercept")
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;")
stancode(count ~ Trt + zBase, epilepsy,
         prior = bprior, stanvars = stanvars)

# define a hierachical prior on the regression coefficients
bprior <- set_prior("normal(0, tau)", class = "b") +
  set_prior("target += normal_lpdf(tau | 0, 10)", check = FALSE)
stanvars <- stanvar(scode = "real<lower=0> tau;",
```

```
                    block = "parameters")
stancode(count ~ Trt + zBase, epilepsy,
         prior = bprior, stanvars = stanvars)

# ensure that 'tau' is passed to the likelihood of a threaded model
# not necessary for this example but may be necessary in other cases
stanvars <- stanvar(scode = "real<lower=0> tau;",
                    block = "parameters", pll_args = "real tau")
stancode(count ~ Trt + zBase, epilepsy,
         stanvars = stanvars, threads = threading(2))
```

`StudentT` *The Student-t Distribution*

Description

Density, distribution function, quantile function and random generation for the Student-t distribution with location `mu`, scale `sigma`, 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, lower.tail = TRUE, log.p = FALSE)

rstudent_t(n, df, mu = 0, sigma = 1)
```

Arguments

`x`	Vector of quantiles.
`df`	Vector of degrees of freedom.
`mu`	Vector of location values.
`sigma`	Vector of scale values.
`log`	Logical; If `TRUE`, values are returned on the log scale.
`q`	Vector of quantiles.
`lower.tail`	Logical; If `TRUE` (default), return P(X <= x). Else, return P(X > x) .
`log.p`	Logical; If `TRUE`, values are returned on the log scale.
`p`	Vector of probabilities.
`n`	Number of draws to sample from the distribution.

Details

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

See Also

TDist

---

summary.brmsfit *Create a summary of a fitted model represented by a* brmsfit *object*

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,
  robust = FALSE,
  mc_se = FALSE,
  ...
)
```

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.
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.
mc_se	Logical; Indicating if the uncertainty in Estimate caused by the MCMC sampling should be shown in the summary. Defaults to FALSE.
...	Other potential arguments

Details

The convergence diagnostics Rhat, Bulk_ESS, and Tail_ESS are described in detail in Vehtari et al. (2020).

References

Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner (2020). Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC. *Bayesian Analysis*. 1–28. dpi:10.1214/20-BA1221

theme_black *(Deprecated) Black Theme for* **ggplot2** *Graphics*

Description

A black theme for ggplot graphics inspired by a blog post of Jon Lefcheck (https://jonlefcheck.net/2013/03/11/black-theme-for-ggplot2-2/).

Usage

```
theme_black(base_size = 12, base_family = "")
```

Arguments

base_size	base font size
base_family	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
ggplot2::theme_set(theme_black())

# change default bayesplot color scheme
bayesplot::color_scheme_set("viridisC")

# fit a simple model
fit <- brm(count ~ zAge + zBase * Trt + (1|patient),
           data = epilepsy, family = poisson(), chains = 2)
summary(fit)

# create various plots
plot(marginal_effects(fit), ask = FALSE)
pp_check(fit)
mcmc_plot(fit, type = "hex", variable = c("b_Intercept", "b_Trt1"))

## End(Not run)
```

theme_default *Default* **bayesplot** *Theme for* **ggplot2** *Graphics*

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.

threading *Threading in Stan*

Description

Use threads for within-chain parallelization in **Stan** via the **brms** interface. Within-chain parallelization is experimental! We recommend its use only if you are experienced with Stan's `reduce_sum` function and have a slow running model that cannot be sped up by any other means.

Usage

```
threading(threads = NULL, grainsize = NULL, static = FALSE, force = FALSE)
```

Arguments

`threads`	Number of threads to use in within-chain parallelization.
`grainsize`	Number of observations evaluated together in one chunk on one of the CPUs used for threading. If `NULL` (the default), `grainsize` is currently chosen as `max(100, N / (2 * threads))`, where `N` is the number of observations in the data. This default is experimental and may change in the future without prior notice.
`static`	Logical. Apply the static (non-adaptive) version of `reduce_sum`? Defaults to `FALSE`. Setting it to `TRUE` is required to achieve exact reproducibility of the model results (if the random seed is set as well).
`force`	Logical. Defaults to `FALSE`. If `TRUE`, this will force the Stan model to compile with threading enabled without altering the Stan code generated by brms. This can be useful if your own custom Stan functions use threading internally.

Details

The adaptive scheduling procedure used by `reduce_sum` will prevent the results to be exactly reproducible even if you set the random seed. If you need exact reproducibility, you have to set argument `static = TRUE` which may reduce efficiency a bit.

To ensure that chunks (whose size is defined by `grainsize`) require roughly the same amount of computing time, we recommend storing observations in random order in the data. At least, please avoid sorting observations after the response values. This is because the latter often cause variations in the computing time of the pointwise log-likelihood, which makes up a big part of the parallelized code.

Value

A `brmsthreads` object which can be passed to the `threads` argument of `brm` and related functions.

Examples

```
## Not run:
# this model just serves as an illustration
# threading may not actually speed things up here
fit <- brm(count ~ zAge + zBase * Trt + (1|patient),
           data = epilepsy, family = negbinomial(),
           chains = 1, threads = threading(2, grainsize = 100),
           backend = "cmdstanr")
summary(fit)

## End(Not run)
```

---

unstr *Set up UNSTR correlation structures*

Description

Set up an unstructured (UNSTR) correlation term in **brms**. The function does not evaluate its arguments – it exists purely to help set up a model with UNSTR terms.

Usage

```
unstr(time, gr)
```

Arguments

`time`	An optional time variable specifying the time ordering of the observations. By default, the existing order of the observations in the data is used.
`gr`	An optional grouping variable. If specified, the correlation structure is assumed to apply only to observations within the same grouping level.

Value

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

See Also

autocor-terms

Examples

```
## Not run:
# add an unstructured correlation matrix for visits within the same patient
fit <- brm(count ~ Trt + unstr(visit, patient), data = epilepsy)
summary(fit)

## End(Not run)
```

---

update.brmsfit *Update* **brms** *models*

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. Data-dependent default priors will not be updated automatically.
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

When updating a brmsfit created with the **cmdstanr** backend in a different R session, a recompilation will be triggered because by default, **cmdstanr** writes the model executable to a temporary directory. To avoid that, set option "cmdstanr_write_stan_file_dir" to a nontemporary path of your choice before creating the original brmsfit (see section 'Examples' below).

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(), init = "0",
               prior = set_prior("normal(0,5)"))
summary(fit4)

## to avoid a recompilation when updating a 'cmdstanr'-backend fit in a fresh
## R session, set option 'cmdstanr_write_stan_file_dir' before creating the
## initial 'brmsfit'
## CAUTION: the following code creates some files in the current working
## directory: two 'model_<hash>.stan' files, one 'model_<hash>(.exe)'
## executable, and one 'fit_cmdstanr_<some_number>.rds' file
set.seed(7)
fname <- paste0("fit_cmdstanr_", sample.int(.Machine$integer.max, 1))
options(cmdstanr_write_stan_file_dir = getwd())
fit_cmdstanr <- brm(rate ~ conc + state,
                    data = Puromycin,
                    backend = "cmdstanr",
                    file = fname)
# now restart the R session and run the following (after attaching 'brms')
set.seed(7)
fname <- paste0("fit_cmdstanr_", sample.int(.Machine$integer.max, 1))
fit_cmdstanr <- brm(rate ~ conc + state,
                    data = Puromycin,
                    backend = "cmdstanr",
                    file = fname)
upd_cmdstanr <- update(fit_cmdstanr,
                       formula. = rate ~ conc)

## End(Not run)
```

`update.brmsfit_multiple`

Update **brms** *models based on multiple data sets*

Description

This method allows to update an existing brmsfit_multiple object.

Usage

```
## S3 method for class 'brmsfit_multiple'
update(object, formula., newdata = NULL, data2 = NULL, ...)
```

Arguments

object	An object of class brmsfit_multiple.
formula.	Changes to the formula; for details see update.formula and brmsformula.
newdata	List of data.frames to update the model with new data. Currently required even if the original data should be used.
data2	A *list* of named lists each of which will be used to fit a separate model. Each of the named lists contains objects representing data which cannot be passed via argument data (see brm for examples). The length of the outer list should match the length of the list passed to the data argument. Currently required even if the original data should be used.
...	Other arguments passed to update.brmsfit and brm_multiple.

Examples

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

# initially fit the model
fit_imp1 <- brm_multiple(bmi ~ age + hyp + chl, data = imp, chains = 1)
summary(fit_imp1)

# update the model using fewer predictors
fit_imp2 <- update(fit_imp1, formula. = . ~ hyp + chl, newdata = imp)
summary(fit_imp2)

## End(Not run)
```

---

update_adterms *Update Formula Addition Terms*

Description

Update additions terms used in formulas of **brms**. See addition-terms for details.

Usage

```
update_adterms(formula, adform, action = c("update", "replace"))
```

Arguments

formula	Two-sided formula to be updated.
adform	One-sided formula containing addition terms to update formula with.
action	Indicates what should happen to the existing addition terms in formula. If "update" (the default), old addition terms that have no corresponding term in adform will be kept. If "replace", all old addition terms will be removed.

Value

An object of class formula.

Examples

```
form <- y | trials(size) ~ x
update_adterms(form, ~ trials(10))
update_adterms(form, ~ weights(w))
update_adterms(form, ~ weights(w), action = "replace")
update_adterms(y ~ x, ~ trials(10))
```

---

validate_newdata *Validate New Data*

Description

Validate new data passed to post-processing methods of **brms**. Unless you are a package developer, you will rarely need to call validate_newdata directly.

Usage

```
validate_newdata(
  newdata,
  object,
  re_formula = NULL,
  allow_new_levels = FALSE,
  newdata2 = NULL,
  resp = NULL,
  check_response = TRUE,
  incl_autocor = TRUE,
  group_vars = NULL,
  req_vars = NULL,
  ...
)
```

Arguments

newdata	A data.frame containing new data to be validated.
object	A brmsfit object.
re_formula	formula containing group-level effects to be considered in the prediction. If NULL (default), include all group-level effects; if NA or ~0, include no group-level effects.
allow_new_levels	A flag indicating if new levels of group-level effects are allowed (defaults to FALSE). Only relevant if newdata is provided.
newdata2	A named list of objects containing new data, which cannot be passed via argument newdata. Required for some objects used in autocorrelation structures, or stanvars.
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
check_response	Logical; Indicates if response variables should be checked as well. Defaults to TRUE.
incl_autocor	A flag indicating if correlation structures originally specified via autocor should be included in the predictions. Defaults to TRUE.
group_vars	Optional names of grouping variables to be validated. Defaults to all grouping variables in the model.
req_vars	Optional names of variables required in newdata. If NULL (the default), all variables in the original data are required (unless ignored for some other reason).
...	Currently ignored.

Value

A validated 'data.frame' based on newdata.

---

validate_prior *Validate Prior for* **brms** *Models*

Description

Validate priors supplied by the user. Return a complete set of priors for the given model, including default priors.

Usage

```
validate_prior(
  prior,
  formula,
  data,
  family = gaussian(),
  sample_prior = "no",
```

```
  data2 = NULL,
  knots = NULL,
  drop_unused_levels = TRUE,
  ...
)
```

Arguments

prior	One or more brmsprior objects created by set_prior or related functions and combined using the c method or the + operator. See also default_prior for more help.
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 data.frame (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 link 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.
sample_prior	Indicate if draws from priors should be drawn additionally to the posterior draws. Options are "no" (the default), "yes", and "only". Among others, these draws can be used to calculate Bayes factors for point hypotheses via hypothesis. Please note that improper priors are not sampled, including the default improper priors used by brm. See set_prior on how to set (proper) priors. Please also note that prior draws for the overall intercept are not obtained by default for technical reasons. See brmsformula how to obtain prior draws for the intercept. If sample_prior is set to "only", draws are drawn solely from the priors ignoring the likelihood, which allows among others to generate draws from the prior predictive distribution. In this case, all parameters must have proper priors.
data2	A named list of objects containing data, which cannot be passed via argument data. Required for some objects used in autocorrelation structures to specify dependency structures as well as for within-group covariance matrices.
knots	Optional list containing user specified knot values to be used for basis construction of smoothing terms. See gamm for more details.
drop_unused_levels	Should unused factors levels in the data be dropped? Defaults to TRUE.
...	Other arguments for internal usage only.

Value

An object of class brmsprior.

See Also

default_prior, set_prior.

Examples

```
prior1 <- prior(normal(0,10), class = b) +
  prior(cauchy(0,2), class = sd)
validate_prior(prior1, count ~ zAge + zBase * Trt + (1|patient),
               data = epilepsy, family = poisson())
```

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

x	An object of class brmsfit.
sigma	Ignored (included for compatibility with VarCorr).
summary	Should summary statistics 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

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.

Examples

```
## Not run:
fit <- brm(count ~ zAge + zBase * Trt + (1+Trt|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, pars = NULL, ...)
```

Arguments

object	An object of class brmsfit.
correlation	Logical; if FALSE (the default), compute the covariance matrix, if TRUE, compute the correlation matrix.
pars	Optional names of coefficients to extract. By default, all coefficients are extracted.
...	Currently ignored.

Details

Estimates are obtained by calculating the maximum likelihood covariances (correlations) of the posterior draws.

Value

covariance or correlation matrix of population-level parameters

Examples

```
## Not run:
fit <- brm(count ~ zAge + zBase * Trt + (1+Trt|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 `mu`, and precision `kappa`.

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

`x`, `q`	Vector of quantiles between `-pi` and `pi`.
`mu`	Vector of location values.
`kappa`	Vector of precision values.
`log`	Logical; If `TRUE`, values are returned on the log scale.
`lower.tail`	Logical; If `TRUE` (default), return P(X <= x). Else, return P(X > x) .
`log.p`	Logical; If `TRUE`, values are returned on the log scale.
`acc`	Accuracy of numerical approximations.
`n`	Number of draws to sample from the distribution.

Details

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

`waic.brmsfit` *Widely Applicable Information Criterion (WAIC)*

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

`x`	A `brmsfit` object.
`...`	More `brmsfit` objects or further arguments passed to the underlying post-processing functions. In particular, see `prepare_predictions` for further supported arguments.
`compare`	A flag indicating if the information criteria of the models should be compared to each other via `loo_compare`.
`resp`	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
`pointwise`	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, `pointwise = TRUE` is the way to go.
`model_names`	If `NULL` (the default) will use model names derived from deparsing the call. Otherwise will use the passed values as model names.

Details

See `loo_compare` for details on model comparisons. For `brmsfit` objects, `WAIC` is an alias of `waic`. Use method `add_criterion` to store information criteria in the fitted model object for later usage.

Value

If just one object is provided, an object of class `loo`. If multiple objects are provided, an object of class `loolist`.

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)
(waic1 <- waic(fit1))

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

# compare both models
loo_compare(waic1, waic2)

## End(Not run)
```

`Wiener` *The Wiener Diffusion Model Distribution*

Description

Density function and random generation for the Wiener diffusion model distribution with boundary separation `alpha`, non-decision time `tau`, bias `beta` and drift rate `delta`.

Usage

```
dwiener(
  x,
  alpha,
  tau,
  beta,
  delta,
  resp = 1,
  log = FALSE,
```

```
  backend = getOption("wiener_backend", "Rwiener")
)

rwiener(
  n,
  alpha,
  tau,
  beta,
  delta,
  types = c("q", "resp"),
  backend = getOption("wiener_backend", "Rwiener")
)
```

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.
backend	Name of the package to use as backend for the computations. Either "Rwiener" (the default) or "rtdists". Can be set globally for the current R session via the "wiener_backend" option (see options).
n	Number of draws to sample 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** or **rtdists** package (depending on the chosen backend). See `vignette("brms_families")` for details on the parameterization.

See Also

wienerdist, Diffusion

ZeroInflated *Zero-Inflated Distributions*

Description

Density and distribution functions for zero-inflated distributions.

Usage

```
dzero_inflated_poisson(x, lambda, zi, log = FALSE)

pzero_inflated_poisson(q, lambda, zi, lower.tail = TRUE, log.p = FALSE)

dzero_inflated_negbinomial(x, mu, shape, zi, log = FALSE)

pzero_inflated_negbinomial(q, mu, shape, zi, lower.tail = TRUE, log.p = FALSE)

dzero_inflated_binomial(x, size, prob, zi, log = FALSE)

pzero_inflated_binomial(q, size, prob, zi, lower.tail = TRUE, log.p = FALSE)

dzero_inflated_beta_binomial(x, size, mu, phi, zi, log = FALSE)

pzero_inflated_beta_binomial(
  q,
  size,
  mu,
  phi,
  zi,
  lower.tail = TRUE,
  log.p = FALSE
)

dzero_inflated_beta(x, shape1, shape2, zi, log = FALSE)

pzero_inflated_beta(q, shape1, shape2, zi, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x	Vector of quantiles.
zi	zero-inflation probability
log	Logical; If TRUE, values are returned on the log scale.
q	Vector of quantiles.
lower.tail	Logical; If TRUE (default), return P(X <= x). Else, return P(X > x) .
log.p	Logical; If TRUE, values are returned on the log scale.

`mu`, `lambda`	location parameter
`shape`, `shape1`, `shape2`	shape parameter
`size`	number of trials
`prob`	probability of success on each trial
`phi`	precision parameter

Details

The density of a zero-inflated distribution can be specified as follows. If $x = 0$ set $f(x) = \theta + (1 - \theta) * g(0)$. Else set $f(x) = (1 - \theta) * g(x)$, where $g(x)$ is the density of the non-zero-inflated part.

Index

∗ **datasets**
epilepsy, 96
inhaler, 118
kidney, 130
loss, 146

acat *(*brmsfamily*)*, 34
acformula, *19*
acformula *(*brmsformula-helpers*)*, 51
add_criterion, 11, *12*, *64*, *137*, *138*, *140*, *206*, *248*
add_ic, *64*
add_ic *(*add_loo*)*, 12
add_ic<- *(*add_loo*)*, 12
add_loo, 12
add_rstan_model, 13
add_waic *(*add_loo*)*, 12
addition-terms, 8
and *(*draws-index-brms*)*, 93
ar, 13, *15*, *19*, *73*, *148*
arma, *14*, 14, *19*, *74*, *148*
as.array.brmsfit *(*as.data.frame.brmsfit*)*, 16
as.brmsprior, 15
as.data.frame, *16*, *178*
as.data.frame.brmsfit, 16
as.matrix.brmsfit, *172*
as.matrix.brmsfit *(*as.data.frame.brmsfit*)*, 16
as.mcmc *(*as.mcmc.brmsfit*)*, 17
as.mcmc.brmsfit, 17
as_draws, *17*, *178*
as_draws *(*draws-brms*)*, 92
as_draws_∗, *17*, *177*
as_draws_array *(*draws-brms*)*, 92
as_draws_df *(*draws-brms*)*, 92
as_draws_list *(*draws-brms*)*, 92
as_draws_matrix *(*draws-brms*)*, 92
as_draws_rvars *(*draws-brms*)*, 92
asym_laplace *(*brmsfamily*)*, 34
AsymLaplace, 18
attributes, *200*
autocor *(*autocor.brmsfit*)*, 20
autocor-terms, 19
autocor.brmsfit, 20

bayes_factor, *7*, *24*, *117*, *182*
bayes_factor *(*bayes_factor.brmsfit*)*, 20
bayes_factor.brmsfit, 20
bayes_R2 *(*bayes_R2.brmsfit*)*, 21
bayes_R2.brmsfit, 21
bayesplot, *7*, *186*
bernoulli *(*brmsfamily*)*, 34
Beta *(*brmsfamily*)*, 34
beta_binomial *(*brmsfamily*)*, 34
BetaBinomial, 23
bf *(*brmsformula*)*, 41
bf-helpers *(*brmsformula-helpers*)*, 51
bridge_sampler, *20*, *21*, *29*, *182*, *228*
bridge_sampler *(*bridge_sampler.brmsfit*)*, 24
bridge_sampler.brmsfit, 24
bridge_sampler.stanfit, *24*
bridgesampling::bayes_factor, *21*
bridgesampling::bridge_sampler, *24*
bridgesampling::post_prob, *182*
brm, *7*, *8*, *10*, 25, *40–42*, *55*, *57*, *59*, *80*, *113*, *125*, *137*, *141*, *154*, *204*, *208*, *215*, *219*, *220*, *226*, *239*, *241*
brm_multiple, *44*, 56, *241*
brms, *32*, *40*, *41*
brms *(*brms-package*)*, 6
brms-package, 6
brmsfamily, *8*, *26*, *30*, *32*, 34, *41*, *48*, *57*, *83*, *85*, *88*, *154*, *227*, *231*, *244*
brmsfit, *8*, *32*
brmsfit *(*brmsfit-class*)*, 40
brmsfit-class, 40
brmsfit_needs_refit, *30*, *58*

brmsformula, *7, 8*, *10*, *19*, *26, 27*, *30*, *32*, *39–41*, 41, *51, 52*, *55–57*, *83–85*, *88*, *110, 111*, *151, 152*, *154*, *156*, *159*, *162, 163*, *213*, *217*, *226, 227*, *231*, *239*, *241*, *244*
brmsformula-helpers, 51
brmshypothesis, 53, *117*
brmsprior, *40*
brmsprior *(*set_prior*)*, 216
brmsprior-class *(*set_prior*)*, 216
brmsterms, 54, *123, 124*

car, *19*, 60, *76*
cat, *30*, *228*
cat *(*addition-terms*)*, 8
categorical *(*brmsfamily*)*, 34
cbind, *157*, *162*
cens *(*addition-terms*)*, 8
chains, *(*draws-index-brms*)*, 93
coef.brmsfit, 61, *116*
combine_models, *58*, 63
compare_ic, 63
conditional_effects, *7*, *71*, *148, 149*, *212*
conditional_effects *(*conditional_effects.brmsfit*)*, 64
conditional_effects.brmsfit, 64
conditional_smooths *(*conditional_smooths.brmsfit*)*, 69
conditional_smooths.brmsfit, 69
constant, 72
control_params, 73
cor_ar, 73, *75, 76*
cor_arma, *74*, 74, *76*, *79*
cor_arma-class *(*cor_arma*)*, 74
cor_brms, *19*, *27*, *41, 42*, *57*, 75, *88*, *227*, *231*
cor_brms-class *(*cor_brms*)*, 75
cor_car, *76*, 76
cor_cosy, 77
cor_cosy-class *(*cor_cosy*)*, 77
cor_errorsar *(*cor_sar*)*, 80
cor_fixed, *76*, 78
cor_icar *(*cor_car*)*, 76
cor_lagsar *(*cor_sar*)*, 80
cor_ma, *75, 76*, 79
cor_sar, *76*, 80
cosy, *19*, *77*, 81
cov_fixed *(*cor_fixed*)*, 78
cox *(*brmsfamily*)*, 34
cratio *(*brmsfamily*)*, 34
create_priorsense_data.brmsfit, 82
cs, 83
cse *(*cs*)*, 83
cumulative *(*brmsfamily*)*, 34
custom_family, *34*, 83
customfamily, *40*
customfamily *(*custom_family*)*, 83
cv_varsel, *106*

dasym_laplace *(*AsymLaplace*)*, 18
dbeta_binomial *(*BetaBinomial*)*, 23
ddirichlet *(*Dirichlet*)*, 91
dec *(*addition-terms*)*, 8
default_prior, *27*, *31*, *57*, 86, *89*, *217*, *221*, *227*, *231*, *244, 245*
default_prior.default, *87*, 87
density, *89*
density_ratio, 89
dexgaussian *(*ExGaussian*)*, 97
dfrechet *(*Frechet*)*, 103
dgen_extreme_value *(*GenExtremeValue*)*, 104
dhurdle_gamma *(*Hurdle*)*, 114
dhurdle_lognormal *(*Hurdle*)*, 114
dhurdle_negbinomial *(*Hurdle*)*, 114
dhurdle_poisson *(*Hurdle*)*, 114
diagnostic-quantities, 90
Diffusion, *250*
dinv_gaussian *(*InvGaussian*)*, 120
Dirichlet, 91
dirichlet *(*brmsfamily*)*, 34
dirichlet_multinomial *(*brmsfamily*)*, 34
dlogistic_normal *(*LogisticNormal*)*, 133
dmulti_normal *(*MultiNormal*)*, 160
dmulti_student_t *(*MultiStudentT*)*, 161
draws, *93*
draws-brms, 92
draws-index-brms, 93
draws. *(*draws-index-brms*)*, 93
dshifted_lnorm *(*Shifted_Lognormal*)*, 222
dskew_normal *(*SkewNormal*)*, 223
dstudent_t *(*StudentT*)*, 234
dvon_mises *(*VonMises*)*, 247
dwiener *(*Wiener*)*, 249
dzero_inflated_beta *(*ZeroInflated*)*, 251
dzero_inflated_beta_binomial *(*ZeroInflated*)*, 251

dzero_inflated_binomial *(*ZeroInflated*)*, 251
dzero_inflated_negbinomial *(*ZeroInflated*)*, 251
dzero_inflated_poisson *(*ZeroInflated*)*, 251

E_loo, *142*
emm_basis.brmsfit *(*emmeans-brms-helpers*)*, 94
emmeans-brms-helpers, 94
empty_prior *(*set_prior*)*, 216
environment, *85*
epilepsy, 96
ExGaussian, 97
exgaussian *(*brmsfamily*)*, 34
exponential *(*brmsfamily*)*, 34
expose_functions *(*expose_functions.brmsfit*)*, 98
expose_functions.brmsfit, 98
expose_stan_functions, *98*
expp1, 98
extend_family, *107*
extract_draws *(*prepare_predictions.brmsfit*)*, 194

facet_wrap, *68*
family, *34*, *39, 40*
family.brmsfit, 99
fcor, *19*, *78*, 99
fitted.brmsfit, 100
fixef *(*fixef.brmsfit*)*, 102
fixef.brmsfit, *62*, 102
formula, *26*, *55, 56*, *88*, *226*, *231*, *244*
Frechet, 103
frechet *(*brmsfamily*)*, 34
future, *29*, *126*, *206*

gam, *43*
gamm, *27*, *43*, *57*, *88*, *219*, *228*, *232*, *244*
Gamma, *39*
gen_extreme_value *(*brmsfamily*)*, 34
GenExtremeValue, 104
geom_contour, *68*
geom_errorbar, *68*
geom_jitter, *67, 68*
geom_point, *67*
geom_raster, *68*
geom_rug, *67, 68*
geom_smooth, *67, 68*
geometric *(*brmsfamily*)*, 34
get_dpar, 105
get_prior *(*default_prior*)*, 86
get_refmodel, *107*
get_refmodel.brmsfit, 106
ggplot, *68*, *150*
ggtheme, *54*, *68*, *167*
gp, *43*, 108, *219*
gr, *27*, *43*, *57*, 110, *227*, *231*
gtable, *168*

horseshoe, 112, *131*, *220*
Hurdle, 114
hurdle_cumulative *(*brmsfamily*)*, 34
hurdle_gamma *(*brmsfamily*)*, 34
hurdle_lognormal *(*brmsfamily*)*, 34
hurdle_negbinomial *(*brmsfamily*)*, 34
hurdle_poisson *(*brmsfamily*)*, 34
hypothesis, *27*, *53, 54*, *57*, *227*, *231*, *244*
hypothesis *(*hypothesis.brmsfit*)*, 115
hypothesis.brmsfit, 115

Index *(*draws-index-brms*)*, 93
index *(*addition-terms*)*, 8
inhaler, 118
init_refmodel, *107*
inits *(*inits.brmsfit*)*, 119
inits.brmsfit, 119
inv_logit_scaled, 121
InvGaussian, 120
is.brmsfit, 121
is.brmsfit_multiple, 122
is.brmsformula, 122
is.brmsprior, 122
is.brmsterms, 123
is.cor_arma *(*is.cor_brms*)*, 123
is.cor_brms, 123
is.cor_car *(*is.cor_brms*)*, 123
is.cor_cosy *(*is.cor_brms*)*, 123
is.cor_fixed *(*is.cor_brms*)*, 123
is.cor_sar *(*is.cor_brms*)*, 123
is.mvbrmsformula, 124
is.mvbrmsterms, 124
iterations, *(*draws-index-brms*)*, 93

kfold, *128, 129*, *207*
kfold *(*kfold.brmsfit*)*, 125

kfold-helpers, *127*
kfold.brmsfit, *107*, 125
kfold_predict, 128
kidney, 130

lasso, 131
launch_shinystan, *132*, *150*
launch_shinystan (launch_shinystan.brmsfit), 132
launch_shinystan.brmsfit, 132
lf (brmsformula-helpers), 51
log_lik, *82*, *85*, *125*, *143*, *144*, *200*
log_lik (log_lik.brmsfit), 134
log_lik.brmsfit, 134, *206*
log_posterior (diagnostic-quantities), 90
log_prob, *13*
logistic_normal (brmsfamily), 34
LogisticNormal, 133
logit_scaled, 133
logLik.brmsfit (log_lik.brmsfit), 134
logm1, 134
lognormal (brmsfamily), 34
LOO (loo.brmsfit), 136
loo, *7*, *63*, *64*, *127*, *135–137*, *141*, *206*, *207*
loo (loo.brmsfit), 136
LOO.brmsfit (loo.brmsfit), 136
loo.brmsfit, 136
loo::kfold_split_grouped, *126*
loo::kfold_split_stratified, *126*
loo::loo_model_weights, *139*
loo::psis, *200*
loo_compare, *63*, *64*, *126*, *136–138*, *146*, *248*
loo_compare (loo_compare.brmsfit), 138
loo_compare.brmsfit, 138
loo_epred (loo_predict.brmsfit), 142
loo_linpred (loo_predict.brmsfit), 142
loo_model_weights, *160*, *169*, *184*
loo_model_weights (loo_model_weights.brmsfit), 139
loo_model_weights.brmsfit, 139
loo_moment_match, *137*, *140*
loo_moment_match (loo_moment_match.brmsfit), 140
loo_moment_match.brmsfit, *137*, 140
loo_predict (loo_predict.brmsfit), 142
loo_predict.brmsfit, 142
loo_predictive_interval (loo_predict.brmsfit), 142
loo_R2 (loo_R2.brmsfit), 144
loo_R2.brmsfit, 144
loo_subsample, *135*, *146*, *196*
loo_subsample (loo_subsample.brmsfit), 145
loo_subsample.brmsfit, 145
loss, 146

ma, *14*, *15*, *19*, *79*, 147
make_conditions, *66*, 148, *212*
make_stancode (stancode), 224
make_standata (standata), 228
marginal_effects (conditional_effects.brmsfit), 64
marginal_smooths (conditional_smooths.brmsfit), 69
MCMC, *167*
mcmc_combo, *168*
mcmc_pairs, *165*
mcmc_plot (mcmc_plot.brmsfit), 149
mcmc_plot.brmsfit, 149
me, *52*, 151, *219*
mgcv::gamm, *213*
mgcv::s, *213*
mgcv::t2, *213*
mi, *44*, *46*, *151*, 152
mirai::daemons, *32*, *59*, *127*, *207*
mixture, *48*, 153
mm, *43*, 155, *157*
mmc, *156*, 157
mo, 158
model_weights, *169*, *184*, *185*
model_weights (model_weights.brmsfit), 159
model_weights.brmsfit, 159
multinomial (brmsfamily), 34
MultiNormal, 160
MultiStudentT, 161
mvbf, *48*
mvbf (mvbrmsformula), 162
mvbind, 162
mvbrmsformula, *26*, *49*, *52*, *55*, *56*, *88*, *162*, 162, *226*, *231*, *244*

nchains (draws-index-brms), 93

ndraws (draws-index-brms), 93
neff_ratio (diagnostic-quantities), 90
negbinomial (brmsfamily), 34
ngrps (ngrps.brmsfit), 163
ngrps.brmsfit, 163
niterations (draws-index-brms), 93
nlf (brmsformula-helpers), 51
nsamples (nsamples.brmsfit), 164
nsamples.brmsfit, 164
nuts_params (diagnostic-quantities), 90
nvariables (draws-index-brms), 93

opencl, *29*, *40*, 164
options, *28–30*, *58*, *228*, *232*, *250*

pairs, *165*
pairs.brmsfit, 165
pareto-k-diagnostic, *199*, *200*
pareto_k_ids, *137*, *140*, *206*
parnames, 166
parse_bf (brmsterms), 54
pasym_laplace (AsymLaplace), 18
pbeta_binomial (BetaBinomial), 23
pexgaussian (ExGaussian), 97
pfrechet (Frechet), 103
pgen_extreme_value (GenExtremeValue), 104
phurdle_gamma (Hurdle), 114
phurdle_lognormal (Hurdle), 114
phurdle_negbinomial (Hurdle), 114
phurdle_poisson (Hurdle), 114
pinv_gaussian (InvGaussian), 120
plan, *29*
plot.brms_conditional_effects (conditional_effects.brmsfit), 64
plot.brmsfit, 166
plot.brmshypothesis (brmshypothesis), 53
post_prob, *21*, *24*, *160*, *169*, *184*
post_prob (post_prob.brmsfit), 182
post_prob.brmsfit, 182
posterior_average, *185*
posterior_average (posterior_average.brmsfit), 168
posterior_average.brmsfit, 168
posterior_epred, *22*, *67*, *85*, *143*, *144*
posterior_epred (posterior_epred.brmsfit), 170
posterior_epred.brmsfit, *95*, *100–102*, 170, *174*
posterior_interval (posterior_interval.brmsfit), 172
posterior_interval.brmsfit, 172
posterior_linpred, *143*
posterior_linpred (posterior_linpred.brmsfit), 173
posterior_linpred.brmsfit, *95*, 173
posterior_predict, *67*, *85*, *143*, *193*
posterior_predict (posterior_predict.brmsfit), 174
posterior_predict.brmsfit, *170*, 174, *189*, *191*
posterior_samples (posterior_samples.brmsfit), 177
posterior_samples.brmsfit, 177
posterior_smooths (posterior_smooths.brmsfit), 178
posterior_smooths.brmsfit, 178
posterior_summary, *62*, *103*, 180, *203*
posterior_table, 181
powerscale, *82*
pp_average, *169*
pp_average (pp_average.brmsfit), 183
pp_average.brmsfit, 183
pp_check, *7*
pp_check (pp_check.brmsfit), 185
pp_check.brmsfit, 185
pp_expect (posterior_epred.brmsfit), 170
pp_mixture (pp_mixture.brmsfit), 187
pp_mixture.brmsfit, 187
PPC, *186*
predict.brmsfit, *186*, 189
predict.refmodel, *107*
predictive_error (predictive_error.brmsfit), 192
predictive_error.brmsfit, 192, *208*, *210*
predictive_interval (predictive_interval.brmsfit), 193
predictive_interval.brmsfit, 193
prepare_predictions, *101*, *105*, *127*, *129*,

135, *136*, *139*, *145*, *160*, *169*, *171*, *174*, *176*, *182*, *184*, *188*, *191*, *193*, *206*, *210*, *248*
prepare_predictions *(*prepare_predictions.brmsfit*)*, 194
prepare_predictions.brmsfit, 194
print.brmsfit, 196
print.brmshypothesis *(*brmshypothesis*)*, 53
print.brmsprior, 197
print.brmssummary *(*print.brmsfit*)*, 196
print.default, *54*
prior *(*set_prior*)*, 216
prior_ *(*set_prior*)*, 216
prior_draws *(*prior_draws.brmsfit*)*, 197
prior_draws.brmsfit, 197
prior_samples *(*prior_draws.brmsfit*)*, 197
prior_string *(*set_prior*)*, 216
prior_summary *(*prior_summary.brmsfit*)*, 198
prior_summary.brmsfit, 198
proj_linpred, *107*
proj_predict, *107*
pshifted_lnorm *(*Shifted_Lognormal*)*, 222
psis, *143*
psis *(*psis.brmsfit*)*, 199
psis.brmsfit, 199
pskew_normal *(*SkewNormal*)*, 223
pstudent_t *(*StudentT*)*, 234
pvon_mises *(*VonMises*)*, 247
pzero_inflated_beta *(*ZeroInflated*)*, 251
pzero_inflated_beta_binomial *(*ZeroInflated*)*, 251
pzero_inflated_binomial *(*ZeroInflated*)*, 251
pzero_inflated_negbinomial *(*ZeroInflated*)*, 251
pzero_inflated_poisson *(*ZeroInflated*)*, 251

qasym_laplace *(*AsymLaplace*)*, 18
qfrechet *(*Frechet*)*, 103
qgen_extreme_value *(*GenExtremeValue*)*, 104
qshifted_lnorm *(*Shifted_Lognormal*)*, 222
qskew_normal *(*SkewNormal*)*, 223
qstudent_t *(*StudentT*)*, 234
quantile, *180*

R2D2, *131*, 201, *220*
ranef *(*ranef.brmsfit*)*, 202
ranef.brmsfit, *62*, *116*, 202
rasym_laplace *(*AsymLaplace*)*, 18
rate *(*addition-terms*)*, 8
rbeta_binomial *(*BetaBinomial*)*, 23
rdirichlet *(*Dirichlet*)*, 91
read_csv_as_stanfit, 203
recompile_model, 205
recover_data.brmsfit *(*emmeans-brms-helpers*)*, 94
reloo, *127*, *137*
reloo *(*reloo.brmsfit*)*, 205
reloo.brmsfit, 205
rename_pars, 207
residuals.brmsfit, 208
resp_bhaz *(*addition-terms*)*, 8
resp_cat *(*addition-terms*)*, 8
resp_cens *(*addition-terms*)*, 8
resp_dec *(*addition-terms*)*, 8
resp_index *(*addition-terms*)*, 8
resp_mi, *152*
resp_mi *(*addition-terms*)*, 8
resp_rate *(*addition-terms*)*, 8
resp_se *(*addition-terms*)*, 8
resp_subset *(*addition-terms*)*, 8
resp_thres *(*addition-terms*)*, 8
resp_trials *(*addition-terms*)*, 8
resp_trunc *(*addition-terms*)*, 8
resp_vint *(*addition-terms*)*, 8
resp_vreal *(*addition-terms*)*, 8
resp_weights *(*addition-terms*)*, 8
restructure, 210
restructure.brmsfit, *211*, 211
rexgaussian *(*ExGaussian*)*, 97
rfrechet *(*Frechet*)*, 103
rgen_extreme_value *(*GenExtremeValue*)*, 104
rhat *(*diagnostic-quantities*)*, 90
rinv_gaussian *(*InvGaussian*)*, 120
rlogistic_normal *(*LogisticNormal*)*, 133
rmulti_normal *(*MultiNormal*)*, 160
rmulti_student_t *(*MultiStudentT*)*, 161
rows2labels, *149*, 212
rshifted_lnorm *(*Shifted_Lognormal*)*, 222
rskew_normal *(*SkewNormal*)*, 223
rstan::stan_model, *30*
rstudent_t *(*StudentT*)*, 234

runApp, *132*
rvon_mises *(*VonMises*)*, 247
rwiener *(*Wiener*)*, 249

s, *43*, 212
sampling, *30*
sar, *19*, *80*, 213
save_pars, *28*, 215
saveRDS, *11*, *30*, *58*
scale_colour_gradient, *68*
scale_colour_grey, *68*
se *(*addition-terms*)*, 8
set.seed, *107*, *116*, *169*, *184*
set_mecor *(*brmsformula-helpers*)*, 51
set_nl *(*brmsformula-helpers*)*, 51
set_prior, *27*, *31*, *46, 47*, *57*, *72*, *87*, *89*, *114*, *131*, *154*, *202*, 216, *227*, *231*, *244, 245*
set_rescor *(*brmsformula-helpers*)*, 51
Shifted_Lognormal, 222
shifted_lognormal *(*brmsfamily*)*, 34
skew_normal *(*brmsfamily*)*, 34
SkewNormal, 223
sratio *(*brmsfamily*)*, 34
Stan, *7*
stan, *29*, *31*, *73*
stancode, *7*, 224, *226*, *233*
stancode.brmsfit, *225*, 225
stancode.default, *225*, 226
standata, *7*, 228
standata.brmsfit, *229*, 229
standata.default, *229, 230*, 230
stanfit, *41*
stanmodel, *13*
stanplot, *7*
stanplot *(*mcmc_plot.brmsfit*)*, 149
stanvar, *27*, *57*, *84, 85*, *227*, *231*, 232
stanvars, *40*, *196*, *230*, *243*
stanvars *(*stanvar*)*, 232
student *(*brmsfamily*)*, 34
StudentT, 234
subset *(*addition-terms*)*, 8
subset_draws, *17*, *92, 93*
summarize_draws, *180*
summary, *7*
summary.brmsfit, *197*, 235

t2, *43*
t2 *(*s*)*, 212
TDist, *235*
theme, *54*, *68*, *167*
theme_black, 236
theme_default, *54*, *68*, *167*, *237*, 237
threading, *29*, *40*, *225*, *228*, *232, 233*, 237
thres *(*addition-terms*)*, 8
trials *(*addition-terms*)*, 8
trunc *(*addition-terms*)*, 8

unstr, *19*, 238
update, *28*, *58*
update.brmsfit, *206*, 239, *241*
update.brmsfit_multiple, 240
update.formula, *239*, *241*
update_adterms, 241

validate_newdata, *196*, *230*, 242
validate_prior, 243
VarCorr, *245*
VarCorr *(*VarCorr.brmsfit*)*, 245
VarCorr.brmsfit, 245
variables, *116*
variables *(*draws-index-brms*)*, 93
variables, *(*draws-index-brms*)*, 93
variables.brmsfit *(*draws-index-brms*)*, 93
varsel, *106*
vb, *30*
vcov.brmsfit, 246
Vectorize, *98*
vint *(*addition-terms*)*, 8
von_mises *(*brmsfamily*)*, 34
VonMises, 247
vreal *(*addition-terms*)*, 8

WAIC *(*waic.brmsfit*)*, 248
waic, *7*, *63, 64*, *135*, *248*
waic *(*waic.brmsfit*)*, 248
WAIC.brmsfit *(*waic.brmsfit*)*, 248
waic.brmsfit, 248
weibull *(*brmsfamily*)*, 34
weights *(*addition-terms*)*, 8
weights(), *200*
Wiener, 249
wiener *(*brmsfamily*)*, 34
wienerdist, *250*

xbeta *(*brmsfamily*)*, 34

zero_inflated_beta *(*brmsfamily*)*, 34

zero_inflated_beta_binomial
(brmsfamily), 34
zero_inflated_binomial (brmsfamily), 34
zero_inflated_negbinomial (brmsfamily),
34
zero_inflated_poisson (brmsfamily), 34
zero_one_inflated_beta (brmsfamily), 34
ZeroInflated, 251