Package ‘bsitar’

March 25, 2026

Type Package

Title Bayesian Super Imposition by Translation and Rotation Growth Curve Analysis

Version 0.3.3

Maintainer Satpal Sandhu <`satpal.sandhu@bristol.ac.uk`>

Description The Super Imposition by Translation and Rotation (SITAR) model is a shape-invariant nonlinear mixed effect model that fits a natural cubic spline mean curve to the growth data and aligns individual-specific growth curves to the underlying mean curve via a set of random effects (see Cole, 2010 <doi:10.1093/ije/dyq115> for details). The non-Bayesian version of the SITAR model can be fit by using the already available R package 'sitar'. Unlike the 'sitar' package which allows modelling of a single outcome only, the 'bsitar' package offers great flexibility in fitting models of varying complexities, including joint modelling of multiple outcomes such as height and weight (multivariate model). Additionally, the 'bsitar' package allows for the simultaneous analysis of an outcome separately for subgroups defined by a factor variable such as gender. This is achieved by fitting separate models for each subgroup (for example males and females for gender variable). An advantage of this approach is that posterior draws for each subgroup are part of a single model object, making it possible to compare coefficients across subgroups and test hypotheses. Since the 'bsitar' package is a front-end to the R package 'brms', it offers excellent support for post-processing of posterior draws via various functions that are directly available from the 'brms' package. In addition, the 'bsitar' package includes various customized functions that allow for the visualization of distance (increase in size with age) and velocity (change in growth rate as a function of age), as well as the estimation of growth spurt parameters such as age at peak growth velocity and peak growth velocity.

License GPL-2

Depends R (>= 3.6)

Imports brms (>= 2.23.0), rstan (>= 2.32.7), loo (>= 2.7.0), dplyr (>= 1.2.0), rlang (>= 1.1.2), Rdpack (>= 2.6.6), insight (>= 1.4.6), data.table (>= 1.18.0), collapse (>= 2.1.6), marginaleffects (>= 0.32.0), sitar (>= 1.5.0), magrittr, methods, utils

Suggests ggplot2 (>= 4.0.2), bayesplot (>= 1.15.0), posterior (>= 1.6.1), testthat (>= 3.3.2), ggridges (>= 0.5.6), jtools (>= 2.3.0), splines2 (>= 0.5.4), scales (>= 1.3.0), kableExtra (>= 1.4.0), knitr (>= 1.51), future (>= 1.69.0), future.callr (>= 0.10.2), future.apply (>= 1.20.0), future.mirai (>= 0.10.1), mirai (>= 2.5.0), doParallel (>= 1.0.17), foreach (>= 1.5.2), doFuture (>= 1.1.2), fastplyr (>= 0.9.0), cheapr (>= 1.3.2), dtplyr (>= 1.3.1), checkmate (>= 2.3.3), installr (>= 0.23.4), bayestestR (>= 0.17.0), R.utils, parallel, MASS, Matrix, tidyr, nlme, purrr, forcats, patchwork, tibble, pracma, extraDistr, bookdown, rmarkdown, spelling, Hmisc, growthcleanr, boot, R.rsp (>= 0.46.0), graphics, grDevices, abind, glue, stats

URL `https://github.com/Sandhu-SS/bsitar`

BugReports `https://github.com/Sandhu-SS/bsitar/issues`

VignetteBuilder knitr, R.rsp

RdMacros Rdpack

Config/testthat/edition 3

Encoding UTF-8

LazyData true

LazyLoad no

LazyDataCompression xz

NeedsCompilation no

RoxygenNote 7.3.3

Language en-US

Author Satpal Sandhu [aut, cre, cph] (ORCID: <`https://orcid.org/0000-0002-8539-6897`>)

Repository CRAN

Date/Publication 2026-03-25 15:40:02 UTC

Contents

add_model_criterion.bgmfit 3
berkeley 6
berkeley_exdata 8
berkeley_exfit 9
bsitar 9
expose_model_functions.bgmfit 50
fitted_draws.bgmfit 53
getNsObject 60
get_comparisons.bgmfit 61
get_growthparameters.bgmfit 75
get_predictions.bgmfit 93

growthparameters.bgmfit 108
hypothesis_test.bgmfit 116
is.bgmfit 129
loo_validation.bgmfit 130
modelbased_growthparameters.bgmfit 133
optimize_model.bgmfit 149
plot_conditional_effects.bgmfit 153
plot_curves.bgmfit 161
plot_ppc.bgmfit 172
predict_draws.bgmfit 176
update_model.bgmfit 184

Index **186**

`add_model_criterion.bgmfit`

Add fit criteria to the Bayesian SITAR Model

Description

The **add_model_criterion()** function is a wrapper around `brms::add_criterion()` that allows adding fit criteria to a model. Note that arguments such as `compare` and `pointwise` are relevant only for brms::add_loo, while `summary`, `robust`, and `probs` are ignored except for the `brms::bayes_R2()`

Usage

```
## S3 method for class 'bgmfit'
add_model_criterion(
  model,
  criterion = c("loo", "waic"),
  ndraws = NULL,
  draw_ids = NULL,
  compare = TRUE,
  pointwise = FALSE,
  model_name = NULL,
  overwrite = FALSE,
  force_save = FALSE,
  file = NULL,
  summary = TRUE,
  robust = FALSE,
  probs = c(0.025, 0.975),
  newdata = NULL,
  resp = NULL,
  cores = 1,
  model_deriv = NULL,
  return_model = TRUE,
  verbose = FALSE,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  envir = NULL,
  ...
)

add_model_criterion(model, ...)
```

Arguments

model	An object of class `bgmfit` representing the model to which the fit criteria will be added.
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).
ndraws	A positive integer indicating the number of posterior draws to use in estimation. If `NULL` (default), all draws are used.
draw_ids	An integer specifying the specific posterior draw(s) to use in estimation (default `NULL`).
compare	A flag indicating if the information criteria of the models should be compared to each other via `loo_compare`.
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_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.
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`.
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`.
summary	A logical value indicating whether only the estimate should be computed (`TRUE`), or whether the estimate along with SE and CI should be returned (`FALSE`, default). Setting `summary` to `FALSE` will increase computation time. Note that `summary = FALSE` is required to obtain correct estimates when `re_formula = NULL`.

`robust`	A logical value to specify the summary options. If `FALSE` (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 median absolute deviation (MAD) are applied instead. Ignored if `summary` is `FALSE`.
`probs`	The percentiles to be computed by the `quantile` function. Only used if `summary` is `TRUE`.
`newdata`	An optional data frame for estimation. If `NULL` (default), `newdata` is retrieved from the `model`.
`resp`	A character string (default `NULL`) to specify the response variable when processing posterior draws for `univariate_by` and `multivariate` models. See `bsitar()` for details on `univariate_by` and `multivariate` models.
`cores`	An integer specifying the number of cores for parallel execution. • If `NULL` (default), the number of cores is set to `future::availableCores() - 1`. • On non-Windows systems, this can be controlled globally via the `mc.cores` option.
`model_deriv`	A logical value specifying whether to estimate the velocity curve from the derivative function or by differentiating the distance curve. Set `model_deriv = TRUE` for functions that require the velocity curve, such as `growthparameters()` and `plot_curves()`. Set it to `NULL` for functions that use the distance curve (i.e., fitted values), such as `loo_validation()` and `plot_ppc()`.
`return_model`	A logical (default `TRUE`) to indicate whether to return the model object or just assign the criteria to the model. When `return_model = NULL`, then `return_model` is automatically assigned `FALSE` if `test_mode = FALSE`, and `TRUE` when `test_mode = TRUE`. Note that when `test_mode = TRUE`, the model object will be returned along with the criteria.
`verbose`	A logical argument (default `FALSE`) to specify whether to print information collected during the setup of the object(s).
`expose_function`	A logical value or `NULL` (default `FALSE`). Controls whether Stan functions are exposed for post-processing. • `TRUE`: Explicitly exposes Stan functions saved from the model fit. This is required when calculating `fit criteria` or `bayes_R2` during model optimization. • `FALSE` (default): Stan functions are not exposed. • `NULL`: The setting is inherited from the original `bsitar()` model fit configuration. **Note**: In the `optimize_model()` function, the default is `NULL` (inheriting behavior), whereas other post-processing functions default to `FALSE`.
`usesavedfuns`	A logical value (default `NULL`) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the `expose_functions` argument from the `bsitar()` call. Manual specification of `usesavedfuns` is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.

`clearenvfuns`	A logical value indicating whether to clear the exposed Stan functions from the environment (`TRUE`) or not (`FALSE`). If `NULL`, `clearenvfuns` is set based on the value of `usesavedfuns`: `TRUE` if `usesavedfuns = TRUE`, or `FALSE` if `usesavedfuns = FALSE`.
`envir`	The environment used for function evaluation. The default is `NULL`, which sets the environment to `parent.frame()`. Since most post-processing functions rely on **brms**, it is recommended to set `envir = globalenv()` or `envir = .GlobalEnv`, especially for derivatives like velocity curves.
`...`	Further arguments passed on to the functions from the **brms**

Value

An object of class `bgmfit` with the specified fit criteria added.

Author(s)

Satpal Sandhu <`satpal.sandhu@bristol.ac.uk`>

See Also

brms::add_loo, `brms::add_ic()`, `brms::add_waic()`, `brms::bayes_R2()`

Examples

```
# Fit Bayesian SITAR model

# To avoid model estimation which can take time, the Bayesian SITAR model fit
# to the 'berkeley_exdata' has been saved as an example fit ('berkeley_exfit').
# See 'bsitar' function for details on 'berkeley_exdata' and 'berkeley_exfit'.

model <- getNsObject(berkeley_exfit)

# Add model fit criteria (e.g., WAIC)
model <- add_model_criterion(model, criterion = c("waic"))
```

`berkeley` *Berkeley Child Guidance Study Data*

Description

Longitudinal growth records for 136 children.

Usage

```
berkeley
```

Format

A data frame with 4884 observations on the following 10 variables:

id Factor variable with levels 201-278 for males and 301-385 for females.

age Age in years (numeric vector).

height Height in cm (numeric vector).

weight Weight in kg (numeric vector).

stem.length Stem length in cm (numeric vector).

bi.acromial Biacromial diameter in cm (numeric vector).

bi.iliac Bi-iliac diameter in cm (numeric vector).

leg.circ Leg circumference in cm (numeric vector).

strength `Dynamometric` strength in pounds (numeric vector).

sex Factor variable with level 1 for male and level 2 for female.

Details

The data was originally included as an appendix in the book *Physical Growth of California Boys and Girls from Birth to Eighteen Years* authored by Tuddenham and Snyder (1954). The dataset was later used as an example in the **sitar** (Cole 2022) package after correcting transcription errors.

A more detailed description, including the frequency of measurements per year, is provided in the **sitar** package (Cole 2022). Briefly, the data consists of repeated growth measurements made on 66 boys and 70 girls (ages 0 to 21). The children were born in 1928-29 in Berkeley, California, and were of northern European ancestry. Measurements were taken at the following ages:

- 0 years (at birth),
- 0.085 years,
- 0.25 to 2 years (every 3 months),
- 2 to 8 years (annually),
- 8 to 21 years (every 6 months).

The data includes measurements for height, weight (undressed), stem length, biacromial diameter, bi-iliac diameter, leg circumference, and `dynamometric` strength.

Value

A data frame with 10 columns.

Author(s)

Satpal Sandhu `<satpal.sandhu@bristol.ac.uk>`

References

Cole T (2022). *sitar: Super Imposition by Translation and Rotation Growth Curve Analysis*. R package version 1.3.0, `https://CRAN.R-project.org/package=sitar`.

Tuddenham RD, Snyder MM (1954). "Physical growth of California boys and girls from birth to eighteen years." *Publications in Child Development. University of California, Berkeley*, **1**(2), 183–364. `https://pubmed.ncbi.nlm.nih.gov/13217130/`.

`berkeley_exdata` *Berkeley Child Guidance Study Data for Females*

Description

A subset of the berkeley data, containing longitudinal growth data for 70 females (aged 8 to 18 years).

Usage

```
berkeley_exdata
```

Format

A data frame with the following 3 variables:

id A factor variable identifying the subject.

age Age in years, numeric vector.

height Height in centimeters, numeric vector.

Details

Detailed information about the full dataset can be found in the berkeley data.

Value

A data frame with 3 columns.

Author(s)

Satpal Sandhu <`satpal.sandhu@bristol.ac.uk`>

`berkeley_exfit` *Model Fit to the Berkeley Child Guidance Study Data for Females*

Description

Bayesian SITAR model fit to the berkeley_exdata dataset (70 females, aged 8 to 18 years).

Usage

```
berkeley_exfit
```

Format

A model fit object containing a summary of the posterior draws.

Details

Detailed information about the data can be found in the berkeley_exdata dataset.

Value

An object of class `bgmfit` containing the posterior draws.

Author(s)

Satpal Sandhu `<satpal.sandhu@bristol.ac.uk>`

`bsitar` *Fit Bayesian SITAR Model*

Description

The **bsitar()** function fits the Bayesian version of the Super Imposition by Translation and Rotation (*SITAR*) model. The *SITAR* model is a nonlinear mixed-effects model that has been widely used to summarize growth processes (such as height and weight) from early childhood through adulthood.

The frequentest version of the *SITAR* model can be fit using the already available R package, **sitar** (Cole 2022). However, the **bsitar** package offers an enhanced Bayesian implementation that improves modeling capabilities. In addition to the conventional univariate analysis (i.e., modeling a single outcome), **bsitar** also supports:

- Univariate-by-subgroup analysis: This allows for simultaneous modeling of a single outcome across different subgroups defined by a factor variable (e.g., gender). The advantage is that posterior draws for each subgroup are part of a single model object, enabling comparisons of coefficients across groups and testing of various hypotheses.
- Multivariate analysis: This approach involves simultaneous joint modeling of two or more outcomes, allowing for more comprehensive growth modeling.

The Bayesian implementation in **bsitar** provides a more flexible and robust framework for growth curve modeling compared to the frequentist approach.

Usage

```
bsitar(
  x,
  y,
  id,
  data,
  df = 4,
  knots = NA,
  knots_selection = NULL,
  fixed = "a + b + c",
  random = "a + b + c",
  xoffset = "mean",
  bstart = xoffset,
  cstart = 0,
  xfun = NULL,
  yfun = NULL,
  xfunxoffset = TRUE,
  bound = 0.04,
  stype = "nsk",
  terms_rhs = NULL,
  a_formula = ~1,
  b_formula = ~1,
  c_formula = ~1,
  d_formula = ~1,
  s_formula = ~1,
  a_formula_gr = ~1,
  b_formula_gr = ~1,
  c_formula_gr = ~1,
  d_formula_gr = ~1,
  a_formula_gr_str = NULL,
  b_formula_gr_str = NULL,
  c_formula_gr_str = NULL,
  d_formula_gr_str = NULL,
  d_adjusted = FALSE,
  sigma_formula = NULL,
  sigma_formula_gr = NULL,
  sigma_formula_gr_str = NULL,
  sigma_formula_manual = NULL,
  sigmax = NULL,
  sigmaid = NULL,
  sigmadf = 4,
  sigmaknots = NA,
  sigmafixed = NULL,
  sigmarandom = "",
  sigmaxoffset = "mean",
  sigmabstart = "sigmaxoffset",
  sigmacstart = 0,
  sigmaxfun = NULL,
  sigmabound = 0.04,
  sigmaxfunxoffset = TRUE,
  dpar_formula = NULL,
  autocor_formula = NULL,
  family = gaussian(),
  custom_family = NULL,
  custom_formula = NULL,
  custom_prior = NULL,
  custom_stanvars = NULL,
 group_arg = list(groupvar = NULL, by = NULL, cor = "un", cov = NULL, dist = "gaussian"),
  sigma_group_arg = list(groupvar = NULL, by = NULL, cor = un, cov = NULL, dist =
    "gaussian"),
  univariate_by = list(by = NA, cor = "un", terms = "subset"),
 multivariate = list(mvar = FALSE, cor = "un", rescor = TRUE, rcorr_by = NULL, rcorr_gr
    = NULL, rcorr_method = NULL, rcorr_prior = NULL),
  a_prior_beta = normal(ymean, ysd, autoscale = FALSE),
  b_prior_beta = normal(0, 2, autoscale = FALSE),
  c_prior_beta = normal(0, 1, autoscale = FALSE),
  d_prior_beta = normal(0, 1, autoscale = FALSE),
  s_prior_beta = normal(lm, lm, autoscale = FALSE),
  a_cov_prior_beta = normal(0, 5, autoscale = FALSE),
  b_cov_prior_beta = normal(0, 1, autoscale = FALSE),
  c_cov_prior_beta = normal(0, 0.1, autoscale = FALSE),
  d_cov_prior_beta = normal(0, 1, autoscale = FALSE),
  s_cov_prior_beta = normal(lm, lm, autoscale = FALSE),
  a_prior_sd = normal(0, ysd, autoscale = FALSE),
  b_prior_sd = normal(0, 2, autoscale = FALSE),
  c_prior_sd = normal(0, 1, autoscale = FALSE),
  d_prior_sd = normal(0, 1, autoscale = FALSE),
  a_cov_prior_sd = normal(0, 5, autoscale = FALSE),
  b_cov_prior_sd = normal(0, 1, autoscale = FALSE),
  c_cov_prior_sd = normal(0, 0.1, autoscale = FALSE),
  d_cov_prior_sd = normal(0, 1, autoscale = FALSE),
  a_prior_sd_str = NULL,
  b_prior_sd_str = NULL,
  c_prior_sd_str = NULL,
  d_prior_sd_str = NULL,
  a_cov_prior_sd_str = NULL,
  b_cov_prior_sd_str = NULL,
  c_cov_prior_sd_str = NULL,
  d_cov_prior_sd_str = NULL,
  sigma_prior_beta = normal(0, 1, autoscale = FALSE),
  sigma_cov_prior_beta = normal(0, 0.5, autoscale = FALSE),
  sigma_prior_sd = normal(0, 0.25, autoscale = FALSE),
  sigma_cov_prior_sd = normal(0, 0.15, autoscale = FALSE),
  sigma_prior_sd_str = NULL,
  sigma_cov_prior_sd_str = NULL,
  rsd_prior_sigma = normal(0, ysd, autoscale = FALSE),
  dpar_prior_sigma = normal(0, ysd, autoscale = FALSE),
  dpar_cov_prior_sigma = normal(0, 1, autoscale = FALSE),
  autocor_prior_acor = uniform(-1, 1, autoscale = FALSE),
  autocor_prior_unstr_acor = lkj(1),
  gr_prior_cor = lkj(1),
  gr_prior_cor_str = lkj(1),
  sigma_prior_cor = lkj(1),
  sigma_prior_cor_str = lkj(1),
  mvr_prior_rescor = lkj(1),
  init = NULL,
  init_r = 0.5,
  a_init_beta = "lm",
  b_init_beta = 0,
  c_init_beta = 0,
  d_init_beta = 0,
  s_init_beta = "lm",
  a_cov_init_beta = 0,
  b_cov_init_beta = 0,
  c_cov_init_beta = 0,
  d_cov_init_beta = 0,
  s_cov_init_beta = 0,
  a_init_sd = "random",
  b_init_sd = "random",
  c_init_sd = "random",
  d_init_sd = "random",
  a_cov_init_sd = "random",
  b_cov_init_sd = "random",
  c_cov_init_sd = "random",
  d_cov_init_sd = "random",
  sigma_init_beta = "random",
  sigma_cov_init_beta = "random",
  sigma_init_sd = "random",
  sigma_cov_init_sd = "random",
  gr_init_cor = "random",
  sigma_init_cor = "random",
  rsd_init_sigma = "random",
  dpar_init_sigma = "random",
  dpar_cov_init_sigma = "random",
  autocor_init_acor = "random",
  autocor_init_unstr_acor = "random",
  mvr_init_rescor = "random",
  r_init_z = "random",
  vcov_init_0 = FALSE,
  jitter_init_beta = NULL,
  jitter_init_sd = NULL,
  jitter_init_cor = NULL,
  prior_data = NULL,
  init_data = NULL,
  init_custom = NULL,
  expose_function = FALSE,
  get_stancode = FALSE,
  get_standata = FALSE,
  get_formula = FALSE,
  get_stanvars = FALSE,
  get_priors = FALSE,
  get_priors_eval = FALSE,
  get_init_eval = FALSE,
  validate_priors = FALSE,
  set_self_priors = NULL,
  add_self_priors = NULL,
  set_replace_priors = NULL,
  set_same_priors_hierarchy = FALSE,
  outliers = NULL,
  unused = NULL,
  chains = 4,
  iter = 2000,
  warmup = floor(iter/2),
  thin = 1,
  cores = getOption("mc.cores", "optimize"),
  backend = getOption("brms.backend", "rstan"),
  threads = getOption("brms.threads", "optimize"),
  opencl = getOption("brms.opencl", NULL),
  normalize = getOption("brms.normalize", TRUE),
  algorithm = getOption("brms.algorithm", "sampling"),
  control = list(adapt_delta = 0.9, max_treedepth = 15),
  empty = FALSE,
  rename = TRUE,
  pathfinder_args = NULL,
  pathfinder_init = FALSE,
  data2 = NULL,
  data_custom = NULL,
  genquant_xyadj = FALSE,
  sample_prior = "no",
  save_pars = NULL,
  drop_unused_levels = TRUE,
  stan_model_args = list(),
  refresh = NULL,
  silent = 1,
  seed = 123,
  save_model = NULL,
  fit = NA,
  file = NULL,
  file_compress = TRUE,
  file_refit = getOption("brms.file_refit", "never"),
  future = getOption("future", FALSE),
  sum_zero = FALSE,
  global_args = NULL,
  parameterization = "ncp",
  verbose = FALSE,
  ...
)
```

Arguments

`x`	Predictor variable (typically age in years). For a `univariate` model, `x` is a single variable. For `univariate_by` and `multivariate` models, `x` can either be the same for all sub-models, or different for each sub-model. For example, when fitting a bivariate model, `x = list(x1, x2)` specifies that `x1` is the predictor variable for the first sub-model, and `x2` is the predictor for the second sub-model. To use `x1` as a common predictor variable for both sub-models, you can specify `x = list(x1)` or simply `x = x1`.
`y`	Response variable (e.g., repeated height measurements). For `univariate` and `univariate_by` models, `y` is specified as a single variable. In the case of a `univariate_by` model, the response vector for each sub-model is created and named internally based on the factor levels of the variable used to define the sub-model. For example, specifying `univariate_by = sex` creates response vectors `Female` and `Male` when `Female` is the first level and `Male` is the second level of the `sex` variable. In a `multivariate` model, the response variables are provided as a list, such as `y = list(y1, y2)`, where `y1` is the response variable for the first sub-model, and `y2` is the response for the second sub-model. Note that for the `multivariate` model, the data are not stacked; instead, response vectors are separate variables in the `data` and must be of equal length.
`id`	A factor variable uniquely identifying the groups (e.g., individuals) in the data frame. For `univariate_by` and `multivariate` models, the `id` can be the same (typically) for all sub-models, or different for each sub-model (see the `x` argument for details on setting different arguments for sub-models).
`data`	A data frame containing variables such as `x`, `y`, `id`, etc.
`df`	Degrees of freedom for the natural cubic spline design matrix (default 4). The `df` is used internally to construct knots (quantiles of the `x` distribution) for the spline design matrix. For `univariate_by` and `multivariate` models, the `df` can be the same across sub-models (e.g., `df = 4`) or different for each sub-model, such as `df = list(4, 5)`, where `df = 4` applies to the first sub-model and `df = 5` applies to the second sub-model.
`knots`	A numeric vector specifying the knots for the natural cubic spline design matrix (default `NULL`). Note that you cannot specify both `df` and `knots` at the same time, nor can both be `NULL`. In other words, either `df` or `knots` must be specified. Like `df`, the `knots` can be the same for all sub-models, or different for each sub-model when fitting `univariate_by` and `multivariate` models (see `df` for details).
`knots_selection`	A named list to control knot optimization during model fitting. This feature explores a larger candidate set (typically `df + 4`) and selects an optimal subset

that minimizes the chosen information criterion. The following elements are available:

`select` Selection strategy for knots and/or degrees of freedom. Options are `"knots"`, `"df"`, or `"both"`. `"knots"` (fully integrated into the **bsitar** workflow) automatically optimizes knot locations for a given `df` and passes the result to internal functions. The `"both"` option first determines the optimal degree of freedom, then selects knot locations using this `df`, which may result in a different (optimized) `df` from the user's initial choice. Users should note this potential change. Alternatively, `select = "df"` can be combined with `return = TRUE` to recover the optimal degree of freedom, which may then be reused as an argument in `bsitar()`.

`all_scores` Logical (default `FALSE`). When `TRUE` and `select = "df"`, returns the information criterion value for each evaluated `df`. Ignored for other selection types.

`plot_all_scores` Logical (default `FALSE`). When `TRUE` and `select = "df"` with `all_scores = TRUE`, plots the criterion for each `df`. Ignored otherwise.

`nsearch` Number of candidate knots in the search space. If `NULL` (default), set automatically to `df + 4` to provide sufficient coverage for optimization.

`criteria` Model selection criterion. Options include `"AIC"`, `"BIC"`, and cross-validation (`"CV"`). Lower values are preferred. Defaults to `"AIC"`. Note: Cross-validation often requires considerably more computation than `"AIC"` or `"BIC"`.

`cvk` Number of folds for cross-validation (default 10). Only used if `criteria = "CV"`.

`cviter` Number of cross-validation iterations to average (default 100). Only used if `criteria = "CV"`.

`bkrange` Logical (default `TRUE`). If `TRUE`, uses the full predictor range for boundary knots. Otherwise, boundary knots are set via quantiles (see `Hmisc::rcspline.eval()`). This option is relevant only for `stype = "rcs"` and overrides any later `bkrange` value globally for `knots_selection`.

`fix_bknots` Logical (default `TRUE`). If `TRUE`, fixes boundary knots during internal optimization. If `FALSE`, boundary knots are also candidates for optimization. Applies only when `stype = "rcs"`; see notes on `bkrange`.

`kspace` Knot spacing approach:

- `"un"`: Uniform quantile-based spacing (default)
- `"nu"`: Non-uniform spacing (algorithm-selected). Note: the actual number of knots may differ from the specified `df`.

`method` Algorithm for knot optimization:

- `"bs"`: built-in systematic workflow for knot optimization (default)
- `"rs"`: Recursive search based on user-defined function. Not implemented yet.

`when` Stage of knot optimization relative to predictor centering:

- `"bc"`: Before centering (raw `x` values; default)
- `"ac"`: After centering (i.e., `x - xoffset`)

`what` Diagnostics or plotting output type. Default `"knots"`. Available options:

- "knots": Optimized internal and boundary knots
- "df": Selected degree of freedom
- "plot1": Data and curve (original knots)
- "plot2": Data and curve (optimized knots)
- "plot3": Data and curves for both knot sets
- "plot4": Data, curve, and confidence bands (original knots)
- "plot5": Data, curve, and confidence bands (optimized knots)
- "plot6": Data, curve, and confidence bands (both knots)
- "plot7": Residual plot (original knots)
- "plot8": Residual plot (optimized knots)
- "plot9": Residual plots (comparison)

`return` Logical. If `TRUE`, returns the diagnostic plot object (as specified by `what`). Default is `FALSE`.

`print` Logical. If `TRUE`, displays the diagnostic plot (as specified by `what`). Default is `FALSE`.

Note that when both `return` and `print` are `FALSE`, the optimized knots are passed to the `bsitar()` function for model fitting. If `return = TRUE`, the knots are returned immediately, and model fitting is not performed. When `return = FALSE` and `print = TRUE`, the selected knots are displayed via a plot, and the model fitting proceeds using `bsitar()`.

Example usage: `bsitar( x = age, y = height, id = id, data = berkeley_exdata, knots_selection = list( select = "knots", nsearch = NULL, criteria = "AIC", bkrange = TRUE, fix_bknots = TRUE, method = "bs", when = "bc", what = "plot3", return = FALSE, print = TRUE ), seed = 123)`

`fixed` A character string specifying the fixed effects structure (default `'a+b+c'`). For `univariate_by` and `multivariate` models, you can specify different fixed effect structures for each sub-model. For example, `fixed = list('a+b+c', 'a+b')` implies that the fixed effects structure for the first sub-model is `'a+b+c'`, and for the second sub-model it is `'a+b'`.

`random` A character string specifying the random effects structure (default `'a+b+c'`). The approach to setting the `random` is the same as for the fixed effects structure (see `fixed`).

`xoffset` An optional character string or numeric value to set the origin of the predictor variable, `x` (i.e., centering of `x`). Available options include:

- `'mean'`: The mean of `x` (i.e., `mean(x)`),
- `'max'`: The maximum value of `x` (i.e., `max(x)`),
- `'min'`: The minimum value of `x` (i.e., `min(x)`),
- `'apv'`: Age at peak velocity estimated from the velocity curve derived from a simple linear model fit to the data,
- Any real number (e.g., `xoffset = 12`). Note that the value should be on the original scale of the `x` variable because this number will be automatically transformed based on the `xfunxoffset` which is set same as `xfun` unless `xfunxoffset` is explicitly set as `FALSE`. See `xfunxoffset` for details. The default is `xoffset = 'mean'`.

For univariate_by and multivariate models, xoffset can be the same or different for each sub-model (see x for details on setting different arguments for sub-models). If xoffset is a numeric value, it will be transformed internally (e.g., log or sqrt) depending on the xfun argument. Similarly, when xoffset is 'mean', 'min', or 'max', these values are calculated after applying the log or sqrt transformation to x.

bstart
: An optional character string or numeric value to set the origin of the fixed effect parameter b. The bstart argument is used to establish the location parameter for location-scale based priors (such as normal()) via the b_prior_beta argument, and/or the initial value via the b_init_beta argument. The available options for bstart are:

- 'mean': The mean of x (i.e., mean(x)),
- 'min': The minimum value of x (i.e., min(x)),
- 'max': The maximum value of x (i.e., max(x)),
- 'apv': Age at peak velocity estimated from the velocity curve derived from a simple linear model fit to the data,
- Any real number (e.g., bstart = 12).

The default is bstart = 'xoffset' (i.e., the same value as xoffset). For univariate_by and multivariate models, bstart can be the same for all sub-models (typically), or different for each sub-model (refer to x for details on setting different arguments for sub-models).

cstart
: An optional character string or numeric value to set the origin of the fixed effect parameter c. The cstart argument is used to establish the location parameter for location-scale based priors (such as normal()) via the c_prior_beta argument, and/or the initial value via the c_init_beta argument. The available options for cstart are:

- 'pv': Peak velocity estimated from the velocity curve derived from the simple linear model fit to the data,
- Any real number (e.g., cstart = 1).

Note that since parameter c is estimated on the exponential scale, the cstart should be adjusted accordingly. The default cstart is '0' (i.e., cstart = '0'). For univariate_by and multivariate models, cstart can be the same for all sub-models (typically), or different for each sub-model (refer to x for details on setting different arguments for sub-models).

xfun
: An optional character string specifying the transformation of the predictor variable x. The default value is NULL, indicating that no transformation is applied and the model is fit to the data with the original scale of x. The available transformation options are:

- 'log': Logarithmic transformation,
- 'sqrt': Square root transformation.

For univariate_by and multivariate models, the xfun can be the same for all sub-models (typically), or different for each sub-model (refer to x for details on setting different arguments for sub-models).

yfun
: An optional character string specifying the transformation of the response variable y. The default value is NULL, indicating that no transformation is applied

and the model is fit to the data with the original scale of `y`. The available transformation options are:

- `'log'`: Logarithmic transformation,
- `'sqrt'`: Square root transformation.

For `univariate_by` and `multivariate` models, the `yfun` can be the same for all sub-models (typically), or different for each sub-model (refer to `x` for details on setting different arguments for sub-models).

`xfunxoffset` Transformation applied to `xoffset` for `x` variable (default `TRUE`). See `xfun` for details. The default `xfunxoffset = TRUE` sets its value to same as `xfun`. Users rarely need to specify `xfunxoffset` themselves. One potential application is when a user wants to turn it off by setting `xfunxoffset = FALSE`. Note that `xfunxoffset` is called only when `xoffset` is a numeric values and not data based such as `xoffset = mean`. This is because even when `xfunxoffset` is not `TRUE`, `xoffset` is still automatically adjusted by `xfun`, as it is inferred from the transformed `x` variable.

`bound` An optional real number specifying the value by which the span of the predictor variable `x` should be extended (default is `0.04`). This extension can help in modeling edge cases. For more details, refer to the **sitar** package documentation. For `univariate_by` and `multivariate` models, the bound can be the same for all sub-models (typically), or different for each sub-model (refer to `x` for details on setting different arguments for sub-models).

`stype` A character string or a named list specifying the spline type to be used. The available options are:

- `'rcs'` (default): Constructs the spline design matrix using the truncated power basis (Harrell's method), implemented in `Hmisc::rcspline.eval()`.
- `'nsk'`: Implements a B-spline based natural cubic spline method, similar to `splines2::nsk()`.
- `'nsp'`: Implements a B-spline based natural cubic spline method, similar to `splines2::nsp()`.

The `'rcs'` method uses a truncated power basis, whereas `'nsk'` and `'nsp'` are B-spline-based methods. Unlike `splines2::nsp()` and `splines2::nsk()`, which normalize the spline basis by default, `'nsk'` and `'nsp'` return the non-normalized version of the spline. If normalization is desired, the user can specify `normalize = TRUE` in a list. For example, to use a normalized `'nsp'`, one can specify `stype = list(type = 'nsp', normalize = TRUE)`.

For more details, see `Hmisc::rcspline.eval()`, `splines2::nsk()`, and `splines2::nsp()`.

`terms_rhs` An optional character string (default `NULL`) specifying terms on the right-hand side of the response variable, but before the formula tilde sign ~. The `terms_rhs` is used when fitting a measurement error model.

For example, when fitting a model with measurement error in the response variable, the formula in `brms::brmsformula()` could be specified as `brmsformula(y | mi(sdy) ~ ...)`. In this case, `mi(sdy)` is passed to the formula via `terms_rhs = 'mi(sdy)'`.

For a `multivariate` model, each outcome can have its own measurement error variable. For instance, the `terms_rhs` can be specified as a list: `terms_rhs = list(mi(sdy1), mi(sdy2))`.

Note that brms::brmsformula() does not allow combining mi() with the subset() formulation which used in fitting univariate_by models.

a_formula
: Formula for the fixed effect parameter, a (default ~ 1). Users can specify different formulas when fitting univariate_by and multivariate models.

For example, a_formula = list(~1, ~1 + cov) specifies that the a_formula for the first sub-model includes only an intercept, while the second sub-model includes both an intercept and a covariate cov. The covariate(s) can be either continuous or factor variables. For factor covariates, dummy variables are created internally using stats::model.matrix().

The formula can include a combination of continuous and factor variables, as well as their interactions.

b_formula
: Formula for the fixed effect parameter, b (default ~ 1). See a_formula for details on how to specify the formula. The behavior and structure of b_formula are similar to a_formula.

c_formula
: Formula for the fixed effect parameter, c (default ~ 1). See a_formula for details on how to specify the formula. The behavior and structure of c_formula are similar to a_formula.

d_formula
: Formula for the fixed effect parameter, d (default ~ 1). See a_formula for details on how to specify the formula. The behavior and structure of d_formula are similar to a_formula.

s_formula
: Formula for the fixed effect parameter, s (default ~ 1). The s_formula sets up the spline design matrix. Typically, covariates are not included in the s_formula to limit the population curve to a single curve for the entire data. In fact, the **sitar** package does not provide an option to include covariates in the s_formula. However, the **bsitar** package allows the inclusion of covariates. In such cases, the user must justify the modeling of separate curves for each category when the covariate is a factor variable.

a_formula_gr
: Formula for the random effect parameter, a (default ~ 1). Similar to a_formula, users can specify different formulas when fitting univariate_by and multivariate models. The formula can include continuous and/or factor variables, including their interactions as covariates (see a_formula for details). In addition to setting up the design matrix for the random effect parameter a, users can define the group identifier and the correlation structure for random effects using the vertical bar || notation. For example, to include only an intercept for the random effects a, b, and c, you can specify:

a_formula_gr = ~1, b_formula_gr = ~1, c_formula_gr = ~1.

To specify the group identifier (e.g., id) and an unstructured correlation structure, use the vertical bar notation:

```
a_formula_gr = ~ (1|i|id)
b_formula_gr = ~ (1|i|id)
c_formula_gr = ~ (1|i|id)
```

Here, i within the vertical bars is a placeholder, and a common identifier (e.g., i) shared across the random effect formulas will model them as unstructured correlated random effects. For more details on this vertical bar approach, please see [brms::brm()].

An alternative approach to specify the group identifier and correlation structure is through the `group_by` argument. To achieve the same setup as described above with the vertical bar approach, users can define the formula part as:

`a_formula_gr = ~1`, `b_formula_gr = ~1`, `c_formula_gr = ~1`,

and use `group_by` as `group_by = list(groupvar = id, cor = un)`,
where `id` specifies the group identifier and `un` sets the unstructured correlation structure. See the `group_by` argument for more details.

`b_formula_gr` Formula for the random effect parameter, `b` (default `~ 1`). Similar to `a_formula_gr`, user can specify different formulas when fitting `univariate_by` and `multivariate` models. The formula can include continuous and/or factor variable(s), including their interactions as covariates (see `a_formula_gr` for details). In addition to setting up the design matrix for the random effect parameter `b`, the user can set up the group identifier and the correlation structure for random effects via the vertical bar `||` approach. For example, consider only an intercept for the random effects `a`, `b`, and `c` specified as
`a_formula_gr = ~1`,
`b_formula_gr = ~1`, and
`c_formula_gr = ~1`.
To specify the group identifier (e.g., `id`) and an unstructured correlation structure, the formula argument can be specified as:
`a_formula_gr = ~ (1|i|id)`
`b_formula_gr = ~ (1|i|id)`
`c_formula_gr = ~ (1|i|id)`
where `i` within the vertical bars `||` is just a placeholder. A common identifier (i.e., `i`) shared across random effect formulas are modeled as unstructured correlated. For more details on the vertical bar approach, please see `brms::brm()`.

`c_formula_gr` Formula for the random effect parameter, `c` (default `~ 1`). See `b_formula_gr` for details.

`d_formula_gr` Formula for the random effect parameter, `d` (default `~ 1`). See `b_formula_gr` for details.

`a_formula_gr_str`

Formula for the random effect parameter, `a` (default `NULL`), used when fitting a hierarchical model with three or more levels of hierarchy. For example, a model applied to data that includes repeated measurements (level 1) on individuals (level 2), which are further nested within growth studies (level 3).

For `a_formula_gr_str` argument, only the vertical bar approach (see `a_formula_gr`) can be used to define the group identifiers and correlation structure. An example of setting up a formula for a three-level model with random effect parameters `a`, `b`, and `c` is as follows:
`a_formula_gr_str = ~ (1|i|id:study) + (1|i2|study)`
`b_formula_gr_str = ~ (1|i|id:study) + (1|i2|study)`
`c_formula_gr_str = ~ (1|i|id:study) + (1|i2|study)`

In this example, `|i|` and `|i2|` set up unstructured correlation structures for the random effects at the individual and study levels, respectively. Note that `|i|` and `|i2|` must be distinct, as random effect parameters cannot be correlated across different levels of hierarchy.

Additionally, users can specify models with any number of hierarchical levels and include covariates in the random effect formula.

`b_formula_gr_str`
: Formula for the random effect parameter, `b` (default `NULL`), used when fitting a hierarchical model with three or more levels of hierarchy. For details, see `a_formula_gr_str`.

`c_formula_gr_str`
: Formula for the random effect parameter, `c` (default `NULL`), used when fitting a hierarchical model with three or more levels of hierarchy. For details, see `a_formula_gr_str`.

`d_formula_gr_str`
: Formula for the random effect parameter, `d` (default `NULL`), used when fitting a hierarchical model with three or more levels of hierarchy. For details, see `a_formula_gr_str`.

`d_adjusted`
: A logical indicator to adjust the scale of the predictor variable `x` when fitting the model with the random effect parameter `d`. The coefficient of parameter `d` is estimated as a linear function of `x`, i.e., `d * x`. If `FALSE` (default), the original `x` is used. When `d_adjusted` = `TRUE`, `x` is adjusted for the timing (`b`) and intensity (`c`) parameters as `x - b`) * `exp(c)` i.e., `d * ((x-b)*exp(c))`. The adjusted scale of `x` reflects individual developmental age rather than chronological age. This makes `d` more sensitive to the timing of puberty in individuals. See `sitar::sitar()` function for details.

`sigma_formula`
: Formula for the fixed effect distributional parameter, `sigma`. The `sigma_formula` sets up the fixed effect design matrix, which may include continuous and/or factor variables (and their interactions) as covariates for the distributional parameter. In other words, setting up the covariates for `sigma_formula` follows the same approach as for other fixed parameters, such as `a` (see `a_formula` for details). Note that `sigma_formula` estimates the `sigma` parameter on the `log` scale. By default, `sigma_formula` is `NULL`, as the `brms::brm()` function itself models `sigma` as a residual standard deviation (`RSD`) parameter on the link scale. The `sigma_formula`, along with the arguments `sigma_formula_gr` and `sigma_formula_gr_str`, allows `sigma` to be estimated as a random effect. The setup for fixed and random effects for `sigma` is similar to the approach used for other parameters such as `a`, `b`, and `c`.

  It is important to note that an alternative way to set up the fixed effect design matrix for the distributional parameter `sigma` is to use the `dpar_formula` argument. The advantage of `dpar_formula` over `sigma_formula` is that it allows users to specify both linear and nonlinear formulations using the `brms::lf()` and `brms::nlf()` syntax. These functions offer more flexibility, such as centering the predictors and enabling or disabling cell mean centering by excluding the intercept via `0 +` formulation. However, a disadvantage of the `dpar_formula` approach is that random effects cannot be included for `sigma`.

  `sigma_formula` and `dpar_formula` cannot be specified together. When either `sigma_formula` or `dpar_formula` is used, the default estimation of `RSD` by `brms::brm()` is automatically turned off.

  Users can specify an external function, such as `poly`, `splines::ns` etc. There are two ways to use external function.

- A conventional approach of using routine function such as `sigma_formula = ~ 1 + splines::ns(age, df = 3)`

- An external function with a single argument (the predictor), e.g., `sigma_formula = poly(age)`. Additional arguments are specified externally. For example, to set the degree of the polynomial to 3, a copy of the `poly` function can be created and modified as follows:
  `mypoly = poly; formals(mypoly)[['degree']] <- 3; mypoly(age)`.
  An advantage of this approach is that spline coefficient names are small.

`sigma_formula_gr`
: Formula for the random effect parameter, `sigma` (default `NULL`). See `a_formula_gr` for details. Similar to `sigma_formula`, external functions such as `poly`, `splines::ns` etc. can be used. For further details, please refer to the description of `sigma_formula`.

`sigma_formula_gr_str`
: Formula for the random effect parameter, `sigma`, when fitting a hierarchical model with three or more levels of hierarchy. See `a_formula_gr_str` for details. As with `sigma_formula`, external functions can be used. For example, `sigma_formula_gr_str = (1 + splines::ns(age, df = 3) | 11 | gr(id, by = NULL))` For further details, please refer to the description of `sigma_formula`.

`sigma_formula_manual`
: A custom formula for advanced modeling of the distributional parameter `sigma` using the `brms::nlf()` and `brms::lf()` functions. This is an advanced and experimental feature for specifying complex variance structures. The `sigma_formula_manual` can be can be a character string or a list of formulas.

Use Cases: The primary use cases for `sigma_formula_manual` are:

1. **Basic variance modelling:** A simple formula for modelling heteroscedasticity (`sigma`).
2. **Advanced variance modelling:** Calculating variance weights where the variance of residuals depends on a specified co variate, replicating functionality from the **nlme** package.
3. **Location-Scale Models:** Applying a full `SITAR`-like model to the scale (`sigma`) parameter in addition to the location (`mu`).

Basic variance modelling: A simple example of modeling `sigma` directly:

```
sigma_formula_manual = list(nlf(sigma ~ z) + lf(z ~ 1 + (1 | gr(id))))
```

Note that use can specify any external function in the linear predictor part of the formula `'lf()'` in order to model nonlinear trajectories. For example, user can include `splines2::nsk()` in the model as follows:

```
sigma_formula_manual = list(nlf(sigma ~ z) +
lf(z ~ 1 + splines2::nsk(age) + (1 | gr(id))))
```

Automatic Prior Assignment:
Priors for these linear predictors are assigned automatically for both mean and group-level random effects. The function uses priors specified via arguments

like 'sigma_prior_beta', 'sigma_cov_prior_beta', 'sigma_prior_sd', etc. These are the same arguments otherwise used for setting priors on parameters defined by 'sigma_formula', 'sigma_formula_gr', and 'sigma_formula_gr_str'.

Advanced variance modelling: This approach models heteroscedasticity using an explicit variance function. The **'bsitar'** package provides six different methods for variance modeling, five of which are implemented in the **'nlme'** package. The variance modeling approach uses a helper function 'sigmavarfun' (short hand form, 'vf') that sets up the appropriate formulation for variance modeling within the **'bsitar'** package.

For multivariate models, the 'sigmavarfun' is appropriately renamed to indicate response specific 'sigmavarfun' by adding '_response' as prefix. The documentation below provides a detailed overview of the available methods and their usage.

Available Methods (short hands in parentheses):

- 'varpower' ('vp'): Implements nlme::varPower().
- 'varconstpower' ('cp'): Implements nlme::varConstPower().
- 'varexp' ('ve'): Implements nlme::varExp() with 'form ~ x' as follows:
  'nlme::varExp(form ~ x)'
- 'fitted' ('fi'): Implements nlme::varExp() with 'form ~ fitted(.)' as follows:
  'nlme::varExp(form ~ fitted(.))'
- 'residual' ('re'): Implements nlme::varExp() with 'form ~ resid(.)' as follows:
  'nlme::varExp(form ~ resid(.))'
- 'mean' ('me'): A sixth method based on an example from the **brms** reference manual, which models the square root of the fitted values as follows:
  'nlme::varExp(form ~ sqrt(fitted(.)))'

Below are examples showing how to use 'sigmavarfun' to specify each of the six variance models. We encourage the use of short hand form, 'vf' to avoid any errors in correctly spelling the full form 'sigmavarfun'

1. varpower:

```
nlf(sigma ~ vf(param1, param2, predictor), method = 'vp') +
lf(param1 + param2 ~ 1)
```

2. varConstPower:

```
nlf(sigma ~ vf(param1, param2, param3, predictor), method = cp') +
lf(param1 + param2 + param3 ~ 1)
```

3. varExp:

```
nlf(sigma ~ vf(param1, param2, predictor), method = 've') +
lf(param1 + param2 ~ 1)
```

4. fitted:

```
nlf(sigma ~ vf(param1, param2, identity()), method = 'fi') +
lf(param1 + param2 ~ 1)
```

5. residual:

```
nlf(sigma ~ vf(param1, param2, identity(), resp), method = 're') +
 lf(param1 + param2 ~ 1)
```

6. mean:

```
nlf(sigma ~ vf(param1, param2, identity()), method = 'me') +
lf(param1 + param2 ~ 1)
```

Internal Predictor Transformations: The function applies internal transformations based on the chosen method:

- For `'varpower'` and `'varConstPower'`, the predictor is transformed to `log(abs(predictor))`.
- For `'varexp'`, the predictor is not transformed.
- For `'fitted'` and `'residual'`, `identity()` is internally set to `fitted(.)`.
- For `'mean'`, `identity()` is internally set to `sqrt(fitted(.))`.

Note that the default `'fitted'` (see above, `4. fitted:`) internally gets coded as `method = 'fittedexp'` (or short hand `method = 'fe'`) which sets up the variance form similar to the `varExp`. To set up the variance form similar to the `varpower`, please use `method = 'fittedpower'` (or short hand `method = 'fp'`). Another form, which models the non zero values for the `'fitted'`, can be specified as `method = 'fittedz'` (or short hand `method = 'fz'`).

Similar to the `fitted` (please above), the default `method = 'residual'` will set up the variance form similar to the `varExp` by internally coding the method argument as `method = 'residualexp'` (i.e., `method = 're'`). The power form can be set as `method = 'residualpower'` (or, `method = 'rp'`).

Like `fitted` and `residual` (please above), the default `method = 'mean'` will set up the variance form similar to the `varExp` by internally coding the method argument as `method = 'meanexp'` (i.e., `method = 'me'`). The power form can be set as `method = 'meanpower'` (or, `method = 'mp'`).

Covariates and Random Effects: The linear predictor, `'lf()'`, can be extended to include covariates and group-level random effects. For example, `'lf(param1 + param2 ~ 1)'` can be expanded as follows:

```
lf(param1 + param2 ~ 1 + covariate + (1 || gr(id, by = groupid)))
```

Automatic Prior Assignment: Priors for these linear predictors are assigned automatically for both mean and group-level random effects. The function uses priors specified via arguments like `'sigma_prior_beta'`, `'sigma_cov_prior_beta'`, `'sigma_prior_sd'`, etc. These are the same arguments otherwise used for setting priors on parameters defined by `'sigma_formula'`, `'sigma_formula_gr'`, and `'sigma_formula_gr_str'`.

To disable this automatic prior assignment, you can add the argument `prior = 'self'` to the `nlf()` function. For example:

```
nlf(..., method = 'vp', prior = 'self')
```

Note: If user disables the automatic prior assignment, it is advised to first get the required prior structure by calling the `bsitar()` with the `'get_prior = TRUE'` argument. The relevant portions of this structure can then be edited and added back to the model via the `'add_self_priors'` argument in `bsitar()`.

For `multivariate` model, the `sigma_formula_manual` can be used set up different form and predictor variables for each outcome separately by using the list approach as shown below:
`sigma_formula_manual = list( nlf(sigma ~ vf(param1, param2, identity()), method = 'fi') + lf(param1 + param2 ~ 1) , nlf(sigma ~ vf(param1, param2, identity()), method = 'fi') + lf(param1 + param2 ~ 1) )`.
Note that formulation name `'nlf()'` should be same across all outcomes. The appropriate response assignment is done internally. Also, parameter should not contain dots or underscores.

Location-Scale Models: Another important use case for `sigma_formula_manual` is modeling `sigma` in a location scale `SITAR` model, where the `SITAR` formula can be applied to the scale parameter (`sigma`) similar to the one used for modelling the location parameter `mu`. An example is:

`nlf(sigma ~ ls(x, sigmaa, sigmab, sigmac, sigmas1, sigmas2, sigmas3, sigmas4), loop = FALSE) + lf(sigmaa ~ 1+(1 |110| gr(id, by = NULL))+(1 |330| gr(study, by = NULL))) + lf(sigmab ~ 1+(1 |110| gr(id, by = NULL))+(1 |330| gr(study, by = NULL))) + lf(sigmac ~ 1+(1 |110| gr(id, by = NULL))+(1 |330| gr(study, by = NULL))) + lf(sigmas1 + sigmas2 + sigmas3 + sigmas4 ~ 1)`.

Here, `ls`, which is an abbreviation for location-scale, is a placeholder and gets replaced by the the name of actual function that is used for modelling the scale `sigma` part of the model. For example, if the name of the `mu` function is `sigmaSITARFun`, then the name of the `sigma` function would be `sigmaSITARFun` and hence `ls` gets replaced as `sigmaSITARFun`. All functions and corresponding names are created internally.

The first argument `x` of the function is again a placeholder for the actual predictor variable defined for the `sigma` modelling via the `sigmax` argument.In other words, the `x` gets replaced by the `sigmax` argument evaluated. For example, when `sigmax = age`, then the `x` gets replaced by `age`. Like `ls`, internal renaming of `x` is handled automatically.

The growth parameters `sigmaa`, `sigmab`, `sigmac` should match the input used for the `sigmafixed` and `sigmarandom` argument. In other words, either `sigmafixed` or `sigmarandom` must include the form for the corresponding growth parameter defined in the `ls` function. For example, when either or both `sigmafixed` and `sigmarandom` are defined as `a+b+c`, then all three growth parameters `sigmaa`, `sigmab`, `sigmac` should be part of the `ls` function. However, when only sunset of parameters are specified via the `sigmafixed` and `sigmarandom` form, say for example `a+b`, the `ls` function should exclude `sigmac` parameter.

Similarly, the number of spline parameters are based on the `sigmadf` argument. For example, when `sigmadf = 4`, then four spline parameters `sigmas1, ..., sigmas4` are included, as shown above in the example.

The other relevant information that is passed to the `ls` function can be set via the `sigmaknots`, `sigmaxoffset`, `sigmaxfun`, and `sigmabound` arguments.

Note that for the `location-scale` model, priors must be set up manually using the `add_self_priors` argument. To see which priors are required, the user can

	run the code with get_priors = TRUE. Also note that the default initial values for location-scale model are random.
sigmax	Predictor for the distributional parameter sigma. See x for details. Ignored if sigma_formula_manual = NULL.
sigmaid	A factor variable uniquely identifying the groups (e.g., individuals) in the data frame. If NULL (default), then sigmaid is set same as id. For univariate_by and multivariate models, the sigmaid can be the same (typically) for all sub-models, or different for each sub-model (see the id argument for details on setting different arguments for sub-models). Ignored if sigma_formula_manual = NULL.
sigmadf	Degree of freedom for the spline function used for sigma. See df for details. Ignored if sigma_formula_manual = NULL.
sigmaknots	Knots for the spline function used for sigma. See knots for details. Ignored if sigma_formula_manual = NULL.
sigmafixed	Fixed effect formula for the sigma structure. See fixed for details. Ignored if sigma_formula_manual = NULL.
sigmarandom	Random effect formula for the sigma structure. See random for details. Ignored if sigma_formula_manual = NULL. Currently not used even when sigma_formula_manual is specified.
sigmaxoffset	Offset for the x in the sigma structure. See xoffset for details. Ignored if sigma_formula_manual = NULL.
sigmabstart	Starting value for the b parameter in the sigma structure. See bstart for details. Ignored if sigma_formula_manual = NULL. Currently not used even when sigma_formula_manual is specified.
sigmacstart	Starting value for the c parameter in the sigma structure. See cstart for details. Ignored if sigma_formula_manual = NULL. Currently not used even when sigma_formula_manual is specified.
sigmaxfun	Transformation of sigmax variable for sigma structure. See xfun for details. Ignored if sigma_formula_manual = NULL.
sigmabound	Bounds for the x in the sigma structure. See bound for details. Ignored if sigma_formula_manual = NULL.
sigmaxfunxoffset	Transformation applied to sigmaxoffset for sigmax variable (default TRUE). See sigmaxfun for details. The default sigmaxfunxoffset = TRUE sets its value to same as sigmaxfun. Users rarely need to specify sigmaxfunxoffset themselves. One potential application is when a user wants to turn it off by setting sigmaxfunxoffset = FALSE. Note that sigmaxfunxoffset is called only when sigmaxoffset is a numeric values and not data based such as sigmaxoffset = mean. This is because even when sigmaxfunxoffset is not TRUE, sigmaxoffset is still automatically adjusted by sigmaxfun, as it is inferred from the transformed sigmax variable.
dpar_formula	Formula for the distributional fixed effect parameter, sigma (default NULL). See sigma_formula for details.
autocor_formula	Formula to set up the autocorrelation structure of residuals (default NULL). Allowed autocorrelation structures include:

- autoregressive moving average (arma) of order p and q, specified as autocor_formula = ~arma(p = 1, q = 1).
- autoregressive (ar) of order p, specified as autocor_formula = ~ar(p = 1).
- moving average (ma) of order q, specified as autocor_formula = ~ma(q = 1).
- unstructured (unstr) over time (and individuals), specified as autocor_formula = ~unstr(time, id).

See brms::brm() for further details on modeling the autocorrelation structure of residuals.

family Family distribution (default gaussian) and the link function (default identity). See brms::brm() for details on available distributions and link functions, and how to specify them. For univariate_by and multivariate models, the family can be the same for all sub-models (e.g., family = gaussian()) or different for each sub-model, such as family = list(gaussian(), student()), which sets gaussian distribution for the first sub-model and student_t distribution for the second. Note that the family argument is ignored if custom_family is specified (i.e., if custom_family is not NULL).

custom_family Specifies custom families (i.e., response distribution). Default is NULL. For details, see brms::custom_family(). Note that user-defined Stan functions must be exposed by setting expose_functions = TRUE.

custom_formula Specifies custom formula. Default is NULL. For details, see brms::brmsformula(). Currently ignored.

custom_prior Specifies custom prior Default is NULL. For details, see brms::prior(). Currently ignored. It is primarily designed to support setting custom prior for custom_formula. Note that custom_prior is different from the set_self_priors which evaluated throughout the call.

custom_stanvars

Allows the preparation and passing of user-defined variables to be added to Stan's program blocks (default NULL). This is primarily useful when defining a custom_family. For more details on specifying stanvars, see brms::custom_family(). Note that custom_stanvars are passed directly without conducting any sanity checks.

group_arg Specify arguments for group-level random effects. The group_arg should be a named list that may include groupvar, dist, cor, and by as described below:

- groupvar specifies the subject identifier. If groupvar = NULL (default), groupvar is automatically assigned based on the id argument.
- dist specifies the distribution from which the random effects are drawn (default gaussian). Currently, gaussian is the only available distribution (as per the brms::brm() documentation).
- by can be used to estimate a separate variance-covariance structure (i.e., standard deviation and correlation parameters) for random effect parameters (default NULL). If specified, the variable used for by must be a factor variable. For example, by = 'sex' estimates separate variance-covariance structures for males and females.

- cor specifies the covariance (i.e., correlation) structure for random effect parameters. The default covariance is unstructured (cor = un) for all model types (i.e., univariate, univariate_by, and multivariate). The alternative correlation structure available for univariate and univariate_by models is diagonal, which estimates only the variance parameters (standard deviations), while setting the covariance (correlation) parameters to zero. For *multivariate* models, options include un, diagonal, and un_s. The un structure models a full unstructured correlation, meaning that the group-level random effects across response variables are drawn from a joint multivariate normal distribution with shared correlation parameters. The cor = diagonal option estimates only variance parameters for each submodel, while setting the correlation parameters to zero. The cor = un_s option allows for separate estimation of unstructured variance-covariance parameters for each response variable.

Note that it is not necessary to define all or any of these options (groupvar, dist, cor, or by), as they will automatically be set to their default values if unspecified. Additionally, only groupvar from the group_arg argument is passed to the *univariate_by* and *multivariate* models, as these models have their own additional options specified via the univariate_by and multivariate arguments. Lastly, the group_arg is ignored when random effects are specified using the vertical bar || approach (see a_formula_gr for details) or when fitting a hierarchical model with three or more levels of hierarchy (see a_formula_gr_str for details).

sigma_group_arg

Specify arguments for modeling distributional-level random effects for sigma. The setup for sigma_group_arg follows the same approach as described for group-level random effects (see group_arg for details).

univariate_by Set up the univariate-by-subgroup model fitting (default NULL) via a named list with the following elements:

- by (optional, character string): Specifies the factor variable used to define the sub-models (default NA).
- cor (optional, character string): Defines the correlation structure. Options include un (default) for a full unstructured variance-covariance structure and diagonal for a structure with only variance parameters (i.e., standard deviations) and no covariance (i.e., correlations set to zero).
- terms (optional, character string): Specifies the method for setting up the sub-models. Options are 'subset' (default) and 'weights'. See brms::`addition-terms` for more details.

multivariate Set up the multivariate model fitting (default NULL) using a named list with the following elements:

- mvar (logical, default FALSE): Indicates whether to fit a multivariate model.
- cor (optional, character string): Specifies the correlation structure for group-level random effects. Available options are:
  - "un" (default): Models a full unstructured correlation, where group-level random effects across response variables are drawn from a joint multivariate normal distribution with shared correlation parameters.

  - "diagonal": Estimates only the variance parameters for each submodel, with the correlation parameters set to zero.
  - "un_s": Estimates unstructured variance-covariance parameters separately for each response variable.
- rescor (logical, default TRUE): Indicates whether to estimate the residual correlation between response variables.
- rcorr_by (character string, default NULL): The variable by which separate residual correlations between response variables are estimated. This must be a factor variable present in the data. Ignored when rescor = FALSE.
- rcorr_gr (character string, default NULL): The grouping variable (typically a level 2 variable like id) for which residual correlations between response variables are estimated. This must be a factor variable present in the data. Ignored when rescor = FALSE.
- rcorr_method (character string, default NULL): Specifies the method for estimating residual correlations: "lkj" (uses an LKJ distribution) or "cde" (uses the Cholesky decomposition of the covariance matrix with a uniform prior of -1, 1 for each off-diagonal element). If NULL, "lkj" is set as the default. Ignored when rescor = FALSE.
- rcorr_prior (numeric vector, default NULL): Used to specify the LKJ prior for each level of rcorr_by when rcorr_method = "lkj". The length of rcorr_prior should be either one (to use the same prior for all levels) or equal to the number of levels in rcorr_by. For rcorr_method = 'cde', uniform priors -1,1 are assigned to all correlation parameters. The argument rcorr_prior is ignored when rescor = FALSE.

a_prior_beta Specify priors for the fixed effect parameter, a. (default normal(ymean, ysd, autoscale = FALSE)). The following key points are applicable for all prior specifications. For full details, see brms::prior():

- Allowed distributions: normal, student_t, cauchy, lognormal, uniform, exponential, gamma, and inv_gamma (inverse gamma).
- For each distribution, upper and lower bounds can be set via lb and ub (default NA).
- Location-scale based distributions (such as normal, student_t, cauchy, and lognormal) have an autoscale option (default FALSE). This option multiplies the scale parameter by a numeric value. While **brms** typically uses a scaling factor of 1.0 or 2.5, the **bsitar** package allows any real number to be used (e.g., autoscale = 5.0).
- For location-scale distributions, fxl (function location) and fxs (function scale) are available to apply transformations to the location and scale parameters. For example, setting normal(2, 5, fxl = 'log', fxs = 'sqrt') translates to normal(log(2), sqrt(5)).
- fxls (function location scale) transforms both location and scale parameters. The transformation applies when both parameters are involved, as in the log-transformation for normal priors: log_location = log(location / sqrt(scale^2 / location^2 + 1)), log_scale = sqrt(log(scale^2 / location^2 + 1)). This can be specified as a character string or a list of functions.

- For strictly positive distributions like `exponential`, `gamma`, and `inv_gamma`, the lower bound (`lb`) is automatically set to zero.
- For uniform distributions, the option `addrange` widens the prior range symmetrically. For example, `uniform(a, b, addrange = 5)` adjusts the range to `uniform(a-5, b+5)`.
- For exponential distributions, the rate parameter is evaluated as the inverse of the specified value. For instance, `exponential(10.0)` is internally treated as `exponential(1.0 / 10.0) = exponential(0.1)`.
- Users do not need to specify each option explicitly, as missing options will automatically default to their respective values. For example, `a_prior_beta = normal(location = 5, scale = 1)` is equivalent to `a_prior_beta = normal(5, 1)`.
- For `univariate_by` and `multivariate` models, priors can either be the same for all submodels (e.g., `a_prior_beta = normal(5, 1)`) or different for each submodel (e.g., `a_prior_beta = list(normal(5, 1), normal(10, 5))`).
- For location-scale distributions, the location parameter can be specified as the mean (`ymean`) or median (`ymedian`) of the response variable, and the scale parameter can be specified as the standard deviation (`ysd`) or median absolute deviation (`ymad`). Alternatively, coefficients from a simple linear model can be used (e.g., `lm(y ~ age)`).
  Example prior specifications include: `a_prior_beta = normal(ymean, ysd)`, `a_prior_beta = normal(ymedian, ymad)`, `a_prior_beta = normal(lm, ysd)`.
  Note that options such as `ymean`, `ymedian`, `ysd`, `ymad`, and `lm` are available only for the fixed effect parameter `a`, not for other parameters like `b`, `c`, or `d`.

`b_prior_beta` Specify priors for the fixed effect parameter, `b`. The default prior is `normal(0, 2.0, autoscale = FALSE)`. For full details on prior specification, please refer to `a_prior_beta`.

- Allowed distributions include `normal`, `student_t`, `cauchy`, `lognormal`, `uniform`, `exponential`, `gamma`, and `inv_gamma`.
- You can set upper and lower bounds (`lb`, `ub`) as needed (default is `NA`).
- The `autoscale` option controls scaling of the prior's scale parameter. By default, this is set to `FALSE`.
- Further customization and transformations can be applied, similar to the `a_prior_beta` specification.

`c_prior_beta` Specify priors for the fixed effect parameter, `c`. The default prior is `normal(0, 1.0, autoscale = FALSE)`. For full details on prior specification, please refer to `a_prior_beta`.

- Allowed distributions include `normal`, `student_t`, `cauchy`, `lognormal`, `uniform`, `exponential`, `gamma`, and `inv_gamma`.
- Upper and lower bounds (`lb`, `ub`) can be set as necessary (default is `NA`).
- The `autoscale` option is also available for scaling the prior's scale parameter (default `FALSE`).

- Similar to `a_prior_beta`, further transformations or customization can be applied.

`d_prior_beta`
: Specify priors for the fixed effect parameter, `d`. The default prior is `normal(0, 1.0, autoscale = FALSE)`. For full details on prior specification, please refer to `a_prior_beta`. Note that to set the scale of location-scale based priors, the user can set scale as standard deviation `'xsd'` or the median absolute deviation `'xmad'` of the predictor variable `'x'`.

  - Allowed distributions include `normal`, `student_t`, `cauchy`, `lognormal`, `uniform`, `exponential`, `gamma`, and `inv_gamma`.
  - The option to set upper and lower bounds (`lb`, `ub`) is available (default is `NA`).
  - `autoscale` allows scaling of the prior's scale parameter and is `FALSE` by default.
  - For more advanced transformations or customization, similar to `a_prior_beta`, these options are available.

`s_prior_beta`
: Specify priors for the fixed effect parameter, `s` (i.e., spline coefficients). The default prior is `normal('lm', 'lm', autoscale = FALSE)`. The general approach is similar to the one described for other fixed effect parameters (see `a_prior_beta` for details). Key points to note:

  - When using location-scale based priors with 'lm' (e.g., `s_prior_beta = normal(lm, 'lm')`), the location parameter is set from the spline coefficients obtained from the simple linear model fit, and the scale parameter is based on the standard deviation of the spline design matrix. The location and scale parameters are typically set to `lm` (default), and `autoscale` is set to `FALSE`.
  - For location-scale based priors, the option `sethp` (logical, default `FALSE`) is available to define hierarchical priors. Setting `sethp = TRUE` alters the prior setup to use hierarchical priors: `s ~ normal(0, 'lm')` becomes `s ~ normal(0, 'hp')`, where `'hp'` is defined as `hp ~ normal(0, 'lm')`. The scale for the hierarchical prior is automatically taken from the `s` parameter, and it can also be defined using the same `sethp` option. For example, `s_prior_beta = normal(0, 'lm', sethp = cauchy)` will result in `s ~ normal(0, 'lm')`, `hp ~ cauchy(0, 'lm')` .
  - For `uniform` priors, you can use the option `addrange` to symmetrically expand the prior range.

  It has been observed that location-scale based prior distributions (such as `normal`, `student_t`, and `cauchy`) typically work well for spline coefficients.

  Note that the scale parameter for above `s_prior_beta = normal(lm, lm)` (which is same as `s_prior_beta = normal(lm,lm1)`) is derived from the standard deviation of the outcome and the spline design matrix as
  `sd(y)/sd(X)` where `y` is the outcome and `X` design matrix. The other variants of scale parameters are:
  `s_prior_beta = normal(lm,lm2)`) for which the scale parameter is defined as: `lm_se/sd(X)` where `lm_se` is the vector of standard error obtained from the linear model fit and `X` is the design matrix.
  `s_prior_beta = normal(lm,lm3)`) for which the scale parameter is defined as

	`lm_se` where `lm_se` is the vector of standard error obtained from the linear model fit.
`a_cov_prior_beta`	Specify priors for the covariate(s) included in the fixed effect parameter, `a` (default `normal(0, 5.0, autoscale = FALSE)`). The approach for specifying priors is similar to `a_prior_beta`, with a few differences: • The options `'ymean'`, `'ymedian'`, `'ysd'`, and `'ymad'` are not allowed for `a_cov_prior_beta`. • The `'lm'` option for the location parameter allows the covariate coefficient(s) to be obtained from a simple linear model fit to the data. Note that the `'lm'` option is only allowed for `a_cov_prior_beta` and not for covariates in other fixed or random effect parameters. • Separate priors can be specified for submodels when fitting `univariate_by` and `a_prior_beta` models (see `a_prior_beta` for details).
`b_cov_prior_beta`	Specify priors for the covariate(s) included in the fixed effect parameter, `b` (default `normal(0, 1.0, autoscale = FALSE)`). See `a_cov_prior_beta` for details.
`c_cov_prior_beta`	Specify priors for the covariate(s) included in the fixed effect parameter, `c` (default `normal(0, 0.1, autoscale = FALSE)`). See `a_cov_prior_beta` for details.
`d_cov_prior_beta`	Specify priors for the covariate(s) included in the fixed effect parameter, `d` (default `normal(0, 1.0, autoscale = FALSE)`). See `a_cov_prior_beta` for details.
`s_cov_prior_beta`	Specify priors for the covariate(s) included in the fixed effect parameter, `s` (default `normal(0, 10.0, autoscale = FALSE)`). As described in `s_formula`, the *SITAR* model does not allow covariates in the spline design matrix. If covariates are specified (see `s_formula`), the approach to setting priors for the covariates in parameter `s` is the same as for `a` (see `a_cov_prior_beta`). For location-scale based priors, the option `'lm'` sets the location parameter based on spline coefficients obtained from fitting a simple linear model to the data.
`a_prior_sd`	Specify priors for the random effect parameter, `a`. (default `normal(0, 'ysd', autoscale = FALSE)`). The prior is applied to the standard deviation (the square root of the variance), not the variance itself. The approach for setting the prior is similar to `a_prior_beta`, with the location parameter always set to zero. The lower bound is automatically set to `0` by `brms::brm()`. For `univariate_by` and `multivariate` models, priors can be the same or different for each submodel (see `a_prior_beta`).
`b_prior_sd`	Specify priors for the random effect parameter, `b`. (default `normal(0, 2.0, autoscale = FALSE)`). See `a_prior_sd` for details.
`c_prior_sd`	Specify priors for the random effect parameter, `c`. (default `normal(0, 1.0, autoscale = FALSE)`). See `a_prior_sd` for details.
`d_prior_sd`	Specify priors for the random effect parameter, `d`. (default `normal(0, 1.0, autoscale = FALSE)`). See `a_prior_sd` for details.

`a_cov_prior_sd`	Specify priors for the covariate(s) included in the random effect parameter, `a`. (default `normal(0, 5.0, autoscale = FALSE)`). The approach is the same as described for `a_cov_prior_beta`, except that no pre-defined options (e.g., `'lm'`) are allowed.
`b_cov_prior_sd`	Specify priors for the covariate(s) included in the random effect parameter, `b`. (default `normal(0, 1.0, autoscale = FALSE)`). See `a_cov_prior_sd` for details.
`c_cov_prior_sd`	Specify priors for the covariate(s) included in the random effect parameter, `c`. (default `normal(0, 0.1, autoscale = FALSE)`). See `a_cov_prior_sd` for details.
`d_cov_prior_sd`	Specify priors for the covariate(s) included in the random effect parameter, `d`. (default `normal(0, 1.0, autoscale = FALSE)`). See `a_cov_prior_sd` for details.
`a_prior_sd_str`	Specify priors for the random effect parameter, `a`, when fitting a hierarchical model with three or more levels of hierarchy. (default `NULL`). The approach is the same as described for `a_prior_sd`.
`b_prior_sd_str`	Specify priors for the random effect parameter, `b`, when fitting a hierarchical model with three or more levels of hierarchy. (default `NULL`). The approach is the same as described for `a_prior_sd_str`.
`c_prior_sd_str`	Specify priors for the random effect parameter, `c`, when fitting a hierarchical model with three or more levels of hierarchy. (default `NULL`). The approach is the same as described for `a_prior_sd_str`.
`d_prior_sd_str`	Specify priors for the random effect parameter, `d`, when fitting a hierarchical model with three or more levels of hierarchy. (default `NULL`). The approach is the same as described for `a_prior_sd_str`.
`a_cov_prior_sd_str`	Specify priors for the covariate(s) included in the random effect parameter, `a`, when fitting a hierarchical model with three or more levels of hierarchy. (default `NULL`). The approach is the same as described for `a_cov_prior_sd`.
`b_cov_prior_sd_str`	Specify priors for the covariate(s) included in the random effect parameter, `b`, when fitting a hierarchical model with three or more levels of hierarchy. (default `NULL`). The approach is the same as described for `a_cov_prior_sd_str`.
`c_cov_prior_sd_str`	Specify priors for the covariate(s) included in the random effect parameter, `c`, when fitting a hierarchical model with three or more levels of hierarchy. (default `NULL`). The approach is the same as described for `a_cov_prior_sd_str`.
`d_cov_prior_sd_str`	Specify priors for the covariate(s) included in the random effect parameter, `d`, when fitting a hierarchical model with three or more levels of hierarchy. (default `NULL`). The approach is the same as described for `a_cov_prior_sd_str`.
`sigma_prior_beta`	Specify priors for the fixed effect distributional parameter, `sigma`. (default `normal(0, 1.0, autoscale = FALSE)`). The approach is similar to that for `a_prior_beta`.

`sigma_cov_prior_beta`
: Specify priors for the covariate(s) included in the fixed effect distributional parameter, `sigma`. (default `normal(0, 0.5, autoscale = FALSE)`). Follows the same approach as `a_cov_prior_beta`.

`sigma_prior_sd`
: Specify priors for the random effect distributional parameter, `sigma`. (default `normal(0, 0.25, autoscale = FALSE)`). Same approach as `a_prior_sd`.

`sigma_cov_prior_sd`
: Specify priors for the covariate(s) included in the random effect distributional parameter, `sigma`. (default `normal(0, 0.15, autoscale = FALSE)`). Follows the same approach as `a_cov_prior_sd`.

`sigma_prior_sd_str`
: Specify priors for the random effect distributional parameter, `sigma`, when fitting a hierarchical model with three or more levels of hierarchy. (default `NULL`). Same approach as `a_prior_sd_str`.

`sigma_cov_prior_sd_str`
: Specify priors for the covariate(s) included in the random effect distributional parameter, `sigma`, when fitting a hierarchical model with three or more levels of hierarchy. (default `NULL`). Follows the same approach as `a_cov_prior_sd_str`.

`rsd_prior_sigma`
: Specify priors for the residual standard deviation parameter `sigma` (default `normal(0, 'ysd', autoscale = FALSE)`). Evaluated when both `dpar_formula` and `sigma_formula` are `NULL`. For location-scale based distributions, user can specify standard deviation (`ysd`) or the median absolute deviation (`ymad`) of outcome as the scale parameter. Also, residual standard deviation from the linear mixed model (`nlme::lme()`) or the linear model (`base::lm()`) fitted to the data. These are specified as `'lme_rsd'` and `'lm_rsd'`, respectively. Note that if `nlme::lme()` fails to converge, the option `'lm_rsd'` is set automatically. The argument `rsd_prior_sigma` is evaluated when both `dpar_formula` and `sigma_formula` are set to `NULL`.

`dpar_prior_sigma`
: Specify priors for the fixed effect distributional parameter `sigma` (default `normal(0, 'ysd', autoscale = FALSE)`). Evaluated when `sigma_formula` is `NULL`. See `rsd_prior_sigma` for details.

`dpar_cov_prior_sigma`
: Specify priors for the covariate(s) included in the fixed effect distributional parameter `sigma`. (default `normal(0, 1.0, autoscale = FALSE)`). Evaluated when `sigma_formula` is `NULL`.

`autocor_prior_acor`
: Specify priors for the autocorrelation parameters when fitting a model with `'arma'`, `'ar'`, or `'ma'` autocorrelation structures (see `autocor_formula`). The only allowed distribution is `uniform`, bounded between -1 and +1 (default `uniform(-1, 1, autoscale = FALSE)`). For the unstructured residual correlation structure, use `autocor_prior_unstr_acor`.

`autocor_prior_unstr_acor`
: Specify priors for the autocorrelation parameters when fitting a model with the unstructured (`'un'`) autocorrelation structure (see `autocor_formula`). The only allowed distribution is `lkj` (default `lkj(1)`). See `gr_prior_cor` for details on setting up the `lkj` prior.

`gr_prior_cor` Specify priors for the correlation parameter(s) of group-level random effects (default `lkj(1)`). The only allowed distribution is `lkj`, specified via a single parameter `eta` (see `brms::prior()` for details).

`gr_prior_cor_str`
Specify priors for the correlation parameter(s) of group-level random effects when fitting a hierarchical model with three or more levels of hierarchy (default `lkj(1)`). Same as `gr_prior_cor`.

`sigma_prior_cor`
Specify priors for the correlation parameter(s) of distributional random effects `sigma` (default `lkj(1)`). The only allowed distribution is `lkj` (see `gr_prior_cor` for details). Note that `brms::brm()` does not currently allow different `lkj` priors for the group level and distributional random effects sharing the same group identifier (`id`).

`sigma_prior_cor_str`
Specify priors for the correlation parameter(s) of distributional random effects `sigma` when fitting a hierarchical model with three or more levels of hierarchy (default `lkj(1)`). Same as `sigma_prior_cor`.

`mvr_prior_rescor`
Specify priors for the residual correlation parameter when fitting a multivariate model (default `lkj(1)`). The only allowed distribution is `lkj` (see `gr_prior_cor` for details).

`init` Initial values for the sampler. Options include:

- `'random'` (default): **Stan** randomly generates initial values for each parameter within a range defined by `init_r` (see below), or between -2 and 2 in unconstrained space if `init_r = NULL`.
- `'0'`: All parameters are initialized to zero.
- `'prior'`: Initializes parameters based on the specified prior.
- `NULL`: Initial values are provided by the corresponding init arguments defined below.

Note that `init = NULL` assigns initials for fixed effects, and variance co variance parameters (`vcov_init_0 = FALSE`) based on the individual setting for each parameter. If you want to initiate all parameters as `'random'`, then you must set `init = 'random'` which will be translated to `init = NULL` argument for `init = 'rstan'` and `init = 'cmdstanr'`.

`init_r` A positive real value specifying the range for random initial values (default `0.5`. This argument is used only when `init = 'random'`. Note that the default setting for `Stan` is `2.0` to assign random initials between a range `-2.0, 2.0` on the unconstrained parameter space.

`a_init_beta` Initial values for the fixed effect parameter, `a` (default `'random'`). Available options include:

- `'0'`: Initializes the parameter to zero.
- `'random'`: Initializes with random values within a specified range.
- `'prior'`: Uses values drawn from the prior distribution.
- `'ymean'`: Initializes with the mean of the response variable.
- `'ymedian'`: Initializes with the median of the response variable.

- `'lm'`: Initializes with the coefficients from a simple linear model fitted to the data.

Note that options `'ymean'`, `'ymedian'`, and `'lm'` are only available for the fixed effect parameter `a`. For `univariate_by` and `multivariate` models, initial values can be the same across submodels (e.g., `a_init_beta = '0'`) or different for each submodel (e.g., `list(a_init_beta = '0', a_init_beta = 'lm')`).

`b_init_beta` Initial values for the fixed effect parameter, `b` (default `'random'`). See `a_init_beta` for details on available options.

`c_init_beta` Initial values for the fixed effect parameter, `c` (default `'random'`). See `a_init_beta` for details on available options.

`d_init_beta` Initial values for the fixed effect parameter, `d` (default `'random'`). See `a_init_beta` for details on available options.

`s_init_beta` Initial values for the fixed effect parameter, `s` (default `'random'`). Available options include:

- `'0'`: Initializes the parameter to zero.
- `'random'`: Initializes with random values within a specified range.
- `'prior'`: Uses values drawn from the prior distribution.
- `'lm'`: Initializes with the coefficients from a simple linear model fitted to the data.

`a_cov_init_beta`

Initial values for the covariate(s) included in the fixed effect parameter, `a` (default `'random'`). Available options include:

- `'0'`: Initializes the covariates to zero.
- `'random'`: Initializes with random values within a specified range.
- `'prior'`: Uses values drawn from the prior distribution.
- `'lm'`: Initializes with the coefficients from a simple linear model fitted to the data.

Note that the `'lm'` option is only available for `a_cov_init_beta` and not for covariates in other parameters such as `b`, `c`, or `d`.

`b_cov_init_beta`

Initial values for the covariate(s) included in the fixed effect parameter, `b` (default `'random'`). See `a_cov_init_beta` for details.

`c_cov_init_beta`

Initial values for the covariate(s) included in the fixed effect parameter, `c` (default `'random'`). See `a_cov_init_beta` for details.

`d_cov_init_beta`

Initial values for the covariate(s) included in the fixed effect parameter, `d` (default `'random'`). See `a_cov_init_beta` for details.

`s_cov_init_beta`

Initial values for the covariate(s) included in the fixed effect parameter, `s` (default `'lm'`). See `a_cov_init_beta` for details. The option `'lm'` sets the spline coefficients obtained from a simple linear model fitted to the data. However, note that `s_cov_init_beta` serves as a placeholder and is not evaluated, as covariates are not allowed for the `s` parameter. For more details on covariates for `s`, refer to `s_formula`.

`a_init_sd` Initial value for the standard deviation of the group-level random effect parameter, `a` (default `'random'`). Available options are:

- `'random'`: Initializes with random values within a specified range.
- `'prior'`: Uses values drawn from the prior distribution.
- `'ysd'`: Sets the standard deviation (`sd`) of the response variable as the initial value.
- `'ymad'`: Sets the median absolute deviation (`mad`) of the response variable as the initial value.
- `'lme_sd_a'`: Sets the initial value based on the standard deviation of the random intercept obtained from a linear mixed model (`nlme::lme()`) fitted to the data. If `nlme::lme()` fails to converge, the option `'lm_sd_a'` will be used automatically.
- `'lm_sd_a'`: Sets the square root of the residual variance obtained from a simple linear model applied to the data as the initial value.

Note that the options `'ysd'`, `'ymad'`, `'lme_sd_a'`, and `'lm_sd_a'` are available only for the random effect parameter `a` and not for other group-level random effects.

Additionally, when fitting `univariate_by` and `multivariate` models, the user can set the same initial values for all sub-models, or different initial values for each sub-model.

`b_init_sd` Initial value for the standard deviation of the group-level random effect parameter, `b` (default `'random'`). Refer to `a_init_sd` for available options and details.

`c_init_sd` Initial value for the standard deviation of the group-level random effect parameter, `c` (default `'random'`). Refer to `a_init_sd` for available options and details.

`d_init_sd` Initial value for the standard deviation of the group-level random effect parameter, `d` (default `'random'`). Refer to `a_init_sd` for available options and details.

`a_cov_init_sd` Initial values for the covariate(s) included in the random effect parameter `a` (default `'random'`). Available options include:

- `'random'`: Random initialization.
- `'prior'`: Uses prior distribution values.

`b_cov_init_sd` Initial values for the covariate(s) included in the random effect parameter `b` (default `'random'`). Refer to `a_cov_init_sd` for available options and details.

`c_cov_init_sd` Initial values for the covariate(s) included in the random effect parameter `c` (default `'random'`). Refer to `a_cov_init_sd` for available options and details.

`d_cov_init_sd` Initial values for the covariate(s) included in the random effect parameter `d` (default `'random'`). Refer to `a_cov_init_sd` for available options and details.

`sigma_init_beta`

Initial values for the fixed effect distributional parameter `sigma` (default `'random'`). Available options include:

- `'random'`: Random initialization.
- `'prior'`: Uses prior distribution values.

`sigma_cov_init_beta`

Initial values for the covariate(s) included in the fixed effect distributional parameter `sigma` (default `'random'`). Refer to `sigma_init_beta` for available options and details.

`sigma_init_sd` Initial value for the standard deviation of the distributional random effect parameter `sigma` (default `'random'`). The approach is the same as described earlier for the group-level random effect parameters such as `a` (See `a_init_sd` for details).

`sigma_cov_init_sd`
Initial values for the covariate(s) included in the distributional random effect parameter `sigma` (default `'random'`). The approach is the same as described for `a_cov_init_sd` (See `a_cov_init_sd` for details).

`gr_init_cor` Initial values for the correlation parameters of group-level random effects parameters (default `'random'`). Allowed options are:

- `'random'`: Random initialization.
- `'prior'`: Uses prior distribution values.
- `'prior'`: A vector of length equal to the number of lower triangle elements. For example, the initials for a model with three random effects parameters can be specified as specified as `gr_init_cor = list(c(0.5. 0.5, 0.5))`

Note that when `vcov_init_0 = TRUE`, the `gr_init_cor` will be set as '0'.

`sigma_init_cor` Initial values for the correlation parameters of distributional random effects parameter `sigma` (default `'random'`). Allowed options are:

- `'random'`: Random initialization.
- `'prior'`: Uses prior distribution values.

`rsd_init_sigma` Initial values for the residual standard deviation parameter, `sigma` (default `'random'`). Options available are:

- `'0'`: Initializes the residual standard deviation to zero.
- `'random'`: Random initialization of the residual standard deviation.
- `'prior'`: Initializes the residual standard deviation based on prior distribution values.
- `'lme_rsd'`: Sets the initial value based on the standard deviation of residuals obtained from the linear mixed model (`nlme::lme()`) fitted to the data.
- `'lm_rsd'`: Sets the initial value as the square root of the residual variance from the simple linear model fitted to the data.

Note that if `nlme::lme()` fails to converge, the option `'lm_rsd'` is set automatically. The argument `rsd_init_sigma` is evaluated when both `dpar_formula` and `sigma_formula` are set to `NULL`.

`dpar_init_sigma`
Initial values for the distributional parameter `sigma` (default `'random'`). The approach and available options are the same as described for `rsd_init_sigma`. This argument is evaluated only when `dpar_formula` is not `NULL`.

`dpar_cov_init_sigma`
Initial values for the covariate(s) included in the distributional parameter `sigma` (default `'random'`). Allowed options are `'0'`, `'random'`, and `'prior'`.

`autocor_init_acor`
Initial values for the autocorrelation parameter (see `autocor_formula` for details). Allowed options are `'0'`, `'random'`, and `'prior'` (default `'random'`).

`autocor_init_unstr_acor`
Initial values for unstructured residual autocorrelation parameters (default `'random'`). Allowed options are `'0'`, `'random'`, and `'prior'`. The approach for setting initials for `autocor_init_unstr_acor` is the same as for `gr_init_cor`.

`mvr_init_rescor`
: Initial values for the residual correlation parameter when fitting a `multivariate` model (default `'random'`). Allowed options are `'0'`, `'random'`, and `'prior'`.

`r_init_z`
: Initial values for the standardized group-level random effect parameters (default `'random'`). These parameters are part of the Non-Centered Parameterization (NCP) approach used in the `brms::brm()`.

`vcov_init_0`
: A logical to set initial values for variance (standard deviation) and covariance (correlation) parameters to zero (when `vcov_init_0 = TRUE`). This allows for setting custom initial values for the fixed effects parameters while keeping the variance-covariance parameters at zero. When `vcov_init_0 = FALSE` (default), then variance-covariance parameters are assigned random initial values unless each individual parameter has it own initial values setting (e.g., `a_init_sd = 0`). Note that `vcov_init_0` is ignored when global initial values are assigned for all parameters via `init` argument.

`jitter_init_beta`
: A named list or numeric value to add a small amount of noise to an initial value or vector of initial values for the population level parameters.

  When `jitter_init_beta` is specified as a numeric value, it is treated as the percentage of perturbation applied to the initials. This value must be between `0` and `100`. Internally, the percentage is converted to a proportion (`percentage / 100`) and passed as the `amount` argument to the `base::jitter()` function. The `factor` argument is kept at its default value, `1`.

  The default, `jitter_init_beta = NULL`, means no perturbation is applied, so the same initial values are used for all chains. For mild perturbation, you might use a value such as `jitter_init_beta = 10`.

  Note that jitter is applied proportionally to the specified initial value, not as an absolute amount. For example, if the initial value is `100`, setting `jitter_init_beta = 0.1` causes the perturbed value to fall within the range `90` to `110`. Conversely, if the initial value is `10`, the perturbed value will be within `9` to `11`.

  If `jitter_init_beta` is provided as a named list, these elements are passed to the `base::jitter()` function. In addition to the `factor` and `amount` arguments, you may specify a `percent` argument, which is handled in the same way as a single numeric value. If both `percent` and `factor` are provided in the list, the effective perturbation is their product.

  To use the default behavior of `base::jitter()`, supply an empty `list()`, which will then be populated with the defaults: `list(..., factor = 1, amount = NULL)`. Please refer to the `base::jitter()` documentation for further details on how the `factor` and `amount` arguments affect the perturbation.

`jitter_init_sd`
: A named list or numeric value to add a small amount of noise to an initial value or vector of initial values for the standard deviation of random effect parameters.For `jitter_init_sd` a reasonable option of setting one percent as the perturbed value `jitter_init_sd = 1` has been found to work well during early testing. See `jitter_init_beta` for details on various options available to perturb the initials.

`jitter_init_cor`
: A named list or numeric value to add a small amount of noise to an initial value or vector of initial values for the correlations of random effect parameters.For

jitter_init_cor a reasonable option of setting one percent as the perturbed value jitter_init_sd = 0.1 has been found to work well during early testing. See jitter_init_beta for details on various options available to perturb the initials.

prior_data
: An optional argument (a named list, default NULL) that can be used to pass information to the prior arguments for each parameter (e.g., a_prior_beta). The prior_data is particularly helpful when passing a long vector or matrix as priors. These vectors and matrices can be created in the R framework and then passed using the prior_data. For example, to pass a vector of location and scale parameters when setting priors for covariate coefficients (with 10 dummy variables) included in the fixed effects parameter a, the following steps can be used:

  - Create the named objects prior_a_cov_location and prior_a_cov_scale in the R environment: prior_a_cov_location <- rnorm(n = 10, mean = 0, sd = 1) prior_a_cov_scale <- rep(5, 10).
  - Specify these objects in the prior_data list: prior_data = list(prior_a_cov_location = prior_a_cov_location, prior_a_cov_scale = prior_a_cov_scale).
  - Use the prior_data objects to set up the priors: a_cov_prior_beta = normal(prior_a_cov_location, prior_a_cov_scale).

init_data
: An optional argument (a named list, default NULL) that can be used to pass information to the initial arguments. The approach is identical to how prior_data is handled (as described above).

init_custom
: Specify a custom initialization object (a named list). The named list is directly passed to the init argument without verifying the dimensions or name matching. If initial values are set for some parameters via parameter-specific arguments (e.g., a_init_beta = 0), init_custom will only be passed to those parameters that do not have initialized values. To override this behavior and use all of init_custom values regardless of parameter-specific initials, set init = 'custom'.

expose_function
: An optional argument (logical, default FALSE) to indicate whether to expose the Stan function used in model fitting.

get_stancode
: An optional argument (logical, default FALSE) to retrieve the Stan code (see [brms::stancode()] for details).

get_standata
: An optional argument (logical, default FALSE) to retrieve the Stan data (see [brms::standata()] for details).

get_formula
: An optional argument (logical, default FALSE) to retrieve the model formula (see [brms::brmsformula()] for details).

get_stanvars
: An optional argument (logical, default FALSE) to retrieve the Stan variables (see [brms::stanvar()] for details).

get_priors
: An optional argument (logical, default FALSE) to retrieve the priors (see [brms::get_prior()] for details). Note that get_priors = TRUE will return priors based on the final code. In case user want to return the basic default priors, then it can be achieved by setting get_priors = "default".

get_priors_eval	An optional argument (logical, default FALSE) to retrieve the priors specified by the user.
get_init_eval	An optional argument (logical, default FALSE) to retrieve the initial values specified by the user.
validate_priors	An optional argument (logical, default FALSE) to validate the specified priors (see [brms::validate_prior()] for details).
set_self_priors	An optional argument (default NULL) to manually specify the priors. set_self_priors is passed directly to [brms::brm()] without performing any checks.
add_self_priors	An optional argument (default NULL) to append part of the prior object. This is for internal use only.
set_replace_priors	An optional argument (default NULL) to replace part of the prior object. This is for internal use only.
set_same_priors_hierarchy	An optional argument (default NULL) to replace part of the prior object. This is for internal use only.
outliers	An optional argument (default NULL) to remove outliers. This should be a named list passed directly to [sitar::velout()] and [sitar::zapvelout()] functions. This is for internal use only.
unused	An optional formula defining variables that are unused in the model but should still be stored in the model's data frame. Useful when variables are needed during post-processing.
chains	The number of Markov chains (default 4).
iter	The total number of iterations per chain, including warmup (default 2000).
warmup	A positive integer specifying the number of warmup (aka burn-in) 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 iterations should not exceed iter, and the default is iter/2.
thin	A positive integer specifying the thinning interval. Set thin > 1 to save memory and computation time if iter is large. Thinning is often used in cases with high autocorrelation of MCMC draws. An indication of high autocorrelation is poor mixing of chains (i.e., high rhat values) despite the model recovering parameters well. A useful diagnostic to check for autocorrelation of MCMC draws is the mcmc_acf function from the **bayesplot** package.
cores	Number of cores to be used when executing the chains in parallel. See brms::brm() for details. Unlike brms::brm(), which defaults the cores argument to cores=getOption("mc.cores", 1), the default cores in the **bsitar** package is cores=getOption("mc.cores", 'optimize'), which optimizes the utilization of system resources. The maximum number of cores that can be deployed is calculated as the maximum number of available cores minus 1. When the number of available cores exceeds the number of chains (see chains), then the number of cores is set equal to the number of chains.

	Another option is to set `cores` as `getOption("mc.cores", 'maximise')`, which sets the number of cores to the maximum number of cores available on the system regardless of the number of chains specified. Alternatively, the user can specify `cores` in the same way as `brms::brm()` with `getOption("mc.cores", 1)`. These options can be set globally using `options(mc.cores = x)`, where `x` can be `'optimize'`, `'maximise'`, or `1`. The `cores` argument can also be directly specified as an integer (e.g., `cores = 4`).
`backend`	A character string specifying the package to be used when executing the Stan model. The available options are `"rstan"` (the default) or `"cmdstanr"`. The backend can also be set globally for the current R session using the `"brms.backend"` option. See `brms::brm()` for more details.
`threads`	Number of threads to be used in within-chain parallelization. Note that unlike the `brms::brm()` which sets the `threads` argument as `getOption("brms.threads", NULL)` implying that no within-chain parallelization is used by default, the **bsitar** package, by default, sets `threads` as `getOption("brms.threads", 'optimize')` to utilize the available resources from the modern computing systems. The number of threads per chain is set as the maximum number of cores available minus 1. Another option is to set `threads` as `getOption("brms.threads", 'maximise')` which set the number threads per chains same as the maximum number of cores available. User can also set the `threads` similar to the `brms` i.e., `getOption("brms.threads", NULL)`. All these three options can be set globally as `options(brms.threads = x)` where `x` can be `'optimize'`, `'maximise'` or `NULL`. Alternatively, the number of threads can be set directly as `threads = threading(x)` where `X` is an integer. Other arguments that can be passed to the `threads` are `grainsize` and the `static`. See `brms::brm()` for further details on within-chain parallelization.
`opencl`	The platform and device IDs of the OpenCL device to use for GPU support during model fitting. If you are unsure about the IDs of your OpenCL device, `c(0,0)` is typically the default that should work. For more details on how to find the correct platform and device IDs, refer to `brms::opencl()`. This parameter can also be set globally for the current R session using the `"brms.opencl"` option.
`normalize`	Logical flag indicating whether normalization constants should be included in the Stan code (default is `TRUE`). If set to `FALSE`, normalization constants are omitted, which may increase sampling efficiency. However, this requires Stan version >= 2.25. Note that setting `normalize = FALSE` will disable some post-processing functions, such as `brms::bridge_sampler()`. This option can be controlled globally via the `brms.normalize` option.
`algorithm`	A character string specifying the estimation method to use. Available options are: • `"sampling"` (default): Markov Chain Monte Carlo (MCMC) method. • `"meanfield"`: Variational inference with independent normal distributions. • `"fullrank"`: Variational inference with a multivariate normal distribution. • `"fixed_param"`: Sampling from fixed parameter values. This parameter can be set globally via the `"brms.algorithm"` option (see `options` for more details).

`control`	A named `list` to control the sampler's behavior. The default settings are the same as those in `brms::brm()`, with one exception: the `max_treedepth` has been increased from 10 to 12 to better explore the typically challenging posterior geometry in nonlinear models. However, the `adapt_delta`, which is often increased for nonlinear models, retains its default value of 0.8 to avoid unnecessarily increasing sampling time. For full details on control parameters and their default values, refer to `brms::brm()`.
`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.
`pathfinder_args`	A named `list` of arguments passed to the `'pathfinder'` algorithm. This is used to set `'pathfinder'`-based initial values for the `'MCMC'` sampling. Note that `'pathfinder_args'` currently only works when `backend = "cmdstanr"`. If `pathfinder_args` is not `NULL` and the user specifies `backend = "rstan"`, the backend will automatically be changed to `cmdstanr`.
`pathfinder_init`	A logical value (default `FALSE`) indicating whether to use initial values from the `'pathfinder'` algorithm when fitting the final model (i.e., `'MCMC'` sampling). Note that `'pathfinder_args'` currently works only when `backend = "cmdstanr"`. If `pathfinder_args` is not `NULL` and the user specifies `backend = "rstan"`, the backend will automatically switch to `cmdstanr`. The arguments passed to the `'pathfinder'` algorithm are specified via `'pathfinder_args'`; if `'pathfinder_args'` is `NULL`, the default arguments from `'cmdstanr'` will be used.
`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.
`data_custom`	A `data.frame` object (default `NULL`). This is mainly for internal testing and not be used for routine model fitting.
`genquant_xyadj`	A logical (default `NULL`) indicating whether to generate `xyadj` in the `generated quantities` block of the `Stan`.
`sample_prior`	A character string indicating whether to draw samples from the priors in addition to the posterior draws. Options are `"no"` (the default), `"yes"`, and `"only"`. These prior draws can be used for various purposes, such as calculating Bayes factors for point hypotheses via `brms::hypothesis()`. Note that improper priors (including the default improper priors used by `brm`) are not sampled. For proper priors, see `brms::set_prior()`. Also, prior draws for the overall intercept are not obtained by default for technical reasons. See `brms::brmsformula()` for instructions on obtaining prior draws for the intercept. If `sample_prior` is set to `"only"`, draws will be taken solely from the priors, ignoring the likelihood, which allows you to generate draws from the prior predictive distribution. In this case, all parameters must have proper priors.
`save_pars`	An object generated by `brms::save_pars()` that controls which parameters should be saved in the model. This argument does not affect the model fitting

process itself but provides control over which parameters are retained in the final output.

drop_unused_levels
: A logical value indicating whether unused factor levels in the data should be dropped. The default is TRUE.

stan_model_args
: A list of additional arguments passed to rstan::stan_model when using the backend = "rstan" or backend = "cmdstanr". This allows customization of how models are compiled.

refresh
: An integer specifying the frequency of printing every nth iteration. By default, NULL indicates that the refresh rate will be automatically set by brms::brm(). Setting refresh is especially useful when thin is greater than 1, in which case the refresh rate is recalculated as (refresh * thin) / thin.

silent
: A verbosity level between 0 and 2. When set to 1 (the default), most informational messages from the compiler and sampler are suppressed. Setting it to 2 suppresses even more messages. The sampling progress is still printed. To turn off all printing, set refresh = 0. Additionally, when using backend = "rstan", you can prevent the opening of additional progress bars by setting open_progress = FALSE.

seed
: An integer or NA (default) specifying the seed for random number generation, ensuring reproducibility of results. If set to NA, **Stan** will randomly select the seed.

save_model
: A character string or NULL (default). If provided, the Stan code for the model will be saved in a text file with the name corresponding to the string specified in save_model.

fit
: An instance of class brmsfit from a previous fit (default is NA). If a brmsfit object is provided, the compiled model associated with the fitted result is reused, and any arguments that modify the model code or data are ignored. It is generally recommended to use the update method for this purpose, rather than directly passing the fit argument.

file
: Either NULL or a character string. If a character string is provided, the fitted model object is saved using saveRDS in a file named after the string supplied in file. The .rds extension is automatically added. If the specified file already exists, the existing model object is loaded and returned instead of refitting the model. To overwrite an existing file, you must manually remove the file or specify the file_refit argument. The file name is stored within the brmsfit object for later use.

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. This can be set globally for the current R session via the "brms.file_refit" option (see options). The possible options are:
  - "never" (default): The fit is always loaded if it exists, and fitting is skipped.
  - "always": The model is always refitted, regardless of existing fits.

- "on_change": The model is refitted only if the model, data, algorithm, priors, sample_prior, stanvars, covariance structure, or similar parameters have changed.

If you believe a false positive occurred, you can use [brms::brmsfit_needs_refit()] to investigate why a refit is deemed necessary. A 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.

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

sum_zero
: Currently ignored. Placeholder for future development.

global_args
: Currently ignored. Placeholder for future development.

parameterization
: A character string specifying the type of parameterization to use for drawing group-level random effects. Options are: 'ncp' for Non-Centered Parameterization (NCP), and 'cp' for Centered Parameterization (CP).

The NCP is generally recommended when the likelihood is weak (e.g., few observations per individual) and is the default approach (and only option) in brms::brm().

The CP parameterization is typically more efficient when a relatively large number of observations are available across individuals. We consider a 'relatively large number' as at least 10 repeated measurements per individual. If there are fewer than 10, NCP is used automatically. This behavior applies only when parameterization = NULL. To explicitly set CP parameterization, use parameterization = 'cp'.

Note that since brms::brm() does not support CP, the stancode generated by brms::brm() is edited internally before fitting the model using rstan::rstan() or "cmdstanr", depending on the chosen backend. Therefore, CP parameterization is considered experimental and may fail if the structure of the generated stancode changes in future versions of brms::brm().

verbose
: An optional argument (logical, default FALSE) to indicate whether to print information collected during setting up the model formula priors, and initials. As an example, the user might be interested in knowing the response variables created for the sub model when fitting a univariate-by-subgroup model. This information can then be used in setting the desired order of options passed to each such model such as df, prior, initials etc.

...
: Further arguments passed to brms::brm(). This may include additional arguments that are either passed directly to the underlying model fitting function, or used for internal purposes. Specifically, the ... can also be used to pass arguments used for testing and debugging, such as: match_sitar_a_form, sigmamatch_sitar_a_form, displayit, setcolh, setcolb, decomp and smat. Note that for all arguments (such as intercept) to correctly pass to the respective functions, use smat and not stype when setting up the spline arguments.

These internal arguments are typically not used in regular model fitting but can be relevant for certain testing scenarios or advanced customization. Users are generally not expected to interact with these unless working on debugging or testing specific features of the model fitting process.

Details

The *SITAR* is a shape-invariant nonlinear mixed-effects growth curve model that fits a population average (i.e., mean) curve to the data and aligns each individual's growth trajectory to the underlying population curve via a set of (typically) three random effects: `size`, `timing`, and `intensity`. Additionally, a slope parameter can be included as a random effect to estimate the variability in adult growth rate (see `sitar::sitar()` for details).

The concept of a shape-invariant model (SIM) was first introduced by Lindstrom (1995), and later used by Beath (2007) to model infant growth data (birth to 2 years). The current version of the *SITAR* model was developed by Cole et al. (2010) and has been extensively used for modeling growth data (see Nembidzane et al. 2020 and Sandhu 2020).

The frequentist version of the *SITAR* model can be fit using the already available R package, **sitar** (Cole 2022). The framework of the Bayesian implementation of the *SITAR* model in the **bsitar** package is similar to the **sitar** package, with the main difference being that **sitar** uses `splines::ns()` to construct the B-splines based natural cubic spline design matrix, whereas **bsitar** implements a different strategy to create natural cubic splines. The **bsitar** offers three different types of splines: **nsp**, **nsk**, and **rcs**. Both **nsp** and **nsk** use the B-splines basis to generate the natural cubic spline design matrix as implemented in `splines2::nsp()` and `splines2::nsk()`, whereas **rcs** is based on the truncated power basis approach (see Harrell and others (2001) and Harrell Jr. (2022) for details) to construct the spline design matrix. While all approaches produce the same growth curves, the model-estimated spline coefficients differ from each other.

A key challenge in spline based regression methods is to select the number and location of knots. The standard approach is to place knots by a regular sequence of quantiles between the outer boundaries. A regression curve can easily be fitted to the sample using a relatively high number of knots. The problem is then over fitting, where a regression model has a good fit to the given sample but does not generalize well to other samples. A low knot count is thus preferred. However, the standard knot selection process can lead to under performance in the sparser regions of the predictor variable, especially when using a low number of knots. It can also lead to over fitting in the denser regions. The **sitar** package offers an option (see `knots_selection` argument) to use a search algorithm that implements a backward method for knot selection that shows reduced prediction error and a lower information criterion (IC) or cross-validation scores compared to the standard knot selection process in simulation experiments. See (Arnes et al. 2023) for details. The approach suggested by (Arnes et al. 2023) is implemented via option `method = 'bs'` when setting up the `knots_selection` argument. We strongly recommend that users carefully consider the placement of knots—whether using the approach suggested here or another method of their choice—to ensure optimal model performance.

Like **sitar**, the (Cole et al. 2010), the **bsitar** package fits the *SITAR* model with (usually) three random effects: size (parameter `a`), timing (parameter `b`), and intensity (parameter `c`). Additionally, there is a slope parameter (parameter `d`) that models the variability in the adult slope of the growth curve (see `sitar::sitar()` for details).

Note that author of the **sitar** package (Cole et al. 2010) enforces the inclusion of `d` parameter as a random effect only, excluding it from the fixed structure of the model. However, the **bsitar** package allows inclusion of the `d` parameter in both the fixed and/or random effects structures of the *SITAR* model.

For the three-parameter version of the *SITAR* model (default), the fixed effects structure (i.e., population average trajectory) is specified as `fixed = 'a+b+c'`, and the random effects structure, capturing the deviation of individual trajectories from the population average curve, is specified as `random = 'a+b+c'`.

The **bsitar** package offers flexibility in model specification. For example:

- A fixed-effect version of the *SITAR* model can be fit by setting `random = ''`.
- The fixed-effect structure can include a subset of parameters, such as size and timing (`fixed = 'a+b'`) or size and intensity (`fixed = 'a+c'`).
- For a four-parameter version of the *SITAR* model, parameter `d` is included in the `fixed` and/or `random` arguments.

The **bsitar** package internally depends on the **brms** package (see Bürkner 2022; Bürkner 2021), which fits a wide range of hierarchical linear and nonlinear regression models, including multivariate models. The **brms** package itself depends on **Stan** for full Bayesian inference (see Stan Development Team 2023; Gelman et al. 2015). Like **brms**, the **bsitar** package allows flexible prior specifications based on user's knowledge of growth processes (e.g., timing and intensity of growth spurts).

The **brms** package uses a combination of `normal` and `student_t` distributions for regression coefficients, group-level random effects, and the distributional parameter (`sigma`), while **rstanarm** uses `normal` distributions for regression coefficients and group-level random effects, but sets `exponential` for the distributional parameter (`sigma`). By default, **bsitar** uses `normal` distributions for all parameters, including regression coefficients, standard deviations of group-level random effects, and the distributional parameter. Additionally, **bsitar** provides flexibility in choosing scale parameters for location-scale distributions (such as `normal` and `student_t`).

The **bsitar** package also allows three types of model specifications: `'univariate'`, `'univariate_by'`, and `'multivariate'`:

- `'univariate'` fits a single model to an outcome variable.
- `'univariate_by'` fits two or more sub-models to an outcome defined by a factor variable (e.g., sex).
- `'multivariate'` fits a joint model to multiple outcomes with shared random effects.

The **bsitar** package offers full flexibility in specifying predictors, degrees of freedom for design matrices, priors, and initial values. The package also allows users to specify options in a user-friendly manner (e.g., `univariate_by = sex` is equivalent to `univariate_by = 'sex'`).

Value

An object of class `brmsfit, bsitar`, which contains the posterior draws, model coefficients, and other useful information related to the model fitting. This object includes details such as the fitted model, the data used, prior distributions, and any other relevant outputs from the Stan model fitting process. The resulting object can be used for further analysis, diagnostics, and post-processing, including model summary statistics, predictions, and visualizations.

Note

The package is under continuous development, and new models, post-processing features, and improvements are being actively worked on. Keep an eye on future releases for additional functionality and updates to enhance model fitting, diagnostics, and analysis capabilities.

Author(s)

Satpal Sandhu <satpal.sandhu@bristol.ac.uk>

References

Arnes JI, Hapfelmeier A, Horsch A, Braaten T (2023). “Greedy knot selection algorithm for restricted cubic spline regression.” *Frontiers in Epidemiology*, **Volume 3 - 2023**. ISSN 2674-1199, doi:10.3389/fepid.2023.1283705, https://www.frontiersin.org/journals/epidemiology/articles/10.3389/fepid.2023.1283705.

Beath KJ (2007). “Infant growth modelling using a shape invariant model with random effects.” *Statistics in Medicine*, **26**(12), 2547–2564. doi:10.1002/sim.2718, Type: Journal article.

Bürkner P (2021). “Bayesian Item Response Modeling in R with brms and Stan.” *Journal of Statistical Software*, **100**(5), 1–54. doi:10.18637/jss.v100.i05.

Bürkner P (2022). *brms: Bayesian Regression Models using Stan*. R package version 2.18.0, https://CRAN.R-project.org/package=brms.

Cole T (2022). *sitar: Super Imposition by Translation and Rotation Growth Curve Analysis*. R package version 1.3.0, https://CRAN.R-project.org/package=sitar.

Cole TJ, Donaldson MDC, Ben-Shlomo Y (2010). “SITAR—a useful instrument for growth curve analysis.” *International Journal of Epidemiology*, **39**(6), 1558–1566. ISSN 0300-5771, doi:10.1093/ije/dyq115, tex.eprint: https://academic.oup.com/ije/article-pdf/39/6/1558/18480886/dyq115.pdf.

Gelman A, Lee D, Guo J (2015). “Stan: A Probabilistic Programming Language for Bayesian Inference and Optimization.” *Journal of Educational and Behavioral Statistics*, **40**(5), 530-543. doi:10.3102/1076998615606113.

Harrell FE, others (2001). *Regression modeling strategies: with applications to linear models, logistic regression, and survival analysis*, volume 608. Springer.

Harrell Jr. FE (2022). *Hmisc: Harrell Miscellaneous*. R package version 4.7-2, https://hbiostat.org/R/Hmisc/.

Lindstrom MJ (1995). “Self-modelling with random shift and scale parameters and a free-knot spline shape function.” *Statistics in Medicine*, **14**(18), 2009-2021. doi:10.1002/sim.4780141807, https://pubmed.ncbi.nlm.nih.gov/8677401/.

Nembidzane C, Lesaoana M, Monyeki KD, Boateng A, Makgae PJ (2020). “Using the SITAR Method to Estimate Age at Peak Height Velocity of Children in Rural South Africa: Ellisras Longitudinal Study.” *Children*, **7**(3), 17. ISSN 2227-9067, doi:10.3390/children7030017.

Sandhu SS (2020). *Analysis of longitudinal jaw growth data to study sex differences in timing and intensity of the adolescent growth spurt for normal growth and skeletal discrepancies*. Thesis, University of Bristol.

Stan Development Team (2023). *Stan Reference Manual version 2.31*. https://mc-stan.org/docs/reference-manual/.

See Also

brms::brm() brms::brmsformula() brms::prior()

Examples

```
# Below, we fit a SITAR model to a subset of the Berkley height data,
# specifically the data for 70 girls between the ages of 8 and 18.
# This subset is used as an example in the vignette for the 'sitar' package.
#
# The original Berkley height data contains repeated growth measurements for
# 66 boys and 70 girls (ages 0-21). For this example, we use a subset of the
# data for 70 girls aged 8 to 18 years.
#
# For details on the full Berkley height dataset, refer to 'sitar' package
# documentation (help file: ?sitar::berkeley). Further details on the subset
# of the data used here can be found in the vignette ('Fitting_models_with_SITAR',
# package = 'sitar').

# Load the 'berkeley_exdata' that has been pre-saved
berkeley_exdata <- getNsObject(berkeley_exdata)

# Fit frequentist SITAR model with df = 3 using the sitar package

model_ml <- sitar::sitar(x = age, y = height, id = id,
                          df = 3,
                          data = berkeley_exdata,
                          xoffset = 'mean',
                          fixed = 'a+b+c',
                          random = 'a+b+c',
                          a.formula = ~1,
                          b.formula = ~1,
                          c.formula = ~1
                          )


# Fit Bayesian SITAR model

# To avoid time-consuming model estimation, the Bayesian SITAR model fit has
# been saved as an example fit ('berkeley_exfit'). This model was fit using
# 2 chains (2000 iterations per chain) with thinning set to 6 for memory
# efficiency. Users are encouraged to refit the model using default settings
# (4 chains, 2000 iterations per chain, thin = 1) as suggested by the Stan
# team. Note that with thinning set to 6 (thin = 6), only one sixth of total
# draws will be saved and hence the effective sample size is expected to be
# small.

# Check if the pre-saved model 'berkeley_exfit' exists
# berkeley_exfit <- bsitar:::berkeley_exfit
```

```
berkeley_exfit <- getNsObject(berkeley_exfit)

if(exists('berkeley_exfit')) {
  model <- berkeley_exfit
} else {
  # Fit model with default priors
  # Refer to the documentation for prior on each parameter
  model <- bsitar(x = age, y = height, id = id,
                  df = 3,
                  data = berkeley_exdata,
                  xoffset = 'mean',
                  fixed = 'a+b+c',
                  random = 'a+b+c',
                  a_formula = ~1,
                  b_formula = ~1,
                  c_formula = ~1,
                  threads = brms::threading(NULL),
                  chains = 2, cores = 2, iter = 1000, thin = 6)

}

# Generate model summary
summary(model)

# Compare model summary with the frequentist SITAR model
print(model_ml)

# Check model fit via posterior predictive checks using plot_ppc.
# This function is based on pp_check from the 'brms' package.
plot_ppc(model, ndraws = NULL)

# Plot distance and velocity curves using plot_conditional_effects.
# This function works like conditional_effects from the 'brms' package,
# with the added option to plot velocity curves.

# Distance curve
plot_conditional_effects(model, deriv = 0)

# Velocity curve
plot_conditional_effects(model, deriv = 1)

# Plot distance and velocity curves along with parameter estimates using
# plot_curves (similar to plot.sitar from the sitar package).
plot_curves(model, apv = TRUE)

# Compare plots with the frequentist SITAR model
plot(model_ml)
```

expose_model_functions.bgmfit

Expose user-defined Stan functions for the Bayesian SITAR model

Description

The **expose_model_functions()** function is a wrapper around `rstan::expose_stan_functions()` that exposes user-defined Stan function(s). These functions are necessary for post-processing the posterior draws.

Usage

```
## S3 method for class 'bgmfit'
expose_model_functions(
  model,
  scode = NULL,
  expose = TRUE,
  select_model = NULL,
  returnobj = TRUE,
  vectorize = FALSE,
  verbose = FALSE,
  sigmafun = FALSE,
  backend = NULL,
  path = NULL,
  envir = NULL,
  ...
)

expose_model_functions(model, ...)
```

Arguments

`model`	An object of class `bgmfit`.
`scode`	A character string containing the user-defined Stan function(s) in `Stan` code. If `NULL` (the default), the `scode` will be retrieved from the `model`.
`expose`	A logical (default `TRUE`) to indicate whether to expose the functions and add them as an attribute to the `model`.
`select_model`	A character string (default `NULL`) to specify the model name. This parameter is for internal use only.
`returnobj`	A logical (default `TRUE`) to specify whether to return the model object. If `expose = TRUE`, it is advisable to set `returnobj = TRUE`.
`vectorize`	A logical (default `FALSE`) to indicate whether the exposed functions should be vectorized using `base::Vectorize()`. Note that currently, `vectorize` should be set to `FALSE`, as setting it to `TRUE` may not work as expected.
`verbose`	A logical argument (default `FALSE`) to specify whether to print information collected during the setup of the object(s).

`sigmafun`	A logical (default `FALSE`) to indicate whether to return the sigma functions. This parameter is for internal use only. Ignored
`backend`	A character string (default `NULL`) to set up the the backend method (`'rstan'` or `'cmdstanr'`) for compiling and exposing `stan` functions. If `NULL`, then the backend is same as the backend method used for model fitting. Note that when `backend = FALSE`, then default backend will be set as `'rstan'`. This is particularly useful because `backend = 'cmdstanr'` will fail on `WSL` system.
`path`	A character string to set up the path of installed `CmdStan`. If `NULL` (default)
`envir`	The environment used for function evaluation. The default is `NULL`, which sets the environment to `parent.frame()`. Since most post-processing functions rely on **brms**, it is recommended to set `envir = globalenv()` or `envir = .GlobalEnv`, especially for derivatives like velocity curves.
`...`	Additional arguments passed to the `rstan::expose_stan_functions()` function. The "..." can be used to set the compiler, which can be either `rstan::stanc()` or `rstan::stan_model()`. You can also pass other compiler-specific arguments such as save_dso for `rstan::stan_model()`. Note that while both `rstan::stanc()` and `rstan::stan_model()` can be used as compilers before calling `rstan::expose_stan_functions()`, it is important to note that the execution time for `rstan::stan_model()` is approximately twice as long as `rstan::stanc()`.

Value

An object of class `bgmfit` if `returnobj = TRUE`; otherwise, it returns `NULL` invisibly.

Author(s)

Satpal Sandhu <`satpal.sandhu@bristol.ac.uk`>

See Also

`rstan::expose_stan_functions()`

Examples

```
# Fit Bayesian SITAR model

# To avoid mode estimation which takes time, the Bayesian SITAR model fit to
# the 'berkeley_exdata' has been saved as an example fit ('berkeley_exfit').
# See 'bsitar' function for details on 'berkeley_exdata' and 'berkeley_exfit'.

# Check and confirm whether the model fit object 'berkeley_exfit' exists
 berkeley_exfit <- getNsObject(berkeley_exfit)

model <- berkeley_exfit

# To save time, argument expose is set as FALSE, which runs a dummy test
# and avoids model compilation that often takes time.
```

```
expose_model_functions(model, expose = FALSE)
```

`fitted_draws.bgmfit` *Estimate fitted (Expected) values for the Bayesian SITAR model*

Description

The **fitted_draws()** function is a wrapper around the `brms::fitted.brmsfit()` function, which allows users to obtain fitted values (and their summaries) from the posterior draws. For more details, refer to the documentation for `brms::fitted.brmsfit()`. An alternative approach is to `get_predictions()` function which is based on the **marginaleffects**.

Usage

```
## S3 method for class 'bgmfit'
fitted_draws(
  model,
  newdata = NULL,
  resp = NULL,
  dpar = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  re_formula = NA,
  allow_new_levels = FALSE,
  sample_new_levels = "uncertainty",
  incl_autocor = TRUE,
  numeric_cov_at = NULL,
  levels_id = NULL,
  avg_reffects = NULL,
  aux_variables = NULL,
  grid_add = NULL,
  ipts = NULL,
  deriv = 0,
  model_deriv = TRUE,
  summary = TRUE,
  robust = FALSE,
  transform_draws = NULL,
  scale = c("response", "linear"),
  probs = c(0.025, 0.975),
  xrange = NULL,
  xrange_search = NULL,
  parms_eval = FALSE,
  parms_method = "getPeak",
  idata_method = NULL,
  verbose = FALSE,
  fullframe = NULL,
  dummy_to_factor = NULL,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  funlist = NULL,
  xvar = NULL,
  difx = NULL,
  idvar = NULL,
  itransform = NULL,
  newdata_fixed = NULL,
  envir = NULL,
  ...
)

fitted_draws(model, ...)
```

Arguments

`model`	An object of class `bgmfit`.
`newdata`	An optional data frame for estimation. If `NULL` (default), `newdata` is retrieved from the `model`.
`resp`	A character string (default `NULL`) to specify the response variable when processing posterior draws for `univariate_by` and `multivariate` models. See `bsitar()` for details on `univariate_by` and `multivariate` models.
`dpar`	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
`ndraws`	A positive integer indicating the number of posterior draws to use in estimation. If `NULL` (default), all draws are used.
`draw_ids`	An integer specifying the specific posterior draw(s) to use in estimation (default `NULL`).
`re_formula`	Option to indicate whether or not to include individual/group-level effects in the estimation. When `NA` (default), individual-level effects are excluded, and population average growth parameters are computed. When `NULL`, individual-level effects are included in the computation, and the resulting growth parameters are individual-specific. In both cases (`NA` or `NULL`), continuous and factor covariates are appropriately included in the estimation. Continuous covariates are set to their means by default (see `numeric_cov_at` for details), while factor covariates remain unaltered, allowing for the estimation of covariate-specific population average and individual-specific growth parameters.
`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`.
`numeric_cov_at`	An optional (named list) argument to specify the value of continuous covariate(s). The default `NULL` option sets the continuous covariate(s) to their mean. Alternatively, a named list can be supplied to manually set these values. For example, `numeric_cov_at = list(xx = 2)` will set the continuous covariate variable 'xx' to 2. The argument `numeric_cov_at` is ignored when no continuous covariates are included in the model.
`levels_id`	An optional argument to specify the `ids` for the hierarchical model (default `NULL`). It is used only when the model is applied to data with three or more levels of hierarchy. For a two-level model, `levels_id` is automatically inferred from the model fit. For models with three or more levels, `levels_id` is inferred from the model fit under the assumption that hierarchy is specified from the lowest to the uppermost level, i.e., `id` followed by `study`, where `id` is nested within `study`. However, it is not guaranteed that `levels_id` is sorted correctly, so it is better to set it manually when fitting a model with three or more levels of hierarchy.
`avg_reffects`	An optional argument (default `NULL`) to calculate (marginal/average) curves and growth parameters, such as APGV and PGV. If specified, it must be a named list indicating the `over` (typically a level 1 predictor, such as age), `feby` (fixed effects, typically a factor variable), and `reby` (typically `NULL`, indicating that parameters are integrated over the random effects). For example, `avg_reffects = list(feby = 'study', reby = NULL, over = 'age')`.
`aux_variables`	An optional argument to specify the variable(s) that can be passed to the `ipts` argument (see below). This is useful when fitting location-scale models and measurement error models. If post-processing functions throw an error such as `variable 'x' not found in either 'data' or 'data2'`, consider using `aux_variables`.
`grid_add`	An optional argument to specify the variable(s) that can be passed to the `marginaleffects::datagrid()` This is useful when fitting location-scale models and measurement error models. If post-processing functions throw an error such as `variable 'x' not found in either 'data' or 'data2'`, consider using `grid_add`. Note that unlike `aux_variables` which are passed to the internal data functions such as `'get.newdata'`, the `grid_add` are passed to the `marginaleffects::datagrid()`.
`ipts`	An integer specifying the number of points for interpolating the predictor variable (e.g., age) to generate smooth curves for predictions and plots. This value is used as the `length.out` argument for `seq()`, controlling the smoothness of distance and velocity curves without altering the predictor range.

NULL **(the default)** Engages automatic behavior based on the dpar argument: it is internally set to 50 for the mean response (dpar = 'mu'), ensuring smooth curves, while for the distributional parameters (e.g., dpar = 'sigma'), no interpolation is performed.

An integer (e.g., 100) Explicitly sets the number of interpolation points.

FALSE Disables all interpolation, forcing predictions to be made only at the original data points of the predictor variable.

This argument affects the following post-processing functions: fitted_draws(), predict_draws(), growthparameters(), plot_curves(), get_predictions(), get_comparisons(), and get_growthparameters().

deriv	An integer indicating whether to estimate the distance curve or its derivative (velocity curve). The default deriv = 0 is for the distance curve, while deriv = 1 is for the velocity curve.
model_deriv	A logical value specifying whether to estimate the velocity curve from the derivative function or by differentiating the distance curve. Set model_deriv = TRUE for functions that require the velocity curve, such as growthparameters() and plot_curves(). Set it to NULL for functions that use the distance curve (i.e., fitted values), such as loo_validation() and plot_ppc().
summary	A logical value indicating whether only the estimate should be computed (TRUE), or whether the estimate along with SE and CI should be returned (FALSE, default). Setting summary to FALSE will increase computation time. Note that summary = FALSE is required to obtain correct estimates when re_formula = NULL.
robust	A logical value to specify the summary options. If FALSE (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 median absolute deviation (MAD) are applied instead. Ignored if summary is FALSE.
transform_draws	A function applied to individual draws from the posterior distribution before computing summaries (default NULL). The argument transform_draws is derived from the marginaleffects::predictions() function and should not be confused with the transform argument from the deprecated brms::posterior_predict() function. It's important to note that for both marginaleffects::predictions() and marginaleffects::avg_predictions(), the transform_draws argument takes precedence over the transform argument. Note that when transform_draws = NULL, an attempt is made to automatically set transform_draws = 'exp' for dpar = 'sigma'. User can set transform_draws = FALSE to turn off this automatic assignment of 'exp' to the transform_draws. It is also important to set transform_draws = FALSE when computing the first derivative (velocity) for dpar = 'sigma'.
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.
probs	The percentiles to be computed by the quantile function. Only used if summary is TRUE.

`xrange`	An integer to set the predictor range (e.g., age) when executing the interpolation via `ipts`. By default, `NULL` sets the individual-specific predictor range. Setting `xrange = 1` applies the same range for individuals within the same higher grouping variable (e.g., study). Setting `xrange = 2` applies an identical range across the entire sample. Alternatively, a numeric vector (e.g., `xrange = c(6, 20)`) can be provided to set the range within the specified values.
`xrange_search`	A vector of length two or a character string `'range'` to set the range of the predictor variable (`x`) within which growth parameters are searched. This is useful when there is more than one peak and the user wants to summarize the peak within a specified range of the `x` variable. The default value is `xrange_search = NULL`.
`parms_eval`	A logical value to specify whether or not to compute growth parameters on the fly. This is for internal use only and is mainly needed for compatibility across internal functions.
`parms_method`	A character string specifying the method used when evaluating `parms_eval`. The default method is `getPeak`, which uses the `sitar::getPeak()` function from the `sitar` package. Alternatively, `findpeaks` uses the `findpeaks` function from the `pracma` package. This parameter is for internal use and ensures compatibility across internal functions.
`idata_method`	A character string specifying the interpolation method (default `NULL`). The number of interpolation points is controlled by the `ipts` argument. Available options: • `'m1'`: Adapted from the **iapvbs** package (documented here). This method internally constructs the data frame based on the model configuration. *Note:* This method may fail if the model includes covariates (especially in `univariate_by` models). • `'m2'`: Based on the **JMbayes** package (documented here). This method uses the exact data frame from the model fit (`fit$data`) and is generally more robust. If `idata_method = NULL`, `'m2'` is automatically selected. It is recommended to use `'m2'` if `'m1'` encounters errors with covariate-dependent models.
`verbose`	A logical argument (default `FALSE`) to specify whether to print information collected during the setup of the object(s).
`fullframe`	A logical value indicating whether to return a `fullframe` object in which `newdata` is bound to the summary estimates. Note that `fullframe` cannot be used with `summary = FALSE`, and it is only applicable when `idata_method = 'm2'`. A typical use case is when fitting a `univariate_by` model. This option is mainly for internal use.
`dummy_to_factor`	A named list (default `NULL`) to convert dummy variables into a factor variable. The list must include the following elements: • `factor.dummy`: A character vector of dummy variables to be converted to factors. • `factor.name`: The name for the newly created factor variable (default is `'factor.var'` if `NULL`).

- `factor.level`: A vector specifying the factor levels. If `NULL`, levels are taken from `factor.dummy`. If `factor.level` is provided, its length must match `factor.dummy`.

`expose_function`
: A logical value or `NULL` (default `FALSE`). Controls whether Stan functions are exposed for post-processing.
  - `TRUE`: Explicitly exposes Stan functions saved from the model fit. This is required when calculating `fit criteria` or `bayes_R2` during model optimization.
  - `FALSE` (default): Stan functions are not exposed.
  - `NULL`: The setting is inherited from the original `bsitar()` model fit configuration.

  Note: In the `optimize_model()` function, the default is `NULL` (inheriting behavior), whereas other post-processing functions default to `FALSE`.

`usesavedfuns`
: A logical value (default `NULL`) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the `expose_functions` argument from the `bsitar()` call. Manual specification of `usesavedfuns` is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.

`clearenvfuns`
: A logical value indicating whether to clear the exposed Stan functions from the environment (`TRUE`) or not (`FALSE`). If `NULL`, `clearenvfuns` is set based on the value of `usesavedfuns`: `TRUE` if `usesavedfuns = TRUE`, or `FALSE` if `usesavedfuns = FALSE`.

`funlist`
: A list (default `NULL`) specifying function names. This is rarely needed, as required functions are typically retrieved automatically. A use case for `funlist` is when `sigma_formula`, `sigma_formula_gr`, or `sigma_formula_gr_str` use an external function (e.g., `poly(age)`). The `funlist` should include function names defined in the `globalenv()`. For functions needing both distance and velocity curves (e.g., `plot_curves(..., opt = 'dv')`), `funlist` must include two functions: one for the distance curve and one for the velocity curve.

`xvar`
: A character string (default `NULL`) specifying the `'x'` variable. Rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.

`difx`
: A character string (default `NULL`) specifying the `'x'` variable that should be used for manual differentiation of the distance curve. Internally, the `xvar` is set as `difx` if specified. The argument `difx` is evaluated only when `dpar = 'sigma'`, ignored otherwise. Note that argument `xvar` itself is rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.

`idvar`
: A character string (default `NULL`) specifying the `'id'` variable. Rarely used because `idvar` is inferred internally.

`itransform`
: A character string (default `NULL`) indicating the variables names that are reverse transformed. Options are `c("x", "y", "sigma")`. The `itransform` is primarily used to get the `xvar` variable at original scale i.e., `itransform = 'x'`. To turn of all transformations, use `itransform = ""`. when `itransform = NULL`, the appropriate transformation for `xvar` is selected automatically. Note that when no

match for `xvar` is found in the `data,frame`, the `itransform` will be ignored within the calling function, `'prepare_transformations()'`.

`newdata_fixed`	An indicator to specify whether to check data format and structure for the user provided `newdata`, and apply needed `prepare_data2` and `prepare_transformations` (`newdata_fixed = NULL`, default), return user provided `newdata` (`newdata = TRUE`) as it is without checking for the data format or applying `prepare_data2` and `prepare_transformations` (`newdata_fixed = 0`), check for the data format and if needed, prepare data format using `prepare_data2` (`newdata_fixed = 1`), or apply `prepare_transformations` only assuming that data format is correct (`newdata_fixed = 2`). It is strongly recommended that user either leave the `newdata = NULL` and `newdata_fixed = NULL` in which case data used in the model fitting is automatically retrieved and checked for the required data format and transformations, and if needed, `prepare_data2` and `prepare_transformations` are applied internally. The other flags provided for `newdata_fixed = 0, 1, 2` are mainly for the internal use during post-processing.
`envir`	The environment used for function evaluation. The default is `NULL`, which sets the environment to `parent.frame()`. Since most post-processing functions rely on **brms**, it is recommended to set `envir = globalenv()` or `envir = .GlobalEnv`, especially for derivatives like velocity curves.
`...`	Additional arguments passed to the `brms::fitted.brmsfit()` function. For details on available options, please refer to `brms::fitted.brmsfit()`.

Details

The **fitted_draws()** function computes the fitted values from the posterior draws. While the `brms::fitted.brmsfit()` function from the **brms** package can be used to obtain fitted (distance) values when the outcome (e.g., height) is untransformed, it returns fitted values on the log or square root scale if the outcome is transformed. In contrast, **fitted_draws()** returns fitted values on the original scale. Additionally, **fitted_draws()** computes the first derivative (velocity) on the original scale, after applying the necessary back-transformation. Apart from these differences, both functions—`brms::fitted.brmsfit()` and `fitted_draws()`—operate in the same manner, allowing users to specify all options available in `brms::fitted.brmsfit()`.

Value

An array of predicted mean response values when `summarise = FALSE`, or a `data.frame` when `summarise = TRUE`. For further details, refer to brms::fitted.brmsfit.

Author(s)

Satpal Sandhu <`satpal.sandhu@bristol.ac.uk`>

See Also

`brms::fitted.brmsfit()`

Examples

```
# Fit Bayesian SITAR model

# To avoid time-consuming model estimation, the Bayesian SITAR model fit to
# the 'berkeley_exdata' has been saved as an example fit ('berkeley_exfit').
# See 'bsitar' function for details on 'berkeley_exdata' and 'berkeley_exfit'.

# Check if the model fit object 'berkeley_exfit' exists
berkeley_exfit <- getNsObject(berkeley_exfit)

model <- berkeley_exfit

# Population average distance curve
fitted_draws(model, deriv = 0, re_formula = NA)

# Individual-specific distance curves
fitted_draws(model, deriv = 0, re_formula = NULL)

# Population average velocity curve
fitted_draws(model, deriv = 1, re_formula = NA)

# Individual-specific velocity curves
fitted_draws(model, deriv = 1, re_formula = NULL)
```

`getNsObject` *Retrieve Bayesian SITAR model object if it exists*

Description

This function checks if an object exists within a specified namespace and returns the object if it exists. The object must be provided as a symbol (not a character string). The function is designed to facilitate the retrieval of model or other objects from a specified environment or namespace. This function is mainly for internal purposes.

Usage

```
getNsObject(object, namespace = NULL, envir = NULL)
```

Arguments

`object`	A symbol representing the object to be retrieved. The input must be a symbol (i.e., not a character string) corresponding to an existing object within the specified namespace.
`namespace`	A character string specifying the namespace to check for the object. If the object exists within the given namespace, it will be returned.
`envir`	An environment in which to search for the object. If set to `NULL` (default), the function uses the global environment.

Value

The object of the same class as the input `object`, if it exists. If the object doesn't exist in the specified namespace or environment, an error is raised.

Author(s)

Satpal Sandhu <`satpal.sandhu@bristol.ac.uk`>

Examples

```
# Check whether model fit object 'berkeley_exfit' exists
 berkeley_exfit <- getNsObject(berkeley_exfit)
```

`get_comparisons.bgmfit` *Estimate and compare growth curves for the Bayesian SITAR model*

Description

The **get_comparisons()** function estimates and compares growth curves such as distance and velocity. This function is a wrapper around `marginaleffects::comparisons()` and `marginaleffects::avg_comparisons()`. The `marginaleffects::comparisons()` function computes unit-level (conditional) estimates, whereas `marginaleffects::avg_comparisons()` returns average (marginal) estimates. A detailed explanation is available here. Note that the **marginaleffects** package is highly flexible, and users are expected to have a strong understanding of its workings. Additionally, since the **marginaleffects** package is evolving rapidly, results from the current implementation should be considered experimental.

Usage

```
## S3 method for class 'bgmfit'
get_comparisons(
  model,
  resp = NULL,
  dpar = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  newdata = NULL,
  datagrid = NULL,
  re_formula = NA,
  newdata2 = NULL,
  allow_new_levels = FALSE,
  sample_new_levels = "gaussian",
  xrange = 1,
  digits = 2,
  numeric_cov_at = NULL,
  aux_variables = NULL,
  grid_add = NULL,
  levels_id = NULL,
  avg_reffects = NULL,
  idata_method = NULL,
  ipts = NULL,
  seed = 123,
  future = FALSE,
  future_session = "multisession",
  future_splits = TRUE,
  future_method = "future",
  future_re_expose = NULL,
  usedtplyr = FALSE,
  usecollapse = TRUE,
  cores = NULL,
  fullframe = FALSE,
  average = FALSE,
  plot = FALSE,
  mapping_facet = NULL,
  showlegends = NULL,
  variables = NULL,
  condition = NULL,
  deriv = 0,
  model_deriv = TRUE,
  method = "custom",
  method_call = NULL,
  marginals = NULL,
  pdrawso = FALSE,
  pdrawsp = FALSE,
  pdrawsh = FALSE,
  comparison = NULL,
  type = NULL,
  by = FALSE,
  conf_level = 0.95,
  transform = NULL,
  transform_draws = NULL,
  cross = FALSE,
  wts = NULL,
  hypothesis = NULL,
  equivalence = NULL,
  eps = NULL,
  constrats_by = NULL,
  constrats_at = NULL,
  reformat = NULL,
  estimate_center = NULL,
  estimate_interval = NULL,
  dummy_to_factor = NULL,
  verbose = FALSE,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  funlist = NULL,
  xvar = NULL,
  difx = NULL,
  idvar = NULL,
  itransform = NULL,
  newdata_fixed = NULL,
  envir = NULL,
  ...
)

get_comparisons(model, ...)

marginal_comparison(model, ...)

marginal_comparisons(model, ...)
```

Arguments

`model`	An object of class `bgmfit`.
`resp`	A character string (default `NULL`) to specify the response variable when processing posterior draws for `univariate_by` and `multivariate` models. See `bsitar()` for details on `univariate_by` and `multivariate` models.
`dpar`	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
`ndraws`	A positive integer indicating the number of posterior draws to use in estimation. If `NULL` (default), all draws are used.
`draw_ids`	An integer specifying the specific posterior draw(s) to use in estimation (default `NULL`).
`newdata`	An optional data frame for estimation. If `NULL` (default), `newdata` is retrieved from the `model`.
`datagrid`	A data frame or named list for setting up a custom grid of predictor values to evaluate the quantities of interest. If `NULL` (default), no custom grid is used. The grid can be constructed using `marginaleffects::datagrid()`. If `datagrid = list()`, essential arguments such as `model` and `newdata` are inferred automatically from the respective arguments in the model fit.
`re_formula`	Option to indicate whether or not to include individual/group-level effects in the estimation. When `NA` (default), individual-level effects are excluded, and population average growth parameters are computed. When `NULL`, individual-level effects are included in the computation, and the resulting growth parameters are individual-specific. In both cases (`NA` or `NULL`), continuous and factor covariates are appropriately included in the estimation. Continuous covariates are set to

their means by default (see `numeric_cov_at` for details), while factor covariates remain unaltered, allowing for the estimation of covariate-specific population average and individual-specific growth parameters.

`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`.
`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.
`xrange`	An integer to set the predictor range (e.g., age) when executing the interpolation via `ipts`. By default, `NULL` sets the individual-specific predictor range. Setting `xrange = 1` applies the same range for individuals within the same higher grouping variable (e.g., study). Setting `xrange = 2` applies an identical range across the entire sample. Alternatively, a numeric vector (e.g., `xrange = c(6, 20)`) can be provided to set the range within the specified values.
`digits`	An integer (default 2) for rounding the estimates to the specified number of decimal places using `base::round()`.
`numeric_cov_at`	An optional (named list) argument to specify the value of continuous covariate(s). The default `NULL` option sets the continuous covariate(s) to their mean. Alternatively, a named list can be supplied to manually set these values. For example, `numeric_cov_at = list(xx = 2)` will set the continuous covariate variable 'xx' to 2. The argument `numeric_cov_at` is ignored when no continuous covariates are included in the model.
`aux_variables`	An optional argument to specify the variable(s) that can be passed to the `ipts` argument (see below). This is useful when fitting location-scale models and measurement error models. If post-processing functions throw an error such as `variable 'x' not found in either 'data' or 'data2'`, consider using `aux_variables`.
`grid_add`	An optional argument to specify the variable(s) that can be passed to the `marginaleffects::datagrid()` This is useful when fitting location-scale models and measurement error models. If post-processing functions throw an error such as `variable 'x' not found in either 'data' or 'data2'`, consider using `grid_add`. Note that unlike `aux_variables` which are passed to the internal data functions such as `'get.newdata'`, the `grid_add` are passed to the `marginaleffects::datagrid()`.

`levels_id`	An optional argument to specify the `ids` for the hierarchical model (default `NULL`). It is used only when the model is applied to data with three or more levels of hierarchy. For a two-level model, `levels_id` is automatically inferred from the model fit. For models with three or more levels, `levels_id` is inferred from the model fit under the assumption that hierarchy is specified from the lowest to the uppermost level, i.e., `id` followed by `study`, where `id` is nested within `study`. However, it is not guaranteed that `levels_id` is sorted correctly, so it is better to set it manually when fitting a model with three or more levels of hierarchy.
`avg_reffects`	An optional argument (default `NULL`) to calculate (marginal/average) curves and growth parameters, such as APGV and PGV. If specified, it must be a named list indicating the `over` (typically a level 1 predictor, such as age), `feby` (fixed effects, typically a factor variable), and `reby` (typically `NULL`, indicating that parameters are integrated over the random effects). For example, `avg_reffects = list(feby = 'study', reby = NULL, over = 'age')`.
`idata_method`	A character string specifying the interpolation method (default `NULL`). The number of interpolation points is controlled by the `ipts` argument. Available options: • `'m1'`: Adapted from the **iapvbs** package (documented here). This method internally constructs the data frame based on the model configuration. *Note:* This method may fail if the model includes covariates (especially in `univariate_by` models). • `'m2'`: Based on the **JMbayes** package (documented here). This method uses the exact data frame from the model fit (`fit$data`) and is generally more robust. If `idata_method = NULL`, `'m2'` is automatically selected. It is recommended to use `'m2'` if `'m1'` encounters errors with covariate-dependent models.
`ipts`	An integer specifying the number of points for interpolating the predictor variable (e.g., age) to generate smooth curves for predictions and plots. This value is used as the `length.out` argument for `seq()`, controlling the smoothness of distance and velocity curves without altering the predictor range. `NULL` **(the default)** Engages automatic behavior based on the `dpar` argument: it is internally set to `50` for the mean response (`dpar = 'mu'`), ensuring smooth curves, while for the distributional parameters (e.g., `dpar = 'sigma'`), no interpolation is performed. **An integer (e.g.,** `100`**)** Explicitly sets the number of interpolation points. `FALSE` Disables all interpolation, forcing predictions to be made only at the original data points of the predictor variable. This argument affects the following post-processing functions: `fitted_draws()`, `predict_draws()`, `growthparameters()`, `plot_curves()`, `get_predictions()`, `get_comparisons()`, and `get_growthparameters()`.
`seed`	An integer (default `123`) that is passed to the estimation method to ensure reproducibility.
`future`	A logical value (default `FALSE`) indicating whether to perform parallel computations. If `TRUE`, posterior summaries are computed in parallel using `future.apply::future_sapply()`.

`future_session` A character string or a named list specifying the parallel plan when `future = TRUE`.

- **Character string**: Defaults to `"multisession"`. Use `"multicore"` for forking (not supported on Windows).
- **Named list**: Useful for advanced plans like `future.mirai::mirai_cluster()`. The list must contain:
  - `future_session`: The planner function (e.g., `mirai_cluster`).
  - Additional named arguments passed to the planner (e.g., `daemons` for `mirai::daemons()`).

  Example: `list(future_session = mirai_cluster, daemons = list(...))`.

`future_splits` A list, numeric vector, or logical value (default `TRUE`). Controls how posterior draws are partitioned into smaller subsets for parallel computation. This helps manage memory and improve performance, particularly on Linux when using `future::plan("multicore")`.

Input options:

- **List**: (e.g., `list(1:6, 7:10)`). Each element is a sequence of numbers passed to `draw_ids` to process in a separate chunk.
- **Numeric vector of length 2**: (e.g., `c(100, 4)`). The first element is the total number of draws, and the second is the number of splits. Indices are generated via `parallel::splitIndices()`.
- **Numeric vector of length 1**: Specifies the total number of draws. Splits are calculated automatically.
- `TRUE`: Automatically creates splits based on `ndraws` and `cores`. If both `ndraws` and `draw_ids` are specified, `draw_ids` takes precedence. This is consistent with post-processing functions in the `bsitar` and `brms` packages.
- `FALSE`: Splitting is disabled.

Note: On Windows with `future::plan("multisession")`, background R processes may not free resources automatically. If needed, use `installr::kill_all_Rscript_s()` to terminate them.

`future_method` A character string (default `'future'`) specifying the parallel computation backend. Options include:

- `'future'`: Uses `future.apply::future_lapply()` for execution.
- `'dofuture'`: Uses `foreach::foreach()` via the `doFuture` package.

`future_re_expose`

A logical value (default `NULL`) indicating whether to re-expose internal `Stan` functions when `future = TRUE`. This is critical when `future::plan()` is set to `"multisession"`, as compiled C++ functions cannot be exported across distinct R sessions.

- If `NULL` (default), it is automatically set to `TRUE` when the plan is `"multisession"`.
- Explicitly setting this to `TRUE` is recommended for improved performance during parallel execution.

`usedtplyr` A logical (default `FALSE`) indicating whether to use the **dtplyr** package for summarizing the draws. This package uses **data.table** as a back-end. It is useful when the data has a large number of observations. For typical use cases, it does not make a significant performance difference. The `usedtplyr` argument is evaluated only when `method = 'custom'`.

`usecollapse`	A logical (default `FALSE`) to indicate whether to use the **collapse** package for summarizing the draws.
`cores`	An integer specifying the number of cores for parallel execution. • If `NULL` (default), the number of cores is set to `future::availableCores() - 1`. • On non-Windows systems, this can be controlled globally via the `mc.cores` option.
`fullframe`	A logical value indicating whether to return a `fullframe` object in which `newdata` is bound to the summary estimates. Note that `fullframe` cannot be used with `summary = FALSE`, and it is only applicable when `idata_method = 'm2'`. A typical use case is when fitting a `univariate_by` model. This option is mainly for internal use.
`average`	A logical indicating whether to call `marginaleffects::comparisons()` (if `FALSE`) or `marginaleffects::avg_comparisons()` (if `TRUE`). Default is `FALSE`.
`plot`	A logical indicating whether to plot the comparisons using `marginaleffects::plot_comparisons()`. Default is `FALSE`.
`mapping_facet`	A named list that can be used to pass the aesthetic `mapping` and `facet` arguments to the `ggplot2` object (default `NULL`). The main use of this is to get overlay line plot instead of separate line plot for each `id` variable. This can also be used to remove a layer. For example, if user wants to remove the confidence intervals band from the lines i.e., `geom_ribbon`, then include `rm_geom_ribbon = TRUE`. Note the prefix `'rm_'`. This is a general approach in which any layer name with prefix `'rm_'` included in the `mapping_facet` will be removed. An example is shown below: `mapping_facet = list(group = 'id', colour = 'id', facet_wrap = c('study'), rm_geom_ribbon = TRUE)`.
`showlegends`	A logical value to specify whether to show legends (`TRUE`) or not (`FALSE`). If `NULL` (default), the value of `showlegends` is internally set to `TRUE` if `re_formula = NA`, and `FALSE` if `re_formula = NULL`.
`variables`	A named list specifying the level 1 predictor, such as `age` or `time`, used for estimating growth parameters in the current use case. The `variables` list is set via the `esp` argument (default value is `1e-6`). If `variables` is `NULL`, the relevant information is retrieved internally from the `model`. Alternatively, users can define `variables` as a named list, e.g., `variables = list('x' = 1e-6)` where `'x'` is the level 1 predictor. By default, `variables = list('age' = 1e-6)` in the **marginaleffects** package, as velocity is usually computed by differentiating the distance curve using the `dydx` approach. When using this default, the argument `deriv` is automatically set to `0` and `model_deriv` to `FALSE`. If parameters are to be estimated based on the model's first derivative, `deriv` must be set to `1` and `variables` will be defined as `variables = list('age' = 0)`. Note that if the default behavior is used (`deriv = 0` and `variables = list('x' = 1e-6)`), additional arguments cannot be passed to `variables`. In contrast, when using an alternative approach (`deriv = 0` and `variables = list('x' = 0)`), additional options can be passed to the `marginaleffects::comparisons()` and `marginaleffects::avg_comparisons()` functions.
`condition`	Conditional slopes

- Character vector (max length 4): Names of the predictors to display.
- Named list (max length 4): List names correspond to predictors. List elements can be:
  - Numeric vector
  - Function which returns a numeric vector or a set of unique categorical values
  - Shortcut strings for common reference values: "minmax", "quartile", "threenum"
- 1: x-axis. 2: color/shape. 3: facet (wrap if no fourth variable, otherwise cols of grid). 4: facet (rows of grid).
- Numeric variables in positions 2 and 3 are summarized by Tukey's five numbers `?stats::fivenum`.

`deriv` A numeric value indicating whether to estimate parameters based on the differentiation of the distance curve or the model's first derivative.

`model_deriv` A logical value specifying whether to estimate the velocity curve from the derivative function or by differentiating the distance curve. Set `model_deriv = TRUE` for functions that require the velocity curve, such as `growthparameters()` and `plot_curves()`. Set it to `NULL` for functions that use the distance curve (i.e., fitted values), such as `loo_validation()` and `plot_ppc()`.

`method` A character string specifying whether to compute comparisons using the **marginaleffects** machinery (`method = 'pkg'`) or via custom functions for efficiency (`method = 'custom'`). Default is `'pkg'`. The `'custom'` method is useful when testing hypotheses and should be used cautiously.

`method_call` A character string specifying whether to compute comparisons using `marginaleffects::predictions(` (`method_call = 'predictions'`) or `marginaleffects::comparisons()` (`method_call = 'comparisons'`). The `'method_call'` is evaluated only when `method = 'custom'` ignored otherwise. Default `method_call = NULL` allows automatic selection of the `'method_call'` depending on whether comparisons are for slopes or predictions.

`marginals` A `list`, `data.frame`, or `tibble` of velocity returned by the **marginaleffects** functions (default `NULL`). This is only evaluated when `method = 'custom'`. The `marginals` can be the output from **marginaleffects** functions or posterior draws from `marginaleffects::posterior_draws()`. The `marginals` argument is primarily used for internal purposes only.

`pdrawso` A character string (default `FALSE`) to indicate whether to return the original posterior draws for parameters. Options include:

- `'return'`: returns the original posterior draws,
- `'add'`: adds the original posterior draws to the outcome.

When `pdrawso = TRUE`, the default behavior is `pdrawso = 'return'`. Note that the posterior draws are returned before calling `marginaleffects::posterior_draws()`.

`pdrawsp` A character string (default `FALSE`) to indicate whether to return the posterior draws for parameters. Options include:

- `'return'`: returns the posterior draws for parameters,
- `'add'`: adds the posterior draws to the outcome.

When `pdrawsp` = `TRUE`, the default behavior is `pdrawsp = 'return'`. The `pdrawsp` represent the parameter estimates for each of the posterior samples, and the summary of these are the estimates returned.

`pdrawsh`
: A character string (default `FALSE`) to indicate whether to return the posterior draws for parameter contrasts. Options include:

- `'return'`: returns the posterior draws for contrasts.

The summary of posterior draws for parameters is the default returned object. The `pdrawsh` represent the contrast estimates for each of the posterior samples, and the summary of these are the contrast returned.

`comparison`
: A character string specifying the comparison type for growth parameter estimation. Options are `'difference'` and `'differenceavg'`. This argument sets up the internal function for estimating parameters using `sitar::getPeak()`, `sitar::getTakeoff()`, and `sitar::getTrough()` functions. These options are restructured according to the user-specified `hypothesis` argument.

`type`
: string indicates the type (scale) of the predictions used to compute contrasts or slopes. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When `type` is `NULL`, the first entry in the error message is used by default. See the Type section in the documentation below.

`by`
: Aggregate unit-level estimates (aka, marginalize, average over). Valid inputs:

- `FALSE`: return the original unit-level estimates.
- `TRUE`: aggregate estimates for each term.
- Character vector of column names in `newdata` or in the data frame produced by calling the function without the `by` argument.
- Data frame with a `by` column of group labels, and merging columns shared by `newdata` or the data frame produced by calling the same function without the `by` argument.
- See examples below.
- For more complex aggregations, you can use the `FUN` argument of the `hypotheses()` function. See that function's documentation and the Hypothesis Test vignettes on the `marginaleffects` website.

`conf_level`
: numeric value between 0 and 1. Confidence level to use to build a confidence interval.

`transform`
: string or function. Transformation applied to unit-level estimates and confidence intervals just before the function returns results. Functions must accept a vector and return a vector of the same length. Support string shortcuts: "exp", "ln"

`transform_draws`
: A function applied to individual draws from the posterior distribution before computing summaries (default `NULL`). The argument `transform_draws` is derived from the `marginaleffects::predictions()` function and should not be confused with the `transform` argument from the deprecated `brms::posterior_predict()` function. It's important to note that for both `marginaleffects::predictions()` and `marginaleffects::avg_predictions()`, the `transform_draws` argument takes precedence over the `transform` argument. Note that when `transform_draws`

= NULL, an attempt is made to automatically set transform_draws = 'exp' for dpar = 'sigma'. User can set transform_draws = FALSE to turn off this automatic assignment of 'exp' to the transform_draws. It is also important to set transform_draws = FALSE when computing the first derivative (velocity) for dpar = 'sigma'.

cross
- FALSE: Contrasts represent the change in adjusted predictions when one predictor changes and all other variables are held constant.
- TRUE: Contrasts represent the changes in adjusted predictions when all the predictors specified in the variables argument are manipulated simultaneously (a "cross-contrast").

wts
logical, string or numeric: weights to use when computing average predictions, contrasts or slopes. These weights only affect the averaging in avg_*() or with the by argument, and not unit-level estimates. See ?weighted.mean

- string: column name of the weights variable in newdata. When supplying a column name to wts, it is recommended to supply the original data (including the weights variable) explicitly to newdata.
- numeric: vector of length equal to the number of rows in the original data or in newdata (if supplied).
- FALSE: Equal weights.
- TRUE: Extract weights from the fitted object with insight::find_weights() and use them when taking weighted averages of estimates. Warning: newdata=datagrid() returns a single average weight, which is equivalent to using wts=FALSE

hypothesis
specify a hypothesis test or custom contrast using a number , formula, string equation, vector, matrix, or function.

- Number: The null hypothesis used in the computation of Z and p (before applying transform).
- String: Equation to specify linear or non-linear hypothesis tests. Two-tailed tests must include an equal = sign. One-tailed tests must start with < or >. If the terms in coef(object) uniquely identify estimates, they can be used in the formula. Otherwise, use b1, b2, etc. to identify the position of each parameter. The b* wildcard can be used to test hypotheses on all estimates. When the hypothesis string represents a two-sided equation, the estimate column holds the value of the left side minus the right side of the equation. If a named vector is used, the names are used as labels in the output. Examples:
  - hp = drat
  - hp + drat = 12
  - b1 + b2 + b3 = 0
  - b* / b1 = 1
  - <= 0
  - >= -3.5
  - b1 >= 10
- Formula: lhs ~ rhs | group
  - lhs
    * ratio (null = 1)

        * `difference` (null = 0)
        * Leave empty for default value
    - `rhs`
        * `pairwise` and `revpairwise`: pairwise differences between estimates in each row.
        * `reference`: differences between the estimates in each row and the estimate in the first row.
        * `sequential`: difference between an estimate and the estimate in the next row.
        * `meandev`: difference between an estimate and the mean of all estimates.
        * `meanotherdev`: difference between an estimate and the mean of all other estimates, excluding the current one.
        * `poly`: polynomial contrasts, as computed by the `stats::contr.poly()` function.
        * `helmert`: Helmert contrasts, as computed by the `stats::contr.helmert()` function. Contrast 2nd level to the first, 3rd to the average of the first two, and so on.
        * `trt_vs_ctrl`: difference between the mean of estimates (except the first) and the first estimate.
        * `I(fun(x))`: custom function to manipulate the vector of estimates `x`. The function `fun()` can return multiple (potentially named) estimates.
    - `group` (optional)
        * Column name of `newdata`. Conduct hypothesis tests withing subsets of the data.
    - Examples:
        * `~ poly`
        * `~ sequential | groupid`
        * `~ reference`
        * `ratio ~ pairwise`
        * `difference ~ pairwise | groupid`
        * `~ I(x - mean(x)) | groupid`
        * `~ I(\(x) c(a = x[1], b = mean(x[2:3]))) | groupid`
- Matrix or Vector: Each column is a vector of weights. The the output is the dot product between these vectors of weights and the vector of estimates. The matrix can have column names to label the estimates.
- Function:
    - Accepts an argument `x`: object produced by a `marginaleffects` function or a data frame with column `rowid` and `estimate`
    - Returns a data frame with columns `term` and `estimate` (mandatory) and `rowid` (optional).
    - The function can also accept optional input arguments: `newdata`, `by`, `draws`.

    - This function approach will not work for Bayesian models or with bootstrapping. In those cases, it is easy to use `get_draws()` to extract and manipulate the draws directly.
- See the Examples section below and the vignette: `https://marginaleffects.com/chapters/hypothesis.html`
- Warning: When calling `predictions()` with `type="invlink(link)"` (the default in some models), `hypothesis` is tested and p values are computed on the link scale.

`equivalence` Numeric vector of length 2: bounds used for the two-one-sided test (TOST) of equivalence, and for the non-inferiority and non-superiority tests. For bayesian models, this report the proportion of posterior draws in the interval and the ROPE. See Details section below.

`eps` NULL or numeric value which determines the step size to use when calculating numerical derivatives: (f(x+eps)-f(x))/eps. When `eps` is `NULL`, the step size is 0.0001 multiplied by the difference between the maximum and minimum values of the variable with respect to which we are taking the derivative. Changing `eps` may be necessary to avoid numerical problems in certain models.

`constrats_by` A character vector (default `NULL`) specifying the variables by which hypotheses should be tested (e.g., for post-draw comparisons). These variables should be a subset of the variables in the by argument of `marginaleffects::comparisons()`.

`constrats_at` A character vector (default `NULL`) specifying the values at which hypotheses should be tested. Useful for large estimates to limit testing to fewer rows.

`reformat` A logical (default `TRUE`) to reformat the output from `marginaleffects::comparisons()` as a data.frame, with column names such as `conf.low` as `Q2.5`, `conf.high` as `Q97.5`, and dropping unnecessary columns like `term`, `contrast`, and `predicted`.

`estimate_center`

A character string (default `NULL`) specifying how to center estimates: either `'mean'` or `'median'`. This option sets the global options as follows: `options("marginaleffects_poste` `= "mean")` or `options("marginaleffects_posterior_center" = "median")`. These global options are restored upon function exit using `base::on.exit()`.

`estimate_interval`

A character string (default `NULL`) to specify the type of credible intervals: `'eti'` for equal-tailed intervals or `'hdi'` for highest density intervals. This option sets the global options as follows: `options("marginaleffects_posterior_interval" = "eti")` or `options("marginaleffects_posterior_interval" = "hdi")`, and is restored on exit using `base::on.exit()`.

`dummy_to_factor`

A named list (default `NULL`) to convert dummy variables into a factor variable. The list must include the following elements:

- `factor.dummy`: A character vector of dummy variables to be converted to factors.
- `factor.name`: The name for the newly created factor variable (default is `'factor.var'` if `NULL`).
- `factor.level`: A vector specifying the factor levels. If `NULL`, levels are taken from `factor.dummy`. If `factor.level` is provided, its length must match `factor.dummy`.

`verbose` A logical argument (default `FALSE`) to specify whether to print information collected during the setup of the object(s).

`expose_function`

A logical value or `NULL` (default `FALSE`). Controls whether Stan functions are exposed for post-processing.

- `TRUE`: Explicitly exposes Stan functions saved from the model fit. This is required when calculating `fit criteria` or `bayes_R2` during model optimization.
- `FALSE` (default): Stan functions are not exposed.
- `NULL`: The setting is inherited from the original `bsitar()` model fit configuration.

Note: In the `optimize_model()` function, the default is `NULL` (inheriting behavior), whereas other post-processing functions default to `FALSE`.

`usesavedfuns` A logical value (default `NULL`) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the `expose_functions` argument from the `bsitar()` call. Manual specification of `usesavedfuns` is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.

`clearenvfuns` A logical value indicating whether to clear the exposed Stan functions from the environment (`TRUE`) or not (`FALSE`). If `NULL`, `clearenvfuns` is set based on the value of `usesavedfuns`: `TRUE` if `usesavedfuns = TRUE`, or `FALSE` if `usesavedfuns = FALSE`.

`funlist` A list (default `NULL`) specifying function names. This is rarely needed, as required functions are typically retrieved automatically. A use case for `funlist` is when `sigma_formula`, `sigma_formula_gr`, or `sigma_formula_gr_str` use an external function (e.g., `poly(age)`). The `funlist` should include function names defined in the `globalenv()`. For functions needing both distance and velocity curves (e.g., `plot_curves(..., opt = 'dv')`), `funlist` must include two functions: one for the distance curve and one for the velocity curve.

`xvar` A character string (default `NULL`) specifying the `'x'` variable. Rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.

`difx` A character string (default `NULL`) specifying the `'x'` variable that should be used for manual differentiation of the distance curve. Internally, the `xvar` is set as `difx` if specified. The argument `difx` is evaluated only when `dpar = 'sigma'`, ignored otherwise. Note that argument `xvar` itself is rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.

`idvar` A character string (default `NULL`) specifying the `'id'` variable. Rarely used because `idvar` is inferred internally.

`itransform` A character string (default `NULL`) indicating the variables names that are reverse transformed. Options are `c("x", "y", "sigma")`. The `itransform` is primarily used to get the `xvar` variable at original scale i.e., `itransform = 'x'`. To turn of all transformations, use `itransform = ""`. when `itransform = NULL`, the appropriate transformation for `xvar` is selected automatically. Note that when no

	match for `xvar` is found in the `data,frame`, the `itransform` will be ignored within the calling function, `'prepare_transformations()'`.
`newdata_fixed`	An indicator to specify whether to check data format and structure for the user provided `newdata`, and apply needed `prepare_data2` and `prepare_transformations` (`newdata_fixed = NULL`, default), return user provided `newdata` (`newdata = TRUE`) as it is without checking for the data format or applying `prepare_data2` and `prepare_transformations` (`newdata_fixed = 0`), check for the data format and if needed, prepare data format using `prepare_data2` (`newdata_fixed = 1`), or apply `prepare_transformations` only assuming that data format is correct (`newdata_fixed = 2`). It is strongly recommended that user either leave the `newdata = NULL` and `newdata_fixed = NULL` in which case data used in the model fitting is automatically retrieved and checked for the required data format and transformations, and if needed, `prepare_data2` and `prepare_transformations` are applied internally. The other flags provided for `newdata_fixed = 0, 1, 2` are mainly for the internal use during post-processing.
`envir`	The environment used for function evaluation. The default is `NULL`, which sets the environment to `parent.frame()`. Since most post-processing functions rely on **brms**, it is recommended to set `envir = globalenv()` or `envir = .GlobalEnv`, especially for derivatives like velocity curves.
`...`	Additional arguments passed to the `brms::fitted.brmsfit()` and `brms::predict()` functions.

Value

A data frame with estimates and confidence intervals for the specified parameters, or a list when `method = 'custom'` and `hypothesis` is not `NULL`.

Author(s)

Satpal Sandhu <`satpal.sandhu@bristol.ac.uk`>

References

There are no references for Rd macro `\insertAllCites` on this help page.

See Also

`marginaleffects::comparisons()`, `marginaleffects::avg_comparisons()`, `marginaleffects::plot_comparisons`

Examples

```
# Fit Bayesian SITAR model

# To avoid mode estimation which takes time, the Bayesian SITAR model fit to
# the 'berkeley_exdata' has been saved as an example fit ('berkeley_exfit').
# See 'bsitar' function for details on 'berkeley_exdata' and 'berkeley_exfit'.

# Note: Since no covariates are included, the 'get_comparisons'
# function is being shown here as a dummy example. In practice, comparisons
# may not make sense without relevant covariates.

# Check and confirm whether model fit object 'berkeley_exfit' exists
berkeley_exfit <- getNsObject(berkeley_exfit)

model <- berkeley_exfit

# Call get_comparisons to demonstrate the function
# Note: since model has no covariate, the example is for illustration purposes
# Comparisons at 1 SD of age
get_comparisons(model, variables = list(age = "sd"),
re_formula = NA, draw_ids = 1:2)

# Comparisons between individuals
get_comparisons(model, variables = list(id = "sequential"),
re_formula = NULL, draw_ids = 1:2)
```

---

get_growthparameters.bgmfit

Estimate and compare growth parameters for the Bayesian SITAR model

---

Description

The **get_growthparameters()** function estimates and compares growth parameters, such as peak growth velocity and the age at peak growth velocity. This function serves as a wrapper around `marginaleffects::comparisons()` and `marginaleffects::avg_comparisons()`. The `marginaleffects::comparison` function computes unit-level (conditional) estimates, whereas `marginaleffects::avg_comparisons()` returns average (marginal) estimates. A detailed explanation is available here. Note that for the current use case— estimating and comparing growth parameters—the arguments `variables` and `comparison` in `marginaleffects::comparisons()` and `marginaleffects::avg_comparisons()` are modified (see below). Furthermore, comparisons of growth parameters are performed via the `hypothesis` argument of both the `marginaleffects::comparisons()` and `marginaleffects::avg_comparisons()` functions. Please note that the **marginaleffects** package is highly flexible, and users are expected to have a strong understanding of its workings. Additionally, since the **marginaleffects** package is rapidly evolving, results obtained from the current implementation should be considered experimental.

Usage

```
## S3 method for class 'bgmfit'
get_growthparameters(
  model,
  resp = NULL,
  dpar = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  newdata = NULL,
  datagrid = NULL,
  re_formula = NA,
  newdata2 = NULL,
  allow_new_levels = FALSE,
  sample_new_levels = "gaussian",
  parameter = NULL,
  xrange = 1,
  acg_velocity = 0.1,
  digits = 2,
  numeric_cov_at = NULL,
  aux_variables = NULL,
  grid_add = NULL,
  levels_id = NULL,
  avg_reffects = NULL,
  idata_method = NULL,
  ipts = NULL,
  seed = 123,
  future = FALSE,
  future_session = "multisession",
  future_splits = TRUE,
  future_method = "future",
  future_re_expose = NULL,
  usedtplyr = FALSE,
  usecollapse = TRUE,
  parallel = FALSE,
  cores = NULL,
  fullframe = FALSE,
  average = FALSE,
  plot = FALSE,
  mapping_facet = NULL,
  showlegends = NULL,
  variables = NULL,
  condition = NULL,
  deriv = NULL,
  model_deriv = NULL,
  method = "custom",
  marginals = NULL,
  preparms = NULL,
  pdraws = FALSE,
  pdrawso = FALSE,
  pdrawsp = FALSE,
  pdrawsh = FALSE,
  comparison = "difference",
  type = NULL,
  by = FALSE,
  bys = NULL,
  conf_level = 0.95,
  transform = NULL,
  transform_draws = NULL,
  cross = FALSE,
  wts = NULL,
  hypothesis = NULL,
  equivalence = NULL,
  equivalence_test = NULL,
  p_direction = NULL,
  eps = NULL,
  constrats_by = FALSE,
  constrats_at = FALSE,
  constrats_subset = FALSE,
  reformat = NULL,
  estimate_center = NULL,
  estimate_interval = NULL,
  dummy_to_factor = NULL,
  verbose = FALSE,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  funlist = NULL,
  xvar = NULL,
  difx = NULL,
  idvar = NULL,
  itransform = NULL,
  newdata_fixed = NULL,
  envir = NULL,
  ...
)

get_growthparameters(model, ...)

get_gp(model, ...)

growthparameters_comparison(model, ...)

marginal_growthparameters(model, ...)
```

Arguments

model	An object of class bgmfit.
resp	A character string (default NULL) to specify the response variable when processing posterior draws for univariate_by and multivariate models. See bsitar() for details on univariate_by and multivariate models.
dpar	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.

`ndraws`	A positive integer indicating the number of posterior draws to use in estimation. If `NULL` (default), all draws are used.
`draw_ids`	An integer specifying the specific posterior draw(s) to use in estimation (default `NULL`).
`newdata`	An optional data frame for estimation. If `NULL` (default), `newdata` is retrieved from the `model`.
`datagrid`	A grid of user-specified values to be used in the `newdata` argument of various functions in the **marginaleffects** package. This allows you to define the regions of the predictor space where you want to evaluate the quantities of interest. See `marginaleffects::datagrid()` for more details. By default, the `datagrid` is set to `NULL`, meaning no custom grid is constructed. To set a custom grid, the argument should either be a data frame created using `marginaleffects::datagrid()`, or a named list, which is internally used for constructing the grid. For convenience, you can also pass an empty list `datagrid = list()`, in which case essential arguments like `model` and `newdata` are inferred from the respective arguments specified elsewhere. Additionally, the first-level predictor (such as age) and any covariates included in the model (e.g., gender) are automatically inferred from the `model` object.
`re_formula`	Option to indicate whether or not to include individual/group-level effects in the estimation. When `NA` (default), individual-level effects are excluded, and population average growth parameters are computed. When `NULL`, individual-level effects are included in the computation, and the resulting growth parameters are individual-specific. In both cases (`NA` or `NULL`), continuous and factor covariates are appropriately included in the estimation. Continuous covariates are set to their means by default (see `numeric_cov_at` for details), while factor covariates remain unaltered, allowing for the estimation of covariate-specific population average and individual-specific growth parameters.
`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`.
`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.

`parameter` A single character string or character vector specifying the growth parameter(s) to estimate. Must be one of the following options, ordered by growth curve progression:

- `'apgv'`: Age at peak growth velocity - age corresponding to the peak growth rate (i.e. maximum growth velocity) during adolescent growth spurt (default when `NULL`)
- `'pgv'`: Peak growth velocity - maximum growth rate during adolescent growth spurt
- `'atgv'`: Age at takeoff growth velocity - age where initial rapid growth begins accelerating toward peak
- `'tgv'`: Takeoff growth velocity - initial high growth rate at curve start
- `'acgv'`: Age at cessation growth velocity - age where growth rate approaches near zero and growth effectively stops
- `'cgv'`: Cessation growth velocity - final low growth rate approaching maturity

```
\item \code{'spgv'}: Size (length/height) at peak growth velocity age,
\code{'apgv'}
\item \code{'stgv'}: Size at takeoff growth velocity age, \code{'atgv'}
\item \code{'scgv'}: Size at cessation growth velocity age, \code{'acgv'}

\item \code{'satXX'}: Size at specific age XX years of predictor
\code{xvar} (e.g., \code{'sat15'} = size at 15 years, \code{'sat12.5'} =
size at 12.5 years)

\item \code{'all'}: All six primary velocity/age parameters
(\code{'apgv','pgv','atgv','tgv','acgv','cgv'}). Cannot be used when
\code{by = TRUE}.
```

Multiple parameters can be requested simultaneously as a vector, e.g., `c('apgv', 'pgv', 'spgv')`. Custom `'satXX'` format requires no spaces between `'sat'` prefix and numeric age.

`xrange` An integer to set the predictor range (e.g., age) when executing the interpolation via `ipts`. By default, `NULL` sets the individual-specific predictor range. Setting `xrange = 1` applies the same range for individuals within the same higher grouping variable (e.g., study). Setting `xrange = 2` applies an identical range across the entire sample. Alternatively, a numeric vector (e.g., `xrange = c(6, 20)`) can be provided to set the range within the specified values.

`acg_velocity` A real number specifying the percentage of peak growth velocity to be used as the cessation velocity when estimating the `cgv` and `acgv` growth parameters. The `acg_velocity` should be greater than `0` and less than `1`. The default value of `acg_velocity = 0.10` indicates that 10 percent of the peak growth velocity will be used to calculate the cessation growth velocity and the corresponding age at cessation velocity. For example, if the peak growth velocity estimate is `10 mm/year`, then the cessation growth velocity will be `1 mm/year`.

`digits` An integer (default 2) specifying the number of decimal places to round the estimated growth parameters. The `digits` value is passed to the `base::round()` function.

`numeric_cov_at`	An optional (named list) argument to specify the value of continuous covariate(s). The default `NULL` option sets the continuous covariate(s) to their mean. Alternatively, a named list can be supplied to manually set these values. For example, `numeric_cov_at = list(xx = 2)` will set the continuous covariate variable 'xx' to 2. The argument `numeric_cov_at` is ignored when no continuous covariates are included in the model.
`aux_variables`	An optional argument to specify the variable(s) that can be passed to the `ipts` argument (see below). This is useful when fitting location-scale models and measurement error models. If post-processing functions throw an error such as `variable 'x' not found in either 'data' or 'data2'`, consider using `aux_variables`.
`grid_add`	An optional argument to specify the variable(s) that can be passed to the `marginaleffects::datagrid()` This is useful when fitting location-scale models and measurement error models. If post-processing functions throw an error such as `variable 'x' not found in either 'data' or 'data2'`, consider using `grid_add`. Note that unlike `aux_variables` which are passed to the internal data functions such as `'get.newdata'`, the `grid_add` are passed to the `marginaleffects::datagrid()`.
`levels_id`	An optional argument to specify the `ids` for the hierarchical model (default `NULL`). It is used only when the model is applied to data with three or more levels of hierarchy. For a two-level model, `levels_id` is automatically inferred from the model fit. For models with three or more levels, `levels_id` is inferred from the model fit under the assumption that hierarchy is specified from the lowest to the uppermost level, i.e., `id` followed by `study`, where `id` is nested within `study`. However, it is not guaranteed that `levels_id` is sorted correctly, so it is better to set it manually when fitting a model with three or more levels of hierarchy.
`avg_reffects`	An optional argument (default `NULL`) to calculate (marginal/average) curves and growth parameters, such as APGV and PGV. If specified, it must be a named list indicating the `over` (typically a level 1 predictor, such as age), `feby` (fixed effects, typically a factor variable), and `reby` (typically `NULL`, indicating that parameters are integrated over the random effects). For example, `avg_reffects = list(feby = 'study', reby = NULL, over = 'age')`.
`idata_method`	A character string specifying the interpolation method (default `NULL`). The number of interpolation points is controlled by the `ipts` argument. Available options: • `'m1'`: Adapted from the **iapvbs** package (documented here). This method internally constructs the data frame based on the model configuration. *Note:* This method may fail if the model includes covariates (especially in `univariate_by` models). • `'m2'`: Based on the **JMbayes** package (documented here). This method uses the exact data frame from the model fit (`fit$data`) and is generally more robust. If `idata_method = NULL`, `'m2'` is automatically selected. It is recommended to use `'m2'` if `'m1'` encounters errors with covariate-dependent models.
`ipts`	An integer specifying the number of points for interpolating the predictor variable (e.g., age) to generate smooth curves for predictions and plots. This value is used as the `length.out` argument for `seq()`, controlling the smoothness of distance and velocity curves without altering the predictor range.

`NULL` **(the default)** Engages automatic behavior based on the `dpar` argument: it is internally set to `50` for the mean response (`dpar = 'mu'`), ensuring smooth curves, while for the distributional parameters (e.g., `dpar = 'sigma'`), no interpolation is performed.

An integer (e.g., `100`**)** Explicitly sets the number of interpolation points.

`FALSE` Disables all interpolation, forcing predictions to be made only at the original data points of the predictor variable.

This argument affects the following post-processing functions: `fitted_draws()`, `predict_draws()`, `growthparameters()`, `plot_curves()`, `get_predictions()`, `get_comparisons()`, and `get_growthparameters()`.

`seed` An integer (default `123`) that is passed to the estimation method to ensure reproducibility.

`future` A logical value (default `FALSE`) indicating whether to perform parallel computations. If `TRUE`, posterior summaries are computed in parallel using `future.apply::future_sapply()`.

`future_session` A character string or a named list specifying the parallel plan when `future = TRUE`.

- **Character string**: Defaults to `"multisession"`. Use `"multicore"` for forking (not supported on Windows).
- **Named list**: Useful for advanced plans like `future.mirai::mirai_cluster()`. The list must contain:
  - `future_session`: The planner function (e.g., `mirai_cluster`).
  - Additional named arguments passed to the planner (e.g., `daemons` for `mirai::daemons()`).

  Example: `list(future_session = mirai_cluster, daemons = list(...))`.

`future_splits` A list, numeric vector, or logical value (default `TRUE`). Controls how posterior draws are partitioned into smaller subsets for parallel computation. This helps manage memory and improve performance, particularly on Linux when using `future::plan("multicore")`.

Input options:

- **List**: (e.g., `list(1:6, 7:10)`). Each element is a sequence of numbers passed to `draw_ids` to process in a separate chunk.
- **Numeric vector of length 2**: (e.g., `c(100, 4)`). The first element is the total number of draws, and the second is the number of splits. Indices are generated via `parallel::splitIndices()`.
- **Numeric vector of length 1**: Specifies the total number of draws. Splits are calculated automatically.
- `TRUE`: Automatically creates splits based on `ndraws` and `cores`. If both `ndraws` and `draw_ids` are specified, `draw_ids` takes precedence. This is consistent with post-processing functions in the `bsitar` and `brms` packages.
- `FALSE`: Splitting is disabled.

Note: On Windows with `future::plan("multisession")`, background R processes may not free resources automatically. If needed, use `installr::kill_all_Rscript_s()` to terminate them.

`future_method` A character string (default `'future'`) specifying the parallel computation backend. Options include:

	• `'future'`: Uses `future.apply::future_lapply()` for execution. • `'dofuture'`: Uses `foreach::foreach()` via the `doFuture` package.
`future_re_expose`	A logical value (default `NULL`) indicating whether to re-expose internal `Stan` functions when `future = TRUE`. This is critical when `future::plan()` is set to `"multisession"`, as compiled C++ functions cannot be exported across distinct R sessions. • If `NULL` (default), it is automatically set to `TRUE` when the plan is `"multisession"`. • Explicitly setting this to `TRUE` is recommended for improved performance during parallel execution.
`usedtplyr`	A logical (default `FALSE`) indicating whether to use the **dtplyr** package for summarizing the draws. This package uses **data.table** as a back-end. It is useful when the data has a large number of observations. For typical use cases, it does not make a significant performance difference. The `usedtplyr` argument is evaluated only when `method = 'custom'`.
`usecollapse`	A logical (default `FALSE`) to indicate whether to use the **collapse** package for summarizing the draws.
`parallel`	A logical (default `FALSE`) indicating whether to use parallel computation (via **doParallel** and **foreach**) when **usecollapse = TRUE**. When `parallel = TRUE`, `parallel::makeCluster()` sets the cluster type as `"PSOCK"`, which works on all operating systems, including `Windows`. If you want to use a faster option for Unix-based systems, you can set `parallel = "FORK"`, but note that it is not compatible with `Windows` systems.
`cores`	A positive integer (default `1`) specifying the number of CPU cores to use when `parallel = TRUE`. To automatically detect the number of available cores, set `cores = NULL`. This is useful for optimizing performance when working with large datasets.
`fullframe`	A logical value indicating whether to return a `fullframe` object in which `newdata` is bound to the summary estimates. Note that `fullframe` cannot be used with `summary = FALSE`, and it is only applicable when `idata_method = 'm2'`. A typical use case is when fitting a `univariate_by` model. This option is mainly for internal use.
`average`	A logical value indicating whether to internally call the `marginaleffects::comparisons()` or the `marginaleffects::avg_comparisons()` function. If `FALSE` (default), `marginaleffects::comparisons()` is called, otherwise `marginaleffects::avg_comparisons()` is used when `average = TRUE`.
`plot`	A logical value specifying whether to plot comparisons by calling the `marginaleffects::plot_compari` function (`TRUE`) or not (`FALSE`). If `FALSE` (default), then `marginaleffects::comparisons()` or `marginaleffects::avg_comparisons()` are called to compute predictions (see the `average` argument for details).
`mapping_facet`	A named list that can be used to pass the aesthetic `mapping` and `facet` arguments to the `ggplot2` object (default `NULL`). The main use of this is to get overlay line plot instead of separate line plot for each `id` variable. This can also be used to remove a layer. For example, if user wants to remove the confidence intervals band from the lines i.e., `geom_ribbon`, then include `rm_geom_ribbon = TRUE`.

Note the prefix `'rm_'`. This is a general approach in which any layer name with prefix `'rm_'` included in the `mapping_facet` will be removed. An example is shown below:
`mapping_facet = list(group = 'id', colour = 'id', facet_wrap = c('study'), rm_geom_ribbon = TRUE)`.

`showlegends` A logical value to specify whether to show legends (`TRUE`) or not (`FALSE`). If `NULL` (default), the value of `showlegends` is internally set to `TRUE` if `re_formula = NA`, and `FALSE` if `re_formula = NULL`.

`variables` A named list specifying the level 1 predictor, such as `age` or `time`, used for estimating growth parameters in the current use case. The `variables` list is set via the `esp` argument (default value is `1e-6`). If `variables` is `NULL`, the relevant information is retrieved internally from the `model`. Alternatively, users can define `variables` as a named list, e.g., `variables = list('x' = 1e-6)` where `'x'` is the level 1 predictor. By default, `variables = list('age' = 1e-6)` in the **marginaleffects** package, as velocity is usually computed by differentiating the distance curve using the `dydx` approach. When using this default, the argument `deriv` is automatically set to `0` and `model_deriv` to `FALSE`. If parameters are to be estimated based on the model's first derivative, `deriv` must be set to `1` and `variables` will be defined as `variables = list('age' = 0)`. Note that if the default behavior is used (`deriv = 0` and `variables = list('x' = 1e-6)`), additional arguments cannot be passed to `variables`. In contrast, when using an alternative approach (`deriv = 0` and `variables = list('x' = 0)`), additional options can be passed to the `marginaleffects::comparisons()` and `marginaleffects::avg_comparisons()` functions.

`condition` Conditional slopes

- Character vector (max length 4): Names of the predictors to display.
- Named list (max length 4): List names correspond to predictors. List elements can be:
  - Numeric vector
  - Function which returns a numeric vector or a set of unique categorical values
  - Shortcut strings for common reference values: "minmax", "quartile", "threenum"
- 1: x-axis. 2: color/shape. 3: facet (wrap if no fourth variable, otherwise cols of grid). 4: facet (rows of grid).
- Numeric variables in positions 2 and 3 are summarized by Tukey's five numbers `?stats::fivenum`.

`deriv` A numeric value specifying whether to estimate parameters based on the differentiation of the distance curve or the model's first derivative. Please refer to the `variables` argument for more details.

`model_deriv` A logical value specifying whether to estimate the velocity curve from the derivative function or by differentiating the distance curve. Set `model_deriv = TRUE` for functions that require the velocity curve, such as `growthparameters()` and `plot_curves()`. Set it to `NULL` for functions that use the distance curve (i.e., fitted values), such as `loo_validation()` and `plot_ppc()`.

`method`	A character string indicating whether to compute estimates using the `'marginaleffects'` package (`method = 'pkg'`) or custom functions for efficiency and speed (`method = 'custom'`, default). The `method = 'pkg'` option is only suitable for simple cases and should be used with caution. `method = 'custom'` is the preferred option because it allows for simultaneous estimation of multiple parameters (e.g., `'apgv'` and `'pgv'`). This method works during the post-draw stage, supports multiple parameter comparisons via the `hypothesis` argument, and allows users to add or return draws (see `pdraws` for details). If `method = 'pkg'`, the `by` argument must not contain the predictor (e.g., `age`), and `variables` must either be `NULL` (which defaults to `list(age = 1e-6)`) or a list with factor variables like `variables = list(class = 'pairwise')` or `variables = list(age = 1e-6, class = 'pairwise')`. With `method = 'custom'`, the `by` argument can include predictors, which will be ignored, and `variables` should not contain predictors, but can accept factor variables as a vector (e.g., `variables = c('class')`). Using `method = 'custom'` is strongly recommended for better performance and flexibility.
`marginals`	A `list`, `data.frame`, or `tibble` of velocity returned by the **marginaleffects** functions (default `NULL`). This is only evaluated when `method = 'custom'`. The `marginals` can be the output from **marginaleffects** functions or posterior draws from `marginaleffects::posterior_draws()`. The `marginals` argument is primarily used for internal purposes only.
`preparms`	A `list`, `data.frame`, or `tibble` of pre computed parameters (default `NULL`). This is only evaluated when `method = 'custom'`. The `preparms` argument is primarily used for internal purposes only.
`pdraws`	A character string (default `FALSE`) that indicates whether to return the raw posterior draws. Options include: • `'return'`: returns the raw draws, • `'add'`: adds the raw draws to the final return object, • `'returns'`: returns the summary of the raw draws, • `'adds'`: adds the summary of raw draws to the final return object. The `pdraws` are the velocity estimates for each posterior sample. For more details, see `marginaleffects::posterior_draws()`.
`pdrawso`	A character string (default `FALSE`) to indicate whether to return the original posterior draws for parameters. Options include: • `'return'`: returns the original posterior draws, • `'add'`: adds the original posterior draws to the outcome. When `pdrawso = TRUE`, the default behavior is `pdrawso = 'return'`. Note that the posterior draws are returned before calling `marginaleffects::posterior_draws()`.
`pdrawsp`	A character string (default `FALSE`) to indicate whether to return the posterior draws for parameters. Options include: • `'return'`: returns the posterior draws for parameters, • `'add'`: adds the posterior draws to the outcome. When `pdrawsp = TRUE`, the default behavior is `pdrawsp = 'return'`. The `pdrawsp` represent the parameter estimates for each of the posterior samples, and the summary of these are the estimates returned.

`pdrawsh`	A character string (default `FALSE`) to indicate whether to return the posterior draws for parameter contrasts. Options include: • `'return'`: returns the posterior draws for contrasts. The summary of posterior draws for parameters is the default returned object. The `pdrawsh` represent the contrast estimates for each of the posterior samples, and the summary of these are the contrast returned.
`comparison`	A character string specifying the comparison type for growth parameter estimation. Options are `'difference'` and `'differenceavg'`. This argument sets up the internal function for estimating parameters using `sitar::getPeak()`, `sitar::getTakeoff()`, and `sitar::getTrough()` functions. These options are restructured according to the user-specified `hypothesis` argument.
`type`	string indicates the type (scale) of the predictions used to compute contrasts or slopes. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When `type` is `NULL`, the first entry in the error message is used by default. See the Type section in the documentation below.
`by`	Aggregate unit-level estimates (aka, marginalize, average over). Valid inputs: • `FALSE`: return the original unit-level estimates. • `TRUE`: aggregate estimates for each term. • Character vector of column names in `newdata` or in the data frame produced by calling the function without the `by` argument. • Data frame with a `by` column of group labels, and merging columns shared by `newdata` or the data frame produced by calling the same function without the `by` argument. • See examples below. • For more complex aggregations, you can use the `FUN` argument of the `hypotheses()` function. See that function's documentation and the Hypothesis Test vignettes on the `marginaleffects` website.
`bys`	A character string (default `NULL`) specifying the variables over which the parameters are summarized. The summary statistics are computed for each unique combination of variables specified.
`conf_level`	numeric value between 0 and 1. Confidence level to use to build a confidence interval.
`transform`	string or function. Transformation applied to unit-level estimates and confidence intervals just before the function returns results. Functions must accept a vector and return a vector of the same length. Support string shortcuts: "exp", "ln"
`transform_draws`	A function applied to individual draws from the posterior distribution before computing summaries (default `NULL`). The argument `transform_draws` is derived from the `marginaleffects::predictions()` function and should not be confused with the `transform` argument from the deprecated `brms::posterior_predict()` function. It's important to note that for both `marginaleffects::predictions()` and `marginaleffects::avg_predictions()`, the `transform_draws` argument takes precedence over the `transform` argument. Note that when `transform_draws`

= `NULL`, an attempt is made to automatically set `transform_draws = 'exp'` for `dpar = 'sigma'`. User can set `transform_draws = FALSE` to turn off this automatic assignment of `'exp'` to the `transform_draws`. It is also important to set `transform_draws = FALSE` when computing the first derivative (velocity) for `dpar = 'sigma'`.

`cross`
- `FALSE`: Contrasts represent the change in adjusted predictions when one predictor changes and all other variables are held constant.
- `TRUE`: Contrasts represent the changes in adjusted predictions when all the predictors specified in the `variables` argument are manipulated simultaneously (a "cross-contrast").

`wts` logical, string or numeric: weights to use when computing average predictions, contrasts or slopes. These weights only affect the averaging in `avg_*()` or with the `by` argument, and not unit-level estimates. See `?weighted.mean`
- string: column name of the weights variable in `newdata`. When supplying a column name to `wts`, it is recommended to supply the original data (including the weights variable) explicitly to `newdata`.
- numeric: vector of length equal to the number of rows in the original data or in `newdata` (if supplied).
- FALSE: Equal weights.
- TRUE: Extract weights from the fitted object with `insight::find_weights()` and use them when taking weighted averages of estimates. Warning: `newdata=datagrid()` returns a single average weight, which is equivalent to using `wts=FALSE`

`hypothesis` specify a hypothesis test or custom contrast using a number , formula, string equation, vector, matrix, or function.
- Number: The null hypothesis used in the computation of Z and p (before applying `transform`).
- String: Equation to specify linear or non-linear hypothesis tests. Two-tailed tests must include an equal = sign. One-tailed tests must start with < or >. If the terms in `coef(object)` uniquely identify estimates, they can be used in the formula. Otherwise, use `b1`, `b2`, etc. to identify the position of each parameter. The `b*` wildcard can be used to test hypotheses on all estimates. When the hypothesis string represents a two-sided equation, the `estimate` column holds the value of the left side minus the right side of the equation. If a named vector is used, the names are used as labels in the output. Examples:
  - `hp = drat`
  - `hp + drat = 12`
  - `b1 + b2 + b3 = 0`
  - `b* / b1 = 1`
  - `<= 0`
  - `>= -3.5`
  - `b1 >= 10`
- Formula: `lhs ~ rhs | group`
  - `lhs`
    * `ratio` (null = 1)

    - `difference` (null = 0)
    - Leave empty for default value
  - `rhs`
    - `pairwise` and `revpairwise`: pairwise differences between estimates in each row.
    - `reference`: differences between the estimates in each row and the estimate in the first row.
    - `sequential`: difference between an estimate and the estimate in the next row.
    - `meandev`: difference between an estimate and the mean of all estimates.
    - `meanotherdev`: difference between an estimate and the mean of all other estimates, excluding the current one.
    - `poly`: polynomial contrasts, as computed by the `stats::contr.poly()` function.
    - `helmert`: Helmert contrasts, as computed by the `stats::contr.helmert()` function. Contrast 2nd level to the first, 3rd to the average of the first two, and so on.
    - `trt_vs_ctrl`: difference between the mean of estimates (except the first) and the first estimate.
    - `I(fun(x))`: custom function to manipulate the vector of estimates `x`. The function `fun()` can return multiple (potentially named) estimates.
  - `group` (optional)
    - Column name of `newdata`. Conduct hypothesis tests withing subsets of the data.
  - Examples:
    - `~ poly`
    - `~ sequential | groupid`
    - `~ reference`
    - `ratio ~ pairwise`
    - `difference ~ pairwise | groupid`
    - `~ I(x - mean(x)) | groupid`
    - `~ I(\(x) c(a = x[1], b = mean(x[2:3]))) | groupid`
- Matrix or Vector: Each column is a vector of weights. The the output is the dot product between these vectors of weights and the vector of estimates. The matrix can have column names to label the estimates.
- Function:
  - Accepts an argument `x`: object produced by a `marginaleffects` function or a data frame with column `rowid` and `estimate`
  - Returns a data frame with columns `term` and `estimate` (mandatory) and `rowid` (optional).
  - The function can also accept optional input arguments: `newdata`, `by`, `draws`.

- This function approach will not work for Bayesian models or with bootstrapping. In those cases, it is easy to use `get_draws()` to extract and manipulate the draws directly.

- See the Examples section below and the vignette: https://marginaleffects.com/chapters/hypothesis.html
- Warning: When calling `predictions()` with `type="invlink(link)"` (the default in some models), `hypothesis` is tested and p values are computed on the link scale.

`equivalence` Numeric vector of length 2: bounds used for the two-one-sided test (TOST) of equivalence, and for the non-inferiority and non-superiority tests. For bayesian models, this report the proportion of posterior draws in the interval and the ROPE. See Details section below.

`equivalence_test`

A named arguments list (default `NULL`) passed to `bayestestR::equivalence_test()` for ROPE-based hypothesis testing. Core arguments:

- `range = "default"`: ROPE bounds (numeric vector or `"default"`). Defaults values that are automatically set up for `range` are: `c(-1, 1)` for age parameters (`apgv`, `atgv`, `acgv`); `c(-0.5, 0.5)` for velocity parameters (`pgv`, `tgv`, `cgv`).
- `ci`: Credible interval level. Default `0.95`.
- `rvar_col`: Random variable column name for long-format data. Default `NULL`.
- `verbose`: Print progress messages if `TRUE`. Default `FALSE`.

Additional package-specific controls:

- `format`: If `TRUE`, merge `ROPE_low`/`ROPE_high` into `ROPE_range` and `HDI_low`/`HDI_high` into `HDI_range`.
- `reformat`: If `TRUE`, standardize `marginaleffects::comparisons()` output (`conf.low` → `Q2.5`, `conf.high` → `Q97.5`) and drop reporting columns (`term`, `contrast`, etc.).
- `get_range_null_form`: If `TRUE`, return expected `range` and `null` structure for manual specification via `comparison_range_null` and `hypothesis_range_null`.
- `digits`: Number of digits to use when printing numeric results. Default `2`.
- `as_percent`: Logical indicating whether to return ROPE results as percentages (`TRUE`, default) or fractions (`FALSE`). Only evaluated when `model` is class `"bsitar"` and engine is `"bayestestR"` or `"mbcombo"`.
- `na.rm`: If `TRUE` (default), remove `NA` values.
- `inline`: Internal use only; executes custom equivalence function (not for users).

`p_direction` A named arguments list (default `NULL`) passed to `bayestestR::p_direction()` for Probability of Direction (pd) computation. Commonly used elements:

- `method`: Computation method. Default `"direct"`.
- `null`: Null value (default `0` for all growth parameters).
- `as_p`: Return p-value-like scale. Default `FALSE`.
- `remove_na`: Remove missing values. Default `TRUE`.

Additional package-specific controls:

- `get_range_null_form`: If `TRUE`, return expected `range` and `null` structure for manual specification via `comparison_range_null` and `hypothesis_range_null`.
- `digits`: Number of digits to use when printing numeric results. Default `2`.
- `as_percent`: Logical indicating whether to return PD results as percentages (`TRUE`, default) or fractions (`FALSE`). Only evaluated when `model` is class `"bsitar"` and `engine` is `"bayestestR"` or `"mbcombo"`.
- `na.rm`: If `TRUE` (default), remove `NA` values.
- `inline`: Internal use only; executes custom equivalence function (not for users).

`eps` NULL or numeric value which determines the step size to use when calculating numerical derivatives: (f(x+eps)-f(x))/eps. When `eps` is `NULL`, the step size is 0.0001 multiplied by the difference between the maximum and minimum values of the variable with respect to which we are taking the derivative. Changing `eps` may be necessary to avoid numerical problems in certain models.

`constrats_by` A character vector (default `FALSE`) specifying the variable(s) by which estimates and contrasts (during the post-draw stage) should be computed using the `hypothesis` argument. The variable(s) in `constrats_by` should be a subset of those specified in the `by` argument. If `constrats_by = NULL`, it will copy all variables from by, except for the level-1 predictor (e.g., `age`). To disable this automatic behavior, use `constrats_by = FALSE`. This argument is evaluated only when `method = 'custom'` and `hypothesis` is not `NULL`.

`constrats_at` A named list (default `FALSE`) to specify the values at which estimates and contrasts should be computed during the post-draw stage using the `hypothesis` argument. The values can be specified as `'max'`, `'min'`, `'unique'`, or `'range'` (e.g., `constrats_at = list(age = 'min')`) or as numeric vectors (e.g., `constrats_at = list(age = c(6, 7))`). If `constrats_at = NULL`, level-1 predictors (e.g., `age`) are automatically set to their unique values (i.e., `constrats_at = list(age = 'unique')`). To turn off this behavior, use `constrats_at = FALSE`. Note that `constrats_at` only affects data subsets prepared via `marginaleffects::datagrid()` or the `newdata` argument. The argument is evaluated only when `method = 'custom'` and `hypothesis` is not `NULL`.

`constrats_subset`
A named list (default `FALSE`) to subset draws for which contrasts should be computed via the `hypothesis` argument. This is similar to `constrats_at`, except that `constrats_subset` filters draws based on the character vector of variable names (e.g., `constrats_subset = list(id = c('id1', 'id2'))`) rather than numeric values. The argument is evaluated only when `method = 'custom'` and `hypothesis` is not `NULL`.

`reformat` A logical (default `TRUE`) indicating whether to reformat the output returned by `marginaleffects` as a data frame. Column names are redefined as `conf.low` to `Q2.5` and `conf.high` to `Q97.5` (assuming `conf_int = 0.95`). Additionally, some columns (`term`, `contrast`, etc.) are dropped from the data frame.

`estimate_center`
A character string (default `NULL`) specifying how to center estimates: either `'mean'` or `'median'`. This option sets the global options as follows: `options("marginaleffects_poste`

= "mean") or options("marginaleffects_posterior_center" = "median"). These global options are restored upon function exit using base::on.exit().

estimate_interval
: A character string (default NULL) to specify the type of credible intervals: 'eti' for equal-tailed intervals or 'hdi' for highest density intervals. This option sets the global options as follows: options("marginaleffects_posterior_interval" = "eti") or options("marginaleffects_posterior_interval" = "hdi"), and is restored on exit using base::on.exit().

dummy_to_factor
: A named list (default NULL) to convert dummy variables into a factor variable. The list must include the following elements:
  - factor.dummy: A character vector of dummy variables to be converted to factors.
  - factor.name: The name for the newly created factor variable (default is 'factor.var' if NULL).
  - factor.level: A vector specifying the factor levels. If NULL, levels are taken from factor.dummy. If factor.level is provided, its length must match factor.dummy.

verbose
: A logical argument (default FALSE) to specify whether to print information collected during the setup of the object(s).

expose_function
: A logical value or NULL (default FALSE). Controls whether Stan functions are exposed for post-processing.
  - TRUE: Explicitly exposes Stan functions saved from the model fit. This is required when calculating fit criteria or bayes_R2 during model optimization.
  - FALSE (default): Stan functions are not exposed.
  - NULL: The setting is inherited from the original bsitar() model fit configuration.

  Note: In the optimize_model() function, the default is NULL (inheriting behavior), whereas other post-processing functions default to FALSE.

usesavedfuns
: A logical value (default NULL) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the expose_functions argument from the bsitar() call. Manual specification of usesavedfuns is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.

clearenvfuns
: A logical value indicating whether to clear the exposed Stan functions from the environment (TRUE) or not (FALSE). If NULL, clearenvfuns is set based on the value of usesavedfuns: TRUE if usesavedfuns = TRUE, or FALSE if usesavedfuns = FALSE.

funlist
: A list (default NULL) specifying function names. This is rarely needed, as required functions are typically retrieved automatically. A use case for funlist is when sigma_formula, sigma_formula_gr, or sigma_formula_gr_str use an external function (e.g., poly(age)). The funlist should include function names defined in the globalenv(). For functions needing both distance and

	velocity curves (e.g., `plot_curves(..., opt = 'dv')`), `funlist` must include two functions: one for the distance curve and one for the velocity curve.
`xvar`	A character string (default `NULL`) specifying the `'x'` variable. Rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.
`difx`	A character string (default `NULL`) specifying the `'x'` variable that should be used for manual differentiation of the distance curve. Internally, the `xvar` is set as `difx` if specified. The argument `difx` is evaluated only when `dpar = 'sigma'`, ignored otherwise. Note that argument `xvar` itself is rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.
`idvar`	A character string (default `NULL`) specifying the `'id'` variable. Rarely used because `idvar` is inferred internally.
`itransform`	A character string (default `NULL`) indicating the variables names that are reverse transformed. Options are `c("x", "y", "sigma")`. The `itransform` is primarily used to get the `xvar` variable at original scale i.e., `itransform = 'x'`. To turn of all transformations, use `itransform = ""`. when `itransform = NULL`, the appropriate transformation for `xvar` is selected automatically. Note that when no match for `xvar` is found in the `data,frame`, the `itransform` will be ignored within the calling function, `'prepare_transformations()'`.
`newdata_fixed`	An indicator to specify whether to check data format and structure for the user provided `newdata`, and apply needed `prepare_data2` and `prepare_transformations` (`newdata_fixed = NULL`, default), return user provided `newdata` (`newdata = TRUE`) as it is without checking for the data format or applying `prepare_data2` and `prepare_transformations` (`newdata_fixed = 0`), check for the data format and if needed, prepare data format using `prepare_data2` (`newdata_fixed = 1`), or apply `prepare_transformations` only assuming that data format is correct (`newdata_fixed = 2`). It is strongly recommended that user either leave the `newdata = NULL` and `newdata_fixed = NULL` in which case data used in the model fitting is automatically retrieved and checked for the required data format and transformations, and if needed, `prepare_data2` and `prepare_transformations` are applied internally. The other flags provided for `newdata_fixed = 0, 1, 2` are mainly for the internal use during post-processing.
`envir`	The environment used for function evaluation. The default is `NULL`, which sets the environment to `parent.frame()`. Since most post-processing functions rely on **brms**, it is recommended to set `envir = globalenv()` or `envir = .GlobalEnv`, especially for derivatives like velocity curves.
`...`	Additional arguments passed to the `brms::fitted.brmsfit()` function. Please see `brms::fitted.brmsfit()` for details on various options available.

Details

The `get_growthparameters` function estimates and returns the following growth parameters:

- pgv - peak growth velocity
- apgv - age at peak growth velocity

- `tgv` - takeoff growth velocity
- `atgv` - age at takeoff growth velocity
- `cgv` - cessation growth velocity
- `acgv` - age at cessation growth velocity

The takeoff growth velocity is the lowest velocity just before the peak begins, indicating the start of the pubertal growth spurt. The cessation growth velocity marks the end of the active pubertal growth spurt and is calculated as a certain percentage of the peak velocity (`pgv`). Typically, 10 percent of `pgv` is considered a good indicator for the cessation of the active pubertal growth spurt (Hardin et al. 2022). This percentage is controlled via the `acg_velocity` argument, which accepts a positive real value bounded between 0 and 1 (with the default value being `0.1`, indicating 10 percent).

Value

A `data.frame` containing posterior estimates and credible intervals (CIs). The output includes the parameter estimates and their corresponding bounds (e.g., `Q2.5` and `Q97.5` for a 95% interval). The specific column names and structure may vary depending on the computation method.

Author(s)

Satpal Sandhu <`satpal.sandhu@bristol.ac.uk`>

References

Hardin AM, Knigge RP, Oh HS, Valiathan M, Duren DL, McNulty KP, Middleton KM, Sherwood RJ (2022). "Estimating Craniofacial Growth Cessation: Comparison of Asymptote- and Rate-Based Methods." *The Cleft Palate Craniofacial Journal*, **59**(2), 230-238. doi:10.1177/10556656211002675, PMID: 33998905.

See Also

`marginaleffects::comparisons()` `marginaleffects::avg_comparisons()` `marginaleffects::plot_comparisons(`

Examples

```
# Fit Bayesian SITAR model

# To avoid mode estimation which takes time, the Bayesian SITAR model fit to
# the 'berkeley_exdata' has been saved as an example fit ('berkeley_exfit').
# See 'bsitar' function for details on 'berkeley_exdata' and 'berkeley_exfit'.

# Note that since no covariate is part of the model fit, the below example
# doesn't make sense and is included here only for the purpose of completeness.

# Check and confirm whether model fit object 'berkeley_exfit' exists
 berkeley_exfit <- getNsObject(berkeley_exfit)

model <- berkeley_exfit

get_growthparameters(model, parameter = 'apgv', ndraws = 10)
```

---

get_predictions.bgmfit *Estimate and plot growth curves for the Bayesian SITAR model*

Description

The **get_predictions()** function estimates and plots growth curves (distance and velocity) by using the **marginaleffects** package as a back-end. This function computes growth curves (`marginaleffects::predictions()`), average growth curves (`marginaleffects::avg_predictions()`), and plots growth them via the `marginaleffects::plot_predictions()`. See here for details. Note that the **marginaleffects** package is highly flexible, and therefore, it is expected that the user has a strong understanding of its workings. Furthermore, since **marginaleffects** is rapidly evolving, the results obtained from the current implementation should be considered experimental. Also note that unlike `get_predictions()`, the `fitted_draws()` and `predict_draws()` uses the **brms** package as a back-end.

Usage

```
## S3 method for class 'bgmfit'
get_predictions(
  model,
  resp = NULL,
  dpar = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  newdata = NULL,
  datagrid = NULL,
  re_formula = NA,
  newdata2 = NULL,
  allow_new_levels = FALSE,
  sample_new_levels = "gaussian",
  parameter = NULL,
  xrange = 1,
  acg_velocity = 0.1,
  digits = 2,
  numeric_cov_at = NULL,
  aux_variables = NULL,
  grid_add = NULL,
  levels_id = NULL,
  avg_reffects = NULL,
  idata_method = NULL,
  ipts = NULL,
  seed = 123,
  future = FALSE,
  future_session = "multisession",
  future_splits = TRUE,
  future_method = "future",
  future_re_expose = NULL,
  usedtplyr = FALSE,
  usecollapse = TRUE,
  cores = NULL,
  fullframe = FALSE,
  average = FALSE,
  plot = FALSE,
  mapping_facet = NULL,
  showlegends = NULL,
  variables = NULL,
  condition = NULL,
  deriv = 0,
  model_deriv = TRUE,
  method = "custom",
  marginals = NULL,
  pdrawso = FALSE,
  pdrawsp = FALSE,
  pdrawsh = FALSE,
  type = NULL,
  by = NULL,
  conf_level = 0.95,
  transform = NULL,
  transform_draws = NULL,
  byfun = NULL,
  wts = NULL,
  hypothesis = NULL,
  equivalence = NULL,
  eps = NULL,
  constrats_by = NULL,
  constrats_at = NULL,
  reformat = NULL,
  estimate_center = NULL,
  estimate_interval = NULL,
  dummy_to_factor = NULL,
  verbose = FALSE,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  funlist = NULL,
  xvar = NULL,
  difx = NULL,
  idvar = NULL,
  itransform = NULL,
  newdata_fixed = NULL,
  envir = NULL,
  ...
```

```
)

get_predictions(model, ...)

marginal_draws(model, ...)
```

Arguments

`model`	An object of class `bgmfit`.
`resp`	A character string (default `NULL`) to specify the response variable when processing posterior draws for `univariate_by` and `multivariate` models. See `bsitar()` for details on `univariate_by` and `multivariate` models.
`dpar`	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
`ndraws`	A positive integer indicating the number of posterior draws to use in estimation. If `NULL` (default), all draws are used.
`draw_ids`	An integer specifying the specific posterior draw(s) to use in estimation (default `NULL`).
`newdata`	An optional data frame for estimation. If `NULL` (default), `newdata` is retrieved from the `model`.
`datagrid`	A grid of user-specified values to be used in the `newdata` argument of various functions in the **marginaleffects** package. This allows you to define the regions of the predictor space where you want to evaluate the quantities of interest. See `marginaleffects::datagrid()` for more details. By default, the `datagrid` is set to `NULL`, meaning no custom grid is constructed. To set a custom grid, the argument should either be a data frame created using `marginaleffects::datagrid()`, or a named list, which is internally used for constructing the grid. For convenience, you can also pass an empty list `datagrid = list()`, in which case essential arguments like `model` and `newdata` are inferred from the respective arguments specified elsewhere. Additionally, the first-level predictor (such as age) and any covariates included in the model (e.g., gender) are automatically inferred from the `model` object.
`re_formula`	Option to indicate whether or not to include individual/group-level effects in the estimation. When `NA` (default), individual-level effects are excluded, and population average growth parameters are computed. When `NULL`, individual-level effects are included in the computation, and the resulting growth parameters are individual-specific. In both cases (`NA` or `NULL`), continuous and factor covariates are appropriately included in the estimation. Continuous covariates are set to their means by default (see `numeric_cov_at` for details), while factor covariates remain unaltered, allowing for the estimation of covariate-specific population average and individual-specific growth parameters.
`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`.
`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.

`parameter`
: A single character string or character vector specifying the growth parameter(s) to estimate. Must be one of the following options, ordered by growth curve progression:

- `'apgv'`: Age at peak growth velocity - age corresponding to the peak growth rate (i.e. maximum growth velocity) during adolescent growth spurt (default when `NULL`)
- `'pgv'`: Peak growth velocity - maximum growth rate during adolescent growth spurt
- `'atgv'`: Age at takeoff growth velocity - age where initial rapid growth begins accelerating toward peak
- `'tgv'`: Takeoff growth velocity - initial high growth rate at curve start
- `'acgv'`: Age at cessation growth velocity - age where growth rate approaches near zero and growth effectively stops
- `'cgv'`: Cessation growth velocity - final low growth rate approaching maturity

```
\item \code{'spgv'}: Size (length/height) at peak growth velocity age,
\code{'apgv'}
\item \code{'stgv'}: Size at takeoff growth velocity age, \code{'atgv'}
\item \code{'scgv'}: Size at cessation growth velocity age, \code{'acgv'}

\item \code{'satXX'}: Size at specific age XX years of predictor
\code{xvar} (e.g., \code{'sat15'} = size at 15 years, \code{'sat12.5'} =
size at 12.5 years)

\item \code{'all'}: All six primary velocity/age parameters
(\code{'apgv','pgv','atgv','tgv','acgv','cgv'}). Cannot be used when
\code{by = TRUE}.
```

Multiple parameters can be requested simultaneously as a vector, e.g., `c('apgv', 'pgv', 'spgv')`. Custom `'satXX'` format requires no spaces between `'sat'` prefix and numeric age.

`xrange`
: An integer to set the predictor range (e.g., age) when executing the interpolation via `ipts`. By default, `NULL` sets the individual-specific predictor range. Setting

	`xrange = 1` applies the same range for individuals within the same higher grouping variable (e.g., study). Setting `xrange = 2` applies an identical range across the entire sample. Alternatively, a numeric vector (e.g., `xrange = c(6, 20)`) can be provided to set the range within the specified values.
`acg_velocity`	A real number specifying the percentage of peak growth velocity to be used as the cessation velocity when estimating the `cgv` and `acgv` growth parameters. The `acg_velocity` should be greater than `0` and less than `1`. The default value of `acg_velocity = 0.10` indicates that 10 percent of the peak growth velocity will be used to calculate the cessation growth velocity and the corresponding age at cessation velocity. For example, if the peak growth velocity estimate is `10 mm/year`, then the cessation growth velocity will be `1 mm/year`.
`digits`	An integer (default 2) to set the decimal places for rounding the results using the `base::round()` function.
`numeric_cov_at`	An optional (named list) argument to specify the value of continuous covariate(s). The default `NULL` option sets the continuous covariate(s) to their mean. Alternatively, a named list can be supplied to manually set these values. For example, `numeric_cov_at = list(xx = 2)` will set the continuous covariate variable 'xx' to 2. The argument `numeric_cov_at` is ignored when no continuous covariates are included in the model.
`aux_variables`	An optional argument to specify the variable(s) that can be passed to the `ipts` argument (see below). This is useful when fitting location-scale models and measurement error models. If post-processing functions throw an error such as `variable 'x' not found in either 'data' or 'data2'`, consider using `aux_variables`.
`grid_add`	An optional argument to specify the variable(s) that can be passed to the `marginaleffects::datagrid()` This is useful when fitting location-scale models and measurement error models. If post-processing functions throw an error such as `variable 'x' not found in either 'data' or 'data2'`, consider using `grid_add`. Note that unlike `aux_variables` which are passed to the internal data functions such as `'get.newdata'`, the `grid_add` are passed to the `marginaleffects::datagrid()`.
`levels_id`	An optional argument to specify the `ids` for the hierarchical model (default `NULL`). It is used only when the model is applied to data with three or more levels of hierarchy. For a two-level model, `levels_id` is automatically inferred from the model fit. For models with three or more levels, `levels_id` is inferred from the model fit under the assumption that hierarchy is specified from the lowest to the uppermost level, i.e., `id` followed by `study`, where `id` is nested within `study`. However, it is not guaranteed that `levels_id` is sorted correctly, so it is better to set it manually when fitting a model with three or more levels of hierarchy.
`avg_reffects`	An optional argument (default `NULL`) to calculate (marginal/average) curves and growth parameters, such as APGV and PGV. If specified, it must be a named list indicating the `over` (typically a level 1 predictor, such as age), `feby` (fixed effects, typically a factor variable), and `reby` (typically `NULL`, indicating that parameters are integrated over the random effects). For example, `avg_reffects = list(feby = 'study', reby = NULL, over = 'age')`.
`idata_method`	A character string specifying the interpolation method (default `NULL`). The number of interpolation points is controlled by the `ipts` argument. Available options:

- `'m1'`: Adapted from the **iapvbs** package (documented here). This method internally constructs the data frame based on the model configuration. *Note:* This method may fail if the model includes covariates (especially in `univariate_by` models).
- `'m2'`: Based on the **JMbayes** package (documented here). This method uses the exact data frame from the model fit (`fit$data`) and is generally more robust.

If `idata_method = NULL`, `'m2'` is automatically selected. It is recommended to use `'m2'` if `'m1'` encounters errors with covariate-dependent models.

`ipts` An integer specifying the number of points for interpolating the predictor variable (e.g., age) to generate smooth curves for predictions and plots. This value is used as the `length.out` argument for `seq()`, controlling the smoothness of distance and velocity curves without altering the predictor range.

`NULL` **(the default)** Engages automatic behavior based on the `dpar` argument: it is internally set to `50` for the mean response (`dpar = 'mu'`), ensuring smooth curves, while for the distributional parameters (e.g., `dpar = 'sigma'`), no interpolation is performed.

An integer (e.g., `100`) Explicitly sets the number of interpolation points.

`FALSE` Disables all interpolation, forcing predictions to be made only at the original data points of the predictor variable.

This argument affects the following post-processing functions: `fitted_draws()`, `predict_draws()`, `growthparameters()`, `plot_curves()`, `get_predictions()`, `get_comparisons()`, and `get_growthparameters()`.

`seed` An integer (default `123`) that is passed to the estimation method to ensure reproducibility.

`future` A logical value (default `FALSE`) indicating whether to perform parallel computations. If `TRUE`, posterior summaries are computed in parallel using `future.apply::future_sapply()`.

`future_session` A character string or a named list specifying the parallel plan when `future = TRUE`.

- **Character string**: Defaults to `"multisession"`. Use `"multicore"` for forking (not supported on Windows).
- **Named list**: Useful for advanced plans like `future.mirai::mirai_cluster()`. The list must contain:
  - `future_session`: The planner function (e.g., `mirai_cluster`).
  - Additional named arguments passed to the planner (e.g., `daemons` for `mirai::daemons()`).

  Example: `list(future_session = mirai_cluster, daemons = list(...))`.

`future_splits` A list, numeric vector, or logical value (default `TRUE`). Controls how posterior draws are partitioned into smaller subsets for parallel computation. This helps manage memory and improve performance, particularly on Linux when using `future::plan("multicore")`.

Input options:

- **List**: (e.g., `list(1:6, 7:10)`). Each element is a sequence of numbers passed to `draw_ids` to process in a separate chunk.

- **Numeric vector of length 2**: (e.g., `c(100, 4)`). The first element is the total number of draws, and the second is the number of splits. Indices are generated via `parallel::splitIndices()`.
- **Numeric vector of length 1**: Specifies the total number of draws. Splits are calculated automatically.
- `TRUE`: Automatically creates splits based on `ndraws` and `cores`. If both `ndraws` and `draw_ids` are specified, `draw_ids` takes precedence. This is consistent with post-processing functions in the `bsitar` and `brms` packages.
- `FALSE`: Splitting is disabled.

Note: On Windows with `future::plan("multisession")`, background R processes may not free resources automatically. If needed, use `installr::kill_all_Rscript_s()` to terminate them.

`future_method` A character string (default `'future'`) specifying the parallel computation backend. Options include:

- `'future'`: Uses `future.apply::future_lapply()` for execution.
- `'dofuture'`: Uses `foreach::foreach()` via the `doFuture` package.

`future_re_expose`

A logical value (default `NULL`) indicating whether to re-expose internal `Stan` functions when `future = TRUE`. This is critical when `future::plan()` is set to `"multisession"`, as compiled C++ functions cannot be exported across distinct R sessions.

- If `NULL` (default), it is automatically set to `TRUE` when the plan is `"multisession"`.
- Explicitly setting this to `TRUE` is recommended for improved performance during parallel execution.

`usedtplyr` A logical (default `FALSE`) indicating whether to use the **dtplyr** package for summarizing the draws. This package uses **data.table** as a back-end. It is useful when the data has a large number of observations. For typical use cases, it does not make a significant performance difference. The `usedtplyr` argument is evaluated only when `method = 'custom'`.

`usecollapse` A logical (default `FALSE`) to indicate whether to use the **collapse** package for summarizing the draws.

`cores` An integer specifying the number of cores for parallel execution.

- If `NULL` (default), the number of cores is set to `future::availableCores() - 1`.
- On non-Windows systems, this can be controlled globally via the `mc.cores` option.

`fullframe` A logical value indicating whether to return a `fullframe` object in which `newdata` is bound to the summary estimates. Note that `fullframe` cannot be used with `summary = FALSE`, and it is only applicable when `idata_method = 'm2'`. A typical use case is when fitting a `univariate_by` model. This option is mainly for internal use.

`average` A logical indicating whether to internally call the `marginaleffects::predictions()` or `marginaleffects::avg_predictions()` function. If `FALSE` (default), `marginaleffects::predicti` is called; otherwise, `marginaleffects::avg_predictions()` is used when `average = TRUE`.

`plot`	A logical specifying whether to plot predictions by calling `marginaleffects::plot_predictions()` (`TRUE`) or not (`FALSE`). If `FALSE` (default), `marginaleffects::predictions()` or `marginaleffects::avg_predictions()` are called to compute predictions (see `average` for details). Note that `marginaleffects::plot_predictions()` allows either `condition` or `by` arguments, but not both. Therefore, when the `condition` argument is not `NULL`, the `by` argument is set to `NULL`. This step is required because **get_predictions()** automatically assigns the by argument when the model includes a covariate.
`mapping_facet`	A named list that can be used to pass the aesthetic `mapping` and `facet` arguments to the `ggplot2` object (default `NULL`). The main use of this is to get overlay line plot instead of separate line plot for each `id` variable. This can also be used to remove a layer. For example, if user wants to remove the confidence intervals band from the lines i.e., `geom_ribbon`, then include `rm_geom_ribbon = TRUE`. Note the prefix `'rm_'`. This is a general approach in which any layer name with prefix `'rm_'` included in the `mapping_facet` will be removed. An example is shown below: `mapping_facet = list(group = 'id', colour = 'id', facet_wrap = c('study'), rm_geom_ribbon = TRUE).`
`showlegends`	A logical value to specify whether to show legends (`TRUE`) or not (`FALSE`). If `NULL` (default), the value of `showlegends` is internally set to `TRUE` if `re_formula = NA`, and `FALSE` if `re_formula = NULL`.
`variables`	A named list specifying the level 1 predictor, such as `age` or `time`, used for estimating growth parameters in the current use case. The `variables` list is set via the `esp` argument (default value is `1e-6`). If `variables` is `NULL`, the relevant information is retrieved internally from the `model`. Alternatively, users can define `variables` as a named list, e.g., `variables = list('x' = 1e-6)` where `'x'` is the level 1 predictor. By default, `variables = list('age' = 1e-6)` in the **marginaleffects** package, as velocity is usually computed by differentiating the distance curve using the `dydx` approach. When using this default, the argument `deriv` is automatically set to `0` and `model_deriv` to `FALSE`. If parameters are to be estimated based on the model's first derivative, `deriv` must be set to `1` and `variables` will be defined as `variables = list('age' = 0)`. Note that if the default behavior is used (`deriv = 0` and `variables = list('x' = 1e-6)`), additional arguments cannot be passed to `variables`. In contrast, when using an alternative approach (`deriv = 0` and `variables = list('x' = 0)`), additional options can be passed to the `marginaleffects::comparisons()` and `marginaleffects::avg_comparisons()` functions.
`condition`	Conditional slopes • Character vector (max length 4): Names of the predictors to display. • Named list (max length 4): List names correspond to predictors. List elements can be: – Numeric vector – Function which returns a numeric vector or a set of unique categorical values – Shortcut strings for common reference values: "minmax", "quartile", "threenum"

	• 1: x-axis. 2: color/shape. 3: facet (wrap if no fourth variable, otherwise cols of grid). 4: facet (rows of grid). • Numeric variables in positions 2 and 3 are summarized by Tukey's five numbers `?stats::fivenum`.
`deriv`	An integer to indicate whether to estimate the distance curve or its derivative (i.e., velocity curve). The `deriv = 0` (default) is for the distance curve, whereas `deriv = 1` is for the velocity curve.
`model_deriv`	A logical value specifying whether to estimate the velocity curve from the derivative function or by differentiating the distance curve. Set `model_deriv = TRUE` for functions that require the velocity curve, such as `growthparameters()` and `plot_curves()`. Set it to `NULL` for functions that use the distance curve (i.e., fitted values), such as `loo_validation()` and `plot_ppc()`.
`method`	A character string specifying the computation method: whether to use the **marginaleffects** machinery at the post-draw stage, i.e., `marginaleffects::comparisons()` (`method = 'pkg'`) or to use custom functions for efficiency and speed (`method = 'custom'`, default). Note that `method = 'custom'` is particularly useful when testing hypotheses. Also, when `method = 'custom'`, `marginaleffects::predictions()` is used internally instead of `marginaleffects::comparisons()`.
`marginals`	A `list`, `data.frame`, or `tibble` of velocity returned by the **marginaleffects** functions (default `NULL`). This is only evaluated when `method = 'custom'`. The `marginals` can be the output from **marginaleffects** functions or posterior draws from `marginaleffects::posterior_draws()`. The `marginals` argument is primarily used for internal purposes only.
`pdrawso`	A character string (default `FALSE`) to indicate whether to return the original posterior draws for parameters. Options include: • `'return'`: returns the original posterior draws, • `'add'`: adds the original posterior draws to the outcome. When `pdrawso = TRUE`, the default behavior is `pdrawso = 'return'`. Note that the posterior draws are returned before calling `marginaleffects::posterior_draws()`.
`pdrawsp`	A character string (default `FALSE`) to indicate whether to return the posterior draws for parameters. Options include: • `'return'`: returns the posterior draws for parameters, • `'add'`: adds the posterior draws to the outcome. When `pdrawsp = TRUE`, the default behavior is `pdrawsp = 'return'`. The `pdrawsp` represent the parameter estimates for each of the posterior samples, and the summary of these are the estimates returned.
`pdrawsh`	A character string (default `FALSE`) to indicate whether to return the posterior draws for parameter contrasts. Options include: • `'return'`: returns the posterior draws for contrasts. The summary of posterior draws for parameters is the default returned object. The `pdrawsh` represent the contrast estimates for each of the posterior samples, and the summary of these are the contrast returned.
`type`	string indicates the type (scale) of the predictions used to compute contrasts or slopes. This can differ based on the model type, but will typically be a string

such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When `type` is `NULL`, the first entry in the error message is used by default. See the Type section in the documentation below.

`by` Aggregate unit-level estimates (aka, marginalize, average over). Valid inputs:

- `FALSE`: return the original unit-level estimates.
- `TRUE`: aggregate estimates for each term.
- Character vector of column names in `newdata` or in the data frame produced by calling the function without the `by` argument.
- Data frame with a `by` column of group labels, and merging columns shared by `newdata` or the data frame produced by calling the same function without the `by` argument.
- See examples below.
- For more complex aggregations, you can use the `FUN` argument of the `hypotheses()` function. See that function's documentation and the Hypothesis Test vignettes on the `marginaleffects` website.

`conf_level` numeric value between 0 and 1. Confidence level to use to build a confidence interval.

`transform` string or function. Transformation applied to unit-level estimates and confidence intervals just before the function returns results. Functions must accept a vector and return a vector of the same length. Support string shortcuts: "exp", "ln"

`transform_draws`

A function applied to individual draws from the posterior distribution before computing summaries (default `NULL`). The argument `transform_draws` is derived from the `marginaleffects::predictions()` function and should not be confused with the `transform` argument from the deprecated `brms::posterior_predict()` function. It's important to note that for both `marginaleffects::predictions()` and `marginaleffects::avg_predictions()`, the `transform_draws` argument takes precedence over the `transform` argument. Note that when `transform_draws = NULL`, an attempt is made to automatically set `transform_draws = 'exp'` for `dpar = 'sigma'`. User can set `transform_draws = FALSE` to turn off this automatic assignment of `'exp'` to the `transform_draws`. It is also important to set `transform_draws = FALSE` when computing the first derivative (velocity) for `dpar = 'sigma'`.

`byfun` A function such as `mean()` or `sum()` used to aggregate estimates within the subgroups defined by the `by` argument. `NULL` uses the `mean()` function. Must accept a numeric vector and return a single numeric value. This is sometimes used to take the sum or mean of predicted probabilities across outcome or predictor levels. See examples section.

`wts` logical, string or numeric: weights to use when computing average predictions, contrasts or slopes. These weights only affect the averaging in `avg_*()` or with the `by` argument, and not unit-level estimates. See `?weighted.mean`

- string: column name of the weights variable in `newdata`. When supplying a column name to `wts`, it is recommended to supply the original data (including the weights variable) explicitly to `newdata`.

- numeric: vector of length equal to the number of rows in the original data or in `newdata` (if supplied).
- FALSE: Equal weights.
- TRUE: Extract weights from the fitted object with `insight::find_weights()` and use them when taking weighted averages of estimates. Warning: `newdata=datagrid()` returns a single average weight, which is equivalent to using `wts=FALSE`

`hypothesis` specify a hypothesis test or custom contrast using a number , formula, string equation, vector, matrix, or function.

- Number: The null hypothesis used in the computation of Z and p (before applying `transform`).
- String: Equation to specify linear or non-linear hypothesis tests. Two-tailed tests must include an equal `=` sign. One-tailed tests must start with `<` or `>`. If the terms in `coef(object)` uniquely identify estimates, they can be used in the formula. Otherwise, use `b1`, `b2`, etc. to identify the position of each parameter. The `b*` wildcard can be used to test hypotheses on all estimates. When the hypothesis string represents a two-sided equation, the `estimate` column holds the value of the left side minus the right side of the equation. If a named vector is used, the names are used as labels in the output. Examples:
  - `hp = drat`
  - `hp + drat = 12`
  - `b1 + b2 + b3 = 0`
  - `b* / b1 = 1`
  - `<= 0`
  - `>= -3.5`
  - `b1 >= 10`
- Formula: `lhs ~ rhs | group`
  - `lhs`
    * `ratio` (null = 1)
    * `difference` (null = 0)
    * Leave empty for default value
  - `rhs`
    * `pairwise` and `revpairwise`: pairwise differences between estimates in each row.
    * `reference`: differences between the estimates in each row and the estimate in the first row.
    * `sequential`: difference between an estimate and the estimate in the next row.
    * `meandev`: difference between an estimate and the mean of all estimates.
    * `meanotherdev`: difference between an estimate and the mean of all other estimates, excluding the current one.
    * `poly`: polynomial contrasts, as computed by the `stats::contr.poly()` function.

- ∗ `helmert`: Helmert contrasts, as computed by the `stats::contr.helmert()` function. Contrast 2nd level to the first, 3rd to the average of the first two, and so on.
- ∗ `trt_vs_ctrl`: difference between the mean of estimates (except the first) and the first estimate.
- ∗ `I(fun(x))`: custom function to manipulate the vector of estimates `x`. The function `fun()` can return multiple (potentially named) estimates.

– `group` (optional)
- ∗ Column name of `newdata`. Conduct hypothesis tests withing subsets of the data.

– Examples:
- ∗ `~ poly`
- ∗ `~ sequential | groupid`
- ∗ `~ reference`
- ∗ `ratio ~ pairwise`
- ∗ `difference ~ pairwise | groupid`
- ∗ `~ I(x - mean(x)) | groupid`
- ∗ `~ I(\(x) c(a = x[1], b = mean(x[2:3]))) | groupid`

- Matrix or Vector: Each column is a vector of weights. The the output is the dot product between these vectors of weights and the vector of estimates. The matrix can have column names to label the estimates.
- Function:
  – Accepts an argument `x`: object produced by a `marginaleffects` function or a data frame with column `rowid` and `estimate`
  – Returns a data frame with columns `term` and `estimate` (mandatory) and `rowid` (optional).
  – The function can also accept optional input arguments: `newdata`, `by`, `draws`.
  – This function approach will not work for Bayesian models or with bootstrapping. In those cases, it is easy to use `get_draws()` to extract and manipulate the draws directly.
- See the Examples section below and the vignette: https://marginaleffects.com/chapters/hypothesis.html
- Warning: When calling `predictions()` with `type="invlink(link)"` (the default in some models), `hypothesis` is tested and p values are computed on the link scale.

`equivalence` Numeric vector of length 2: bounds used for the two-one-sided test (TOST) of equivalence, and for the non-inferiority and non-superiority tests. For bayesian models, this report the proportion of posterior draws in the interval and the ROPE. See Details section below.

`eps` NULL or numeric value which determines the step size to use when calculating numerical derivatives: (f(x+eps)-f(x))/eps. When `eps` is `NULL`, the step size is 0.0001 multiplied by the difference between the maximum and minimum values of the variable with respect to which we are taking the derivative. Changing `eps` may be necessary to avoid numerical problems in certain models.

`constrats_by`
: A character vector (default `NULL`) specifying the variable(s) by which hypotheses (at the post-draw stage) should be tested. Note that the variable(s) in `constrats_by` should be a subset of the variables included in the `'by'` argument.

`constrats_at`
: A character vector (default `NULL`) specifying the variable(s) at which hypotheses (at the post-draw stage) should be tested. `constrats_at` is particularly useful when the number of rows in the estimates is large because **marginaleffects** does not allow hypotheses testing when the number of rows exceeds 25.

`reformat`
: A logical (default `TRUE`) indicating whether to reformat the output returned by `marginaleffects` as a data frame. Column names are redefined as `conf.low` to `Q2.5` and `conf.high` to `Q97.5` (assuming `conf_int = 0.95`). Additionally, some columns (`term`, `contrast`, etc.) are dropped from the data frame.

`estimate_center`
: A character string (default `NULL`) specifying how to center estimates: either `'mean'` or `'median'`. This option sets the global options as follows: `options("marginaleffects_poste` `= "mean")` or `options("marginaleffects_posterior_center" = "median")`. These global options are restored upon function exit using `base::on.exit()`.

`estimate_interval`
: A character string (default `NULL`) to specify the type of credible intervals: `'eti'` for equal-tailed intervals or `'hdi'` for highest density intervals. This option sets the global options as follows: `options("marginaleffects_posterior_interval" = "eti")` or `options("marginaleffects_posterior_interval" = "hdi")`, and is restored on exit using `base::on.exit()`.

`dummy_to_factor`
: A named list (default `NULL`) to convert dummy variables into a factor variable. The list must include the following elements:

  - `factor.dummy`: A character vector of dummy variables to be converted to factors.
  - `factor.name`: The name for the newly created factor variable (default is `'factor.var'` if `NULL`).
  - `factor.level`: A vector specifying the factor levels. If `NULL`, levels are taken from `factor.dummy`. If `factor.level` is provided, its length must match `factor.dummy`.

`verbose`
: A logical argument (default `FALSE`) to specify whether to print information collected during the setup of the object(s).

`expose_function`
: A logical value or `NULL` (default `FALSE`). Controls whether Stan functions are exposed for post-processing.

  - `TRUE`: Explicitly exposes Stan functions saved from the model fit. This is required when calculating `fit criteria` or `bayes_R2` during model optimization.
  - `FALSE` (default): Stan functions are not exposed.
  - `NULL`: The setting is inherited from the original `bsitar()` model fit configuration.

  Note: In the `optimize_model()` function, the default is `NULL` (inheriting behavior), whereas other post-processing functions default to `FALSE`.

`usesavedfuns`	A logical value (default `NULL`) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the `expose_functions` argument from the `bsitar()` call. Manual specification of `usesavedfuns` is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.
`clearenvfuns`	A logical value indicating whether to clear the exposed Stan functions from the environment (`TRUE`) or not (`FALSE`). If `NULL`, `clearenvfuns` is set based on the value of `usesavedfuns`: `TRUE` if `usesavedfuns = TRUE`, or `FALSE` if `usesavedfuns = FALSE`.
`funlist`	A list (default `NULL`) specifying function names. This is rarely needed, as required functions are typically retrieved automatically. A use case for `funlist` is when `sigma_formula`, `sigma_formula_gr`, or `sigma_formula_gr_str` use an external function (e.g., `poly(age)`). The `funlist` should include function names defined in the `globalenv()`. For functions needing both distance and velocity curves (e.g., `plot_curves(..., opt = 'dv')`), `funlist` must include two functions: one for the distance curve and one for the velocity curve.
`xvar`	A character string (default `NULL`) specifying the `'x'` variable. Rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.
`difx`	A character string (default `NULL`) specifying the `'x'` variable that should be used for manual differentiation of the distance curve. Internally, the `xvar` is set as `difx` if specified. The argument `difx` is evaluated only when `dpar = 'sigma'`, ignored otherwise. Note that argument `xvar` itself is rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.
`idvar`	A character string (default `NULL`) specifying the `'id'` variable. Rarely used because `idvar` is inferred internally.
`itransform`	A character string (default `NULL`) indicating the variables names that are reverse transformed. Options are `c("x", "y", "sigma")`. The `itransform` is primarily used to get the `xvar` variable at original scale i.e., `itransform = 'x'`. To turn of all transformations, use `itransform = ""`. when `itransform = NULL`, the appropriate transformation for `xvar` is selected automatically. Note that when no match for `xvar` is found in the `data,frame`, the `itransform` will be ignored within the calling function, `'prepare_transformations()'`.
`newdata_fixed`	An indicator to specify whether to check data format and structure for the user provided `newdata`, and apply needed `prepare_data2` and `prepare_transformations` (`newdata_fixed = NULL`, default), return user provided `newdata` (`newdata = TRUE`) as it is without checking for the data format or applying `prepare_data2` and `prepare_transformations` (`newdata_fixed = 0`), check for the data format and if needed, prepare data format using `prepare_data2` (`newdata_fixed = 1`), or apply `prepare_transformations` only assuming that data format is correct (`newdata_fixed = 2`). It is strongly recommended that user either leave the `newdata = NULL` and `newdata_fixed = NULL` in which case data used in the model fitting is automatically retrieved and checked for the required data format and transformations, and if needed, `prepare_data2` and `prepare_transformations` are applied internally. The other flags provided for `newdata_fixed = 0, 1, 2` are mainly for the internal use during post-processing.

`envir`	The environment used for function evaluation. The default is `NULL`, which sets the environment to `parent.frame()`. Since most post-processing functions rely on **brms**, it is recommended to set `envir = globalenv()` or `envir = .GlobalEnv`, especially for derivatives like velocity curves.
`...`	Additional arguments passed to the `brms::fitted.brmsfit()` function. Please see `brms::fitted.brmsfit()` for details on various options available.

Details

The **get_predictions**() function estimates fitted values (via `brms::fitted.brmsfit()`) or the posterior draws from the posterior distribution (via `brms::predict.brmsfit()`) depending on the `type` argument.

Value

An array of predicted mean response values. See brms::fitted.brmsfit for details.

Author(s)

Satpal Sandhu <`satpal.sandhu@bristol.ac.uk`>

See Also

`marginaleffects::predictions()` `marginaleffects::avg_predictions()` `marginaleffects::plot_predictions(`

Examples

```
# Fit Bayesian SITAR model

# To avoid mode estimation which takes time, the Bayesian SITAR model fit to
# the 'berkeley_exdata' has been saved as an example fit ('berkeley_exfit').
# See 'bsitar' function for details on 'berkeley_exdata' and 'berkeley_exfit'.

# Check and confirm whether model fit object 'berkeley_exfit' exists
berkeley_exfit <- getNsObject(berkeley_exfit)

model <- berkeley_exfit

# Population average distance curve
get_predictions(model, deriv = 0, re_formula = NA, draw_ids = 1:2)

# Individual-specific distance curves
get_predictions(model, deriv = 0, re_formula = NULL, draw_ids = 1:2)

# Population average velocity curve
get_predictions(model, deriv = 1, re_formula = NA, draw_ids = 1:2)

# Individual-specific velocity curves
get_predictions(model, deriv = 1, re_formula = NULL, draw_ids = 1:2)
```

`growthparameters.bgmfit` *Estimate growth parameters for the Bayesian SITAR model*

Description

The **growthparameters()** function estimates both population-average and individual-specific growth parameters (e.g., age at peak growth velocity). It also provides measures of uncertainty, including standard errors (SE) and credible intervals (CIs). For a more advanced analysis, consider using the `get_growthparameters()` function, which not only estimates adjusted parameters but also enables comparisons of these parameters across different groups.The `get_growthparameters()` function is based on the **marginaleffects** package.

Usage

```
## S3 method for class 'bgmfit'
growthparameters(
  model,
  newdata = NULL,
  resp = NULL,
  dpar = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  summary = FALSE,
  robust = FALSE,
  transform_draws = NULL,
  scale = c("response", "linear"),
  re_formula = NA,
  peak = TRUE,
  takeoff = FALSE,
  trough = FALSE,
  acgv = FALSE,
  acgv_velocity = 0.1,
  estimation_method = "fitted",
  allow_new_levels = FALSE,
  sample_new_levels = "uncertainty",
  incl_autocor = TRUE,
  numeric_cov_at = NULL,
  levels_id = NULL,
  avg_reffects = NULL,
  aux_variables = NULL,
  grid_add = NULL,
  ipts = NULL,
  model_deriv = TRUE,
  conf = 0.95,
  xrange = NULL,
  xrange_search = NULL,
  digits = 2,
  seed = 123,
  future = FALSE,
  future_session = "multisession",
  cores = NULL,
  parms_eval = FALSE,
  idata_method = NULL,
  parms_method = "getPeak",
  verbose = FALSE,
  fullframe = NULL,
  dummy_to_factor = NULL,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  funlist = NULL,
  xvar = NULL,
  difx = NULL,
  idvar = NULL,
  itransform = NULL,
  newdata_fixed = NULL,
  envir = NULL,
  ...
)

growthparameters(model, ...)
```

Arguments

model	An object of class `bgmfit`.
newdata	An optional data frame for estimation. If `NULL` (default), `newdata` is retrieved from the `model`.
resp	A character string (default `NULL`) to specify the response variable when processing posterior draws for `univariate_by` and `multivariate` models. See `bsitar()` for details on `univariate_by` and `multivariate` models.
dpar	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
ndraws	A positive integer indicating the number of posterior draws to use in estimation. If `NULL` (default), all draws are used.
draw_ids	An integer specifying the specific posterior draw(s) to use in estimation (default `NULL`).
summary	A logical value indicating whether only the estimate should be computed (`TRUE`), or whether the estimate along with SE and CI should be returned (`FALSE`, default). Setting `summary` to `FALSE` will increase computation time. Note that `summary = FALSE` is required to obtain correct estimates when `re_formula = NULL`.
robust	A logical value to specify the summary options. If `FALSE` (default), the mean is used as the measure of central tendency and the standard deviation as the mea-

sure of variability. If TRUE, the median and median absolute deviation (MAD) are applied instead. Ignored if summary is FALSE.

transform_draws
: A function applied to individual draws from the posterior distribution before computing summaries (default NULL). The argument transform_draws is derived from the marginaleffects::predictions() function and should not be confused with the transform argument from the deprecated brms::posterior_predict() function. It's important to note that for both marginaleffects::predictions() and marginaleffects::avg_predictions(), the transform_draws argument takes precedence over the transform argument. Note that when transform_draws = NULL, an attempt is made to automatically set transform_draws = 'exp' for dpar = 'sigma'. User can set transform_draws = FALSE to turn off this automatic assignment of 'exp' to the transform_draws. It is also important to set transform_draws = FALSE when computing the first derivative (velocity) for dpar = 'sigma'.

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.

re_formula
: Option to indicate whether or not to include individual/group-level effects in the estimation. When NA (default), individual-level effects are excluded, and population average growth parameters are computed. When NULL, individual-level effects are included in the computation, and the resulting growth parameters are individual-specific. In both cases (NA or NULL), continuous and factor covariates are appropriately included in the estimation. Continuous covariates are set to their means by default (see numeric_cov_at for details), while factor covariates remain unaltered, allowing for the estimation of covariate-specific population average and individual-specific growth parameters.

peak
: A logical value (default TRUE) indicating whether to calculate the age at peak velocity (APGV) and the peak velocity (PGV) parameters.

takeoff
: A logical value (default FALSE) indicating whether to calculate the age at takeoff velocity (ATGV) and the takeoff growth velocity (TGV) parameters.

trough
: A logical value (default FALSE) indicating whether to calculate the age at cessation of growth velocity (ACGV) and the cessation of growth velocity (CGV) parameters.

acgv
: A logical value (default FALSE) indicating whether to calculate the age at cessation of growth velocity from the velocity curve. If TRUE, the age at cessation of growth velocity (ACGV) and the cessation growth velocity (CGV) are calculated based on the percentage of the peak growth velocity, as defined by the acgv_velocity argument (see below). The acgv_velocity is typically set at 10 percent of the peak growth velocity. ACGV and CGV are calculated along with the uncertainty (SE and CI) around the ACGV and CGV parameters.

acgv_velocity
: The percentage of the peak growth velocity to use when estimating acgv. The default value is 0.10, i.e., 10 percent of the peak growth velocity.

estimation_method
: A character string specifying the estimation method when calculating the velocity from the posterior draws. The 'fitted' method internally calls fitted_draws(),

while the 'predict' method calls predict_draws(). See brms::fitted.brmsfit() and brms::predict.brmsfit() for details.

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.

numeric_cov_at
: An optional (named list) argument to specify the value of continuous covariate(s). The default NULL option sets the continuous covariate(s) to their mean. Alternatively, a named list can be supplied to manually set these values. For example, numeric_cov_at = list(xx = 2) will set the continuous covariate variable 'xx' to 2. The argument numeric_cov_at is ignored when no continuous covariates are included in the model.

levels_id
: An optional argument to specify the ids for the hierarchical model (default NULL). It is used only when the model is applied to data with three or more levels of hierarchy. For a two-level model, levels_id is automatically inferred from the model fit. For models with three or more levels, levels_id is inferred from the model fit under the assumption that hierarchy is specified from the lowest to the uppermost level, i.e., id followed by study, where id is nested within study. However, it is not guaranteed that levels_id is sorted correctly, so it is better to set it manually when fitting a model with three or more levels of hierarchy.

avg_reffects
: An optional argument (default NULL) to calculate (marginal/average) curves and growth parameters, such as APGV and PGV. If specified, it must be a named list indicating the over (typically a level 1 predictor, such as age), feby (fixed effects, typically a factor variable), and reby (typically NULL, indicating that parameters are integrated over the random effects). For example, avg_reffects = list(feby = 'study', reby = NULL, over = 'age').

aux_variables
: An optional argument to specify the variable(s) that can be passed to the ipts argument (see below). This is useful when fitting location-scale models and measurement error models. If post-processing functions throw an error such as variable 'x' not found in either 'data' or 'data2', consider using aux_variables.

`grid_add`	An optional argument to specify the variable(s) that can be passed to the `marginaleffects::datagrid()` This is useful when fitting location-scale models and measurement error models. If post-processing functions throw an error such as `variable 'x' not found in either 'data' or 'data2'`, consider using `grid_add`. Note that unlike `aux_variables` which are passed to the internal data functions such as `'get.newdata'`, the `grid_add` are passed to the `marginaleffects::datagrid()`.
`ipts`	An integer specifying the number of points for interpolating the predictor variable (e.g., age) to generate smooth curves for predictions and plots. This value is used as the `length.out` argument for `seq()`, controlling the smoothness of distance and velocity curves without altering the predictor range. `NULL` **(the default)** Engages automatic behavior based on the `dpar` argument: it is internally set to `50` for the mean response (`dpar = 'mu'`), ensuring smooth curves, while for the distributional parameters (e.g., `dpar = 'sigma'`), no interpolation is performed. **An integer (e.g.,** `100`**)** Explicitly sets the number of interpolation points. `FALSE` Disables all interpolation, forcing predictions to be made only at the original data points of the predictor variable. This argument affects the following post-processing functions: `fitted_draws()`, `predict_draws()`, `growthparameters()`, `plot_curves()`, `get_predictions()`, `get_comparisons()`, and `get_growthparameters()`.
`model_deriv`	A logical value specifying whether to estimate the velocity curve from the derivative function or by differentiating the distance curve. Set `model_deriv = TRUE` for functions that require the velocity curve, such as `growthparameters()` and `plot_curves()`. Set it to `NULL` for functions that use the distance curve (i.e., fitted values), such as `loo_validation()` and `plot_ppc()`.
`conf`	A numeric value (default `0.95`) to compute the confidence interval (CI). Internally, `conf` is translated into paired probability values as `c((1 - conf)/2, 1 - (1 - conf)/2)`. For `conf = 0.95`, this computes a 95% CI where the lower and upper limits are named `Q.2.5` and `Q.97.5`, respectively.
`xrange`	An integer to set the predictor range (e.g., age) when executing the interpolation via `ipts`. By default, `NULL` sets the individual-specific predictor range. Setting `xrange = 1` applies the same range for individuals within the same higher grouping variable (e.g., study). Setting `xrange = 2` applies an identical range across the entire sample. Alternatively, a numeric vector (e.g., `xrange = c(6, 20)`) can be provided to set the range within the specified values.
`xrange_search`	A vector of length two or a character string `'range'` to set the range of the predictor variable (x) within which growth parameters are searched. This is useful when there is more than one peak and the user wants to summarize the peak within a specified range of the `x` variable. The default value is `xrange_search = NULL`.
`digits`	An integer (default 2) to set the decimal places for rounding the results using the `base::round()` function.
`seed`	An integer (default `123`) that is passed to the estimation method to ensure reproducibility.
`future`	A logical value (default `FALSE`) indicating whether to perform parallel computations. If `TRUE`, posterior summaries are computed in parallel using `future.apply::future_sapply()`.

`future_session`	A character string or a named list specifying the parallel plan when `future = TRUE`. • **Character string**: Defaults to `"multisession"`. Use `"multicore"` for forking (not supported on Windows). • **Named list**: Useful for advanced plans like `future.mirai::mirai_cluster()`. The list must contain: – `future_session`: The planner function (e.g., `mirai_cluster`). – Additional named arguments passed to the planner (e.g., daemons for `mirai::daemons()`). Example: `list(future_session = mirai_cluster, daemons = list(...))`.
`cores`	An integer specifying the number of cores for parallel execution. • If `NULL` (default), the number of cores is set to `future::availableCores() - 1`. • On non-Windows systems, this can be controlled globally via the `mc.cores` option.
`parms_eval`	A logical value to specify whether or not to compute growth parameters on the fly. This is for internal use only and is mainly needed for compatibility across internal functions.
`idata_method`	A character string specifying the interpolation method (default `NULL`). The number of interpolation points is controlled by the `ipts` argument. Available options: • `'m1'`: Adapted from the **iapvbs** package (documented here). This method internally constructs the data frame based on the model configuration. *Note:* This method may fail if the model includes covariates (especially in `univariate_by` models). • `'m2'`: Based on the **JMbayes** package (documented here). This method uses the exact data frame from the model fit (`fit$data`) and is generally more robust. If `idata_method = NULL`, `'m2'` is automatically selected. It is recommended to use `'m2'` if `'m1'` encounters errors with covariate-dependent models.
`parms_method`	A character string specifying the method used when evaluating `parms_eval`. The default method is `getPeak`, which uses the `sitar::getPeak()` function from the `sitar` package. Alternatively, `findpeaks` uses the `findpeaks` function from the `pracma` package. This parameter is for internal use and ensures compatibility across internal functions.
`verbose`	A logical argument (default `FALSE`) to specify whether to print information collected during the setup of the object(s).
`fullframe`	A logical value indicating whether to return a `fullframe` object in which `newdata` is bound to the summary estimates. Note that `fullframe` cannot be used with `summary = FALSE`, and it is only applicable when `idata_method = 'm2'`. A typical use case is when fitting a `univariate_by` model. This option is mainly for internal use.
`dummy_to_factor`	A named list (default `NULL`) to convert dummy variables into a factor variable. The list must include the following elements:

- `factor.dummy`: A character vector of dummy variables to be converted to factors.
- `factor.name`: The name for the newly created factor variable (default is `'factor.var'` if `NULL`).
- `factor.level`: A vector specifying the factor levels. If `NULL`, levels are taken from `factor.dummy`. If `factor.level` is provided, its length must match `factor.dummy`.

`expose_function`
: A logical value or `NULL` (default `FALSE`). Controls whether Stan functions are exposed for post-processing.

- `TRUE`: Explicitly exposes Stan functions saved from the model fit. This is required when calculating `fit criteria` or `bayes_R2` during model optimization.
- `FALSE` (default): Stan functions are not exposed.
- `NULL`: The setting is inherited from the original `bsitar()` model fit configuration.

Note: In the `optimize_model()` function, the default is `NULL` (inheriting behavior), whereas other post-processing functions default to `FALSE`.

`usesavedfuns`
: A logical value (default `NULL`) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the `expose_functions` argument from the `bsitar()` call. Manual specification of `usesavedfuns` is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.

`clearenvfuns`
: A logical value indicating whether to clear the exposed Stan functions from the environment (`TRUE`) or not (`FALSE`). If `NULL`, `clearenvfuns` is set based on the value of `usesavedfuns`: `TRUE` if `usesavedfuns = TRUE`, or `FALSE` if `usesavedfuns = FALSE`.

`funlist`
: A list (default `NULL`) specifying function names. This is rarely needed, as required functions are typically retrieved automatically. A use case for `funlist` is when `sigma_formula`, `sigma_formula_gr`, or `sigma_formula_gr_str` use an external function (e.g., `poly(age)`). The `funlist` should include function names defined in the `globalenv()`. For functions needing both distance and velocity curves (e.g., `plot_curves(..., opt = 'dv')`), `funlist` must include two functions: one for the distance curve and one for the velocity curve.

`xvar`
: A character string (default `NULL`) specifying the `'x'` variable. Rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.

`difx`
: A character string (default `NULL`) specifying the `'x'` variable that should be used for manual differentiation of the distance curve. Internally, the `xvar` is set as `difx` if specified. The argument `difx` is evaluated only when `dpar = 'sigma'`, ignored otherwise. Note that argument `xvar` itself is rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.

`idvar`
: A character string (default `NULL`) specifying the `'id'` variable. Rarely used because `idvar` is inferred internally.

`itransform`	A character string (default `NULL`) indicating the variables names that are reverse transformed. Options are `c("x", "y", "sigma")`. The `itransform` is primarily used to get the `xvar` variable at original scale i.e., `itransform = 'x'`. To turn of all transformations, use `itransform = ""`. when `itransform = NULL`, the appropriate transformation for `xvar` is selected automatically. Note that when no match for `xvar` is found in the `data,frame`, the `itransform` will be ignored within the calling function, `'prepare_transformations()'`.
`newdata_fixed`	An indicator to specify whether to check data format and structure for the user provided `newdata`, and apply needed `prepare_data2` and `prepare_transformations` (`newdata_fixed = NULL`, default), return user provided `newdata` (`newdata = TRUE`) as it is without checking for the data format or applying `prepare_data2` and `prepare_transformations` (`newdata_fixed = 0`), check for the data format and if needed, prepare data format using `prepare_data2` (`newdata_fixed = 1`), or apply `prepare_transformations` only assuming that data format is correct (`newdata_fixed = 2`). It is strongly recommended that user either leave the `newdata = NULL` and `newdata_fixed = NULL` in which case data used in the model fitting is automatically retrieved and checked for the required data format and transformations, and if needed, `prepare_data2` and `prepare_transformations` are applied internally. The other flags provided for `newdata_fixed = 0, 1, 2` are mainly for the internal use during post-processing.
`envir`	The environment used for function evaluation. The default is `NULL`, which sets the environment to `parent.frame()`. Since most post-processing functions rely on **brms**, it is recommended to set `envir = globalenv()` or `envir = .GlobalEnv`, especially for derivatives like velocity curves.
`...`	Additional arguments passed to the `brms::fitted.brmsfit()` and `brms::predict()` functions.

Details

The **growthparameters()** function internally calls either the `fitted_draws()` or the `predict_draws()` function to estimate first-derivative growth parameters for each posterior draw. The estimated growth parameters include:

- Age at Peak Growth Velocity (APGV)
- Peak Growth Velocity (PGV)
- Age at Takeoff Growth Velocity (ATGV)
- Takeoff Growth Velocity (TGV)
- Age at Cessation of Growth Velocity (ACGV)
- Cessation Growth Velocity (CGV)

APGV and PGV are estimated using the `sitar::getPeak()` function, while ATGV and TGV are estimated using the `sitar::getTakeoff()` function. The `sitar::getTrough()` function is employed to estimate ACGV and CGV. The parameters from each posterior draw are then summarized to provide estimates along with uncertainty measures (SEs and CIs).

Please note that estimating cessation and takeoff growth parameters may not be possible if there are no distinct pre-peak or post-peak troughs in the data.

Value

A data frame with either five columns (when `summary = TRUE`) or two columns (when `summary = FALSE`, assuming `re_formual = NULL`). The first two columns, common to both scenarios, are `'Parameter'` and `'Estimate'`, representing the growth parameter (e.g., APGV, PGV) and its estimate. When `summary = TRUE`, three additional columns are included: `'Est.Error'` and two columns representing the lower and upper bounds of the confidence intervals, named `Q.2.5` and `Q.97.5` (for the 95% CI). If `re_formual = NULL`, an additional column with individual identifiers (e.g., `id`) is included.

Author(s)

Satpal Sandhu <`satpal.sandhu@bristol.ac.uk`>

Examples

```
# Fit Bayesian SITAR Model

# To avoid mode estimation, which takes time, the Bayesian SITAR model fit
# to the 'berkeley_exdata' has been saved as an example fit ('berkeley_exfit').
# See 'bsitar' function for details on 'berkeley_exdata' and 'berkeley_exfit'.

# Check if the model fit object 'berkeley_exfit' exists and load it
berkeley_exfit <- getNsObject(berkeley_exfit)

model <- berkeley_exfit

# Population average age and velocity during the peak growth spurt
growthparameters(model, re_formula = NA)

# Population average age and velocity during the take-off and peak
# growth spurt (APGV, PGV, ATGV, TGV)
growthparameters(model, re_formula = NA, peak = TRUE, takeoff = TRUE)

# Individual-specific age and velocity during the take-off and peak
# growth spurt (APGV, PGV, ATGV, TGV)
growthparameters(model, re_formula = NULL, peak = TRUE, takeoff = TRUE)
```

`hypothesis_test.bgmfit` *Comprehensive hypothesis testing framework for the Bayesian SITAR model*

Description

Performs non-linear hypothesis testing via `brms::hypothesis()`, Region of Practical Equivalence (ROPE) via `bayestestR::equivalence_test()`, probability of direction (pd) via `bayestestR::p_direction()`, and pairwise and contrast-style comparisons via `marginaleffects::comparisons()`, using a unified interface.

The `ROPE` evaluates whether the parameter values should be accepted or rejected against an explicitly formulated "null hypothesis". It checks the percentage of the posterior credible intervals such as the High Density Intervals (`HDI`) defined as the null region (the `ROPE`). If this percentage is sufficiently low, the null hypothesis is rejected. If this percentage is sufficiently high, the null hypothesis is accepted. If the HDI is completely outside the `ROPE`, the "null hypothesis" for this parameter is "rejected". If the `ROPE` completely covers the HDI, i.e., all most credible values of a parameter are inside the region of practical equivalence, the null hypothesis is accepted. Else, it is undecided whether to accept or reject the null hypothesis. See `bayestestR::equivalence_test()` for further details.

The `PD`, also known as the Maximum Probability of Effect (`MPE`) is the probability that a parameter (described by its posterior distribution) is strictly positive or negative (whichever is the most probable). Although differently expressed, this index is fairly similar (i.e., is strongly correlated) to the frequentest p-value (see details). See `bayestestR::p_direction()` for further details.

Usage

```
## S3 method for class 'bgmfit'
hypothesis_test(
  model,
  by = NULL,
  hypothesis = NULL,
  hypothesis_str = NULL,
  parameters = NULL,
  parameter = NULL,
  class = "b",
  group = "",
  scope = c("standard", "ranef", "coef"),
  robust = FALSE,
  seed = NULL,
  equivalence_test = NULL,
  p_direction = NULL,
  rope_test = NULL,
  pd_test = NULL,
  range = "default",
  method = "direct",
  method_call = "pkg",
  null = NULL,
  as_p = FALSE,
  remove_na = TRUE,
  rvar_col = NULL,
  effects = "fixed",
  component = "conditional",
  evaluate_comparison = NULL,
  comparison_by = NULL,
  comparison_range_null = NULL,
  comparison_range = NULL,
  comparison_null = NULL,
  evaluate_hypothesis = NULL,
  hypothesis_by = NULL,
  hypothesis_range_null = NULL,
  hypothesis_range = NULL,
  hypothesis_null = NULL,
  rope_as_percent = TRUE,
  pd_as_percent = TRUE,
  format = NULL,
  reformat = TRUE,
  estimate_center = NULL,
  estimate_interval = NULL,
  na.rm = TRUE,
  alpha = 0.05,
  ci = 0.95,
  conf_level = NULL,
  probs = c(0.025, 0.975),
  get_range_null_form = NULL,
  get_range_null_value = NULL,
  plot = FALSE,
  digits = 2,
  engine = NULL,
  usesavedfuns = FALSE,
  verbose = FALSE,
  envir = NULL,
  ...
)

hypothesis_test(model, ...)
```

Arguments

`model`	A fitted model object of class `bgmfit`, or a posterior data frame. See Details for engine-specific support.
`by`	Optional grouping variable(s) used to stratify summaries or comparisons. The exact meaning depends on the selected engine and methods.
`hypothesis`	specify a hypothesis test or custom contrast using a number , formula, string equation, vector, matrix, or function. • Number: The null hypothesis used in the computation of Z and p (before applying `transform`). • String: Equation to specify linear or non-linear hypothesis tests. Two-tailed tests must include an equal = sign. One-tailed tests must start with < or >. If the terms in `coef(object)` uniquely identify estimates, they can be used in the formula. Otherwise, use `b1`, `b2`, etc. to identify the position of each parameter. The `b*` wildcard can be used to test hypotheses on all estimates. When the hypothesis string represents a two-sided equation, the `estimate` column holds the value of the left side minus the right side of

the equation. If a named vector is used, the names are used as labels in the output. Examples:

- `hp = drat`
- `hp + drat = 12`
- `b1 + b2 + b3 = 0`
- `b* / b1 = 1`
- `<= 0`
- `>= -3.5`
- `b1 >= 10`

• Formula: `lhs ~ rhs | group`
- `lhs`
  - `ratio` (null = 1)
  - `difference` (null = 0)
  - Leave empty for default value
- `rhs`
  - `pairwise` and `revpairwise`: pairwise differences between estimates in each row.
  - `reference`: differences between the estimates in each row and the estimate in the first row.
  - `sequential`: difference between an estimate and the estimate in the next row.
  - `meandev`: difference between an estimate and the mean of all estimates.
  - `meanotherdev`: difference between an estimate and the mean of all other estimates, excluding the current one.
  - `poly`: polynomial contrasts, as computed by the `stats::contr.poly()` function.
  - `helmert`: Helmert contrasts, as computed by the `stats::contr.helmert()` function. Contrast 2nd level to the first, 3rd to the average of the first two, and so on.
  - `trt_vs_ctrl`: difference between the mean of estimates (except the first) and the first estimate.
  - `I(fun(x))`: custom function to manipulate the vector of estimates `x`. The function `fun()` can return multiple (potentially named) estimates.
- `group` (optional)
  - Column name of `newdata`. Conduct hypothesis tests withing subsets of the data.
- Examples:
  - `~ poly`
  - `~ sequential | groupid`
  - `~ reference`
  - `ratio ~ pairwise`
  - `difference ~ pairwise | groupid`

    * ~ I(x - mean(x)) | groupid
    * ~ I(\(x) c(a = x[1], b = mean(x[2:3]))) | groupid

- Matrix or Vector: Each column is a vector of weights. The the output is the dot product between these vectors of weights and the vector of estimates. The matrix can have column names to label the estimates.
- Function:
  - Accepts an argument x: object produced by a marginaleffects function or a data frame with column rowid and estimate
  - Returns a data frame with columns term and estimate (mandatory) and rowid (optional).
  - The function can also accept optional input arguments: newdata, by, draws.
  - This function approach will not work for Bayesian models or with bootstrapping. In those cases, it is easy to use get_draws() to extract and manipulate the draws directly.
- See the Examples section below and the vignette: https://marginaleffects.com/chapters/hypothesis.html
- Warning: When calling predictions() with type="invlink(link)" (the default in some models), hypothesis is tested and p values are computed on the link scale.

hypothesis_str A character vector specifying one or more non-linear hypothesis concerning parameters of the model. Note that the argument hypothesis_str is the renamed version of the hypothesis argument for the brms::hypothesis(). This renaming is done to avoid conflicts with the hypothesis argument for the marginaleffects::comparison

parameters Regular expression pattern that describes the parameters that should be returned. Meta-parameters (like lp__ or prior_) are filtered by default, so only parameters that typically appear in the summary() are returned. Use parameters to select specific parameters for the output.

parameter A single character string or character vector specifying the growth parameter(s) to estimate. Must be one of the following options, ordered by growth curve progression:

- 'apgv': Age at peak growth velocity - age corresponding to the peak growth rate (i.e. maximum growth velocity) during adolescent growth spurt (default when NULL)
- 'pgv': Peak growth velocity - maximum growth rate during adolescent growth spurt
- 'atgv': Age at takeoff growth velocity - age where initial rapid growth begins accelerating toward peak
- 'tgv': Takeoff growth velocity - initial high growth rate at curve start
- 'acgv': Age at cessation growth velocity - age where growth rate approaches near zero and growth effectively stops
- 'cgv': Cessation growth velocity - final low growth rate approaching maturity

  \item \code{'spgv'}: Size (length/height) at peak growth velocity age, \code{'apgv'}

```
\item \code{'stgv'}: Size at takeoff growth velocity age, \code{'atgv'}
\item \code{'scgv'}: Size at cessation growth velocity age, \code{'acgv'}

\item \code{'satXX'}: Size at specific age XX years of predictor
\code{xvar} (e.g., \code{'sat15'} = size at 15 years, \code{'sat12.5'} =
size at 12.5 years)

\item \code{'all'}: All six primary velocity/age parameters
(\code{'apgv','pgv','atgv','tgv','acgv','cgv'}). Cannot be used when
\code{by = TRUE}.
```

Multiple parameters can be requested simultaneously as a vector, e.g., `c('apgv', 'pgv', 'spgv')`. Custom `'satXX'` format requires no spaces between `'sat'` prefix and numeric age.

`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.

`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).

`equivalence_test`
: A named arguments list (default `NULL`) passed to `bayestestR::equivalence_test()` for ROPE-based hypothesis testing. Core arguments:

  - `range = "default"`: ROPE bounds (numeric vector or `"default"`). Defaults values that are automatically set up for `range` are: `c(-1, 1)` for age parameters (`apgv`, `atgv`, `acgv`); `c(-0.5, 0.5)` for velocity parameters (`pgv`, `tgv`, `cgv`).
  - `ci`: Credible interval level. Default `0.95`.
  - `rvar_col`: Random variable column name for long-format data. Default `NULL`.
  - `verbose`: Print progress messages if `TRUE`. Default `FALSE`.

  Additional package-specific controls:

  - `format`: If `TRUE`, merge `ROPE_low`/`ROPE_high` into `ROPE_range` and `HDI_low`/`HDI_high` into `HDI_range`.

- `reformat`: If `TRUE`, standardize `marginaleffects::comparisons()` output (`conf.low` → `Q2.5`, `conf.high` → `Q97.5`) and drop reporting columns (`term`, `contrast`, etc.).
- `get_range_null_form`: If `TRUE`, return expected `range` and `null` structure for manual specification via `comparison_range_null` and `hypothesis_range_null`.
- `digits`: Number of digits to use when printing numeric results. Default `2`.
- `as_percent`: Logical indicating whether to return ROPE results as percentages (`TRUE`, default) or fractions (`FALSE`). Only evaluated when `model` is class `"bsitar"` and engine is `"bayestestR"` or `"mbcombo"`.
- `na.rm`: If `TRUE` (default), remove `NA` values.
- `inline`: Internal use only; executes custom equivalence function (not for users).

`p_direction` A named arguments list (default `NULL`) passed to `bayestestR::p_direction()` for Probability of Direction (pd) computation. Commonly used elements:

- `method`: Computation method. Default `"direct"`.
- `null`: Null value (default `0` for all growth parameters).
- `as_p`: Return p-value-like scale. Default `FALSE`.
- `remove_na`: Remove missing values. Default `TRUE`.

Additional package-specific controls:

- `get_range_null_form`: If `TRUE`, return expected `range` and `null` structure for manual specification via `comparison_range_null` and `hypothesis_range_null`.
- `digits`: Number of digits to use when printing numeric results. Default `2`.
- `as_percent`: Logical indicating whether to return PD results as percentages (`TRUE`, default) or fractions (`FALSE`). Only evaluated when `model` is class `"bsitar"` and engine is `"bayestestR"` or `"mbcombo"`.
- `na.rm`: If `TRUE` (default), remove `NA` values.
- `inline`: Internal use only; executes custom equivalence function (not for users).

`rope_test` Logical; if `TRUE`, run ROPE-based practical equivalence testing via `bayestestR::equivalence_test()`. This is rarely used because appropriate setting `TRUE/FALSE` is assigned automatically depending on the `equivalence_test`. Note that when `equivalence_test` is used to set up the `rope_test`, then all missing arguments to the `equivalence_test` list are assigned from the default or user defined arguments included in the `hypothesis_test()` call. See `equivalence_test` for details.

`pd_test` Logical; if `TRUE`, compute probability of direction via `bayestestR::p_direction()`. This is rarely used because appropriate setting `TRUE/FALSE` is assigned automatically depending on the `p_direction`. Note that when `p_direction` is used to set up the `pd_test`, then all missing arguments to the `p_direction` list are assigned from the default or user defined arguments included in the `hypothesis_test()` call. See `p_direction` for details.

`range` ROPE's lower and higher bounds. Should be `"default"` or depending on the number of outcome variables a vector or a list. For models with one response, `range` can be:

- a vector of length two (e.g., `c(-0.1, 0.1)`),

	• a list of numeric vector of the same length as numbers of parameters (see 'Examples'). • a list of *named* numeric vectors, where names correspond to parameter names. In this case, all parameters that have no matching name in `range` will be set to `"default"`. In multivariate models, `range` should be a list with another list (one for each response variable) of numeric vectors . Vector names should correspond to the name of the response variables. If `"default"` and input is a vector, the range is set to `c(-0.1, 0.1)`. If `"default"` and input is a Bayesian model, `rope_range()` is used. See 'Examples'.
`method`	Can be `"direct"` or one of methods of `estimate_density()`, such as `"kernel"`, `"logspline"` or `"KernSmooth"`. See details.
`method_call`	A character string to pass `'method'` argument to the `'get_growthparameters'` function. Default `'pkg'`.
`null`	The value considered as a "null" effect. Traditionally 0, but could also be 1 in the case of ratios of change (OR, IRR, ...).
`as_p`	If `TRUE`, the p-direction (pd) values are converted to a frequentist p-value using `pd_to_p()`.
`remove_na`	Should missing values be removed before computation? Note that `Inf` (infinity) are *not* removed.
`rvar_col`	A single character - the name of an `rvar` column in the data frame to be processed. See example in `p_direction()`.
`effects`	Should variables for fixed effects (`"fixed"`), random effects (`"random"`) or both (`"all"`) be returned? Only applies to mixed models. May be abbreviated. For models of from packages **brms** or **rstanarm** there are additional options: • `"fixed"` returns fixed effects. • `"random_variance"` return random effects parameters (variance and correlation components, e.g. those parameters that start with `sd_` or `cor_`). • `"grouplevel"` returns random effects group level estimates, i.e. those parameters that start with `r_`. • `"random"` returns both `"random_variance"` and `"grouplevel"`. • `"all"` returns fixed effects and random effects variances. • `"full"` returns all parameters.
`component`	Which type of parameters to return, such as parameters for the conditional model, the zero-inflated part of the model, the dispersion term, etc. See details in section *Model Components*. May be abbreviated. Note that the *conditional* component also refers to the *count* or *mean* component - names may differ, depending on the modeling package. There are three convenient shortcuts (not applicable to *all* model classes): • `component = "all"` returns all possible parameters. • If `component = "location"`, location parameters such as `conditional`, `zero_inflated`, `smooth_terms`, or `instruments` are returned (everything that are fixed or random effects - depending on the `effects` argument - but no auxiliary parameters).

- For component = "distributional" (or "auxiliary"), components like sigma, dispersion, beta or precision (and other auxiliary parameters) are returned.

evaluate_comparison
: Logical indicating whether to compute and return comparisons from marginaleffects::comparisons() via the get_growthparameters(). Defaults to NULL, in which case it is automatically determined from other arguments. Only evaluated when model is class "bsitar" and engine = "marginaleffects" or "mbcombo".

comparison_by
: Character string specifying the by variable for comparisons. Defaults to NULL, in which case it is inferred from the by argument.

comparison_range_null
: Data frame with columns parameter, by variables, range (for bayestestR::equivalence_test()), and null (for bayestestR::p_direction()). Defaults to NULL, in which case range and null values are set automatically.

comparison_range
: Like comparison_range_null but without the null column. Used when equivalence_test = TRUE and p_direction = FALSE.

comparison_null
: Like comparison_range_null but without the range column. Used when equivalence_test = FALSE and p_direction = TRUE.

evaluate_hypothesis
: Logical indicating whether to compute and return hypothesis from marginaleffects::comparisons() via the get_growthparameters(). Defaults to NULL, in which case it is automatically determined from other arguments. Only evaluated when model is class "bsitar" and engine = "marginaleffects" or "mbcombo".

hypothesis_by
: Character string specifying the by variable for hypotheses. Defaults to NULL, in which case it is inferred from the by argument.

hypothesis_range_null
: Data frame with columns parameter, by variables, range (for bayestestR::equivalence_test()), and null (for bayestestR::p_direction()). Defaults to NULL, in which case range and null values are set automatically.

hypothesis_range
: Like hypothesis_range_null but without the null column. Used when equivalence_test = TRUE and p_direction = FALSE.

hypothesis_null
: Like hypothesis_range_null but without the range column. Used when equivalence_test = FALSE and p_direction = TRUE.

rope_as_percent
: Logical indicating whether to return ROPE results as percentages (TRUE, default) or fractions (FALSE). Only evaluated when model is class "bsitar" and engine is "bayestestR" or "mbcombo".

pd_as_percent
: Logical indicating whether to return PD results as percentages (TRUE, default) or fractions (FALSE). Only evaluated when model is class "bsitar" and engine is "bayestestR" or "mbcombo".

format
: Logical; if TRUE, merge interval-bound columns returned by bayestestR::equivalence_test() into compact range columns. Specifically, ROPE_low and ROPE_high are combined into ROPE_range, and HDI_low and HDI_high are combined into HDI_range. This option only modifies results originating from bayestestR::equivalence_test().

`reformat`	Logical; if `TRUE`, post-process the data frame returned by `marginaleffects::comparisons()` to make it consistent with posterior-summary conventions. This includes renaming interval columns such as `conf.low` to `Q2.5` and `conf.high` to `Q97.5`, and optionally dropping columns which are often not needed for reporting (e.g., `term`, `contrast`, etc.).
`estimate_center`	A character string (default `NULL`) specifying how to center estimates: either `'mean'` or `'median'`. This option sets the global options as follows: `options("marginaleffects_poste` `= "mean")` or `options("marginaleffects_posterior_center" = "median")`. These global options are restored upon function exit using `base::on.exit()`.
`estimate_interval`	A character string (default `NULL`) to specify the type of credible intervals: `'eti'` for equal-tailed intervals or `'hdi'` for highest density intervals. This option sets the global options as follows: `options("marginaleffects_posterior_interval" = "eti")` or `options("marginaleffects_posterior_interval" = "hdi")`, and is restored on exit using `base::on.exit()`.
`na.rm`	Logical; if `TRUE` (default), then remove `NA` values.
`alpha`	Alpha level for hypothesis tests used in `brms::hypothesis()`. The default is `NULL`, in which case `alpha` is inferred from `probs`. With the default `probs`, this corresponds to `alpha = 0.05`, which is also the default in `brms::hypothesis()`.
`ci`	Confidence level used in `bayestestR::equivalence_test()` and `bayestestR::p_direction()`. The default is `NULL`, in which case `ci` is inferred from `probs`. With the default `probs`, this corresponds to `ci = 0.95`, which matches the defaults in those functions.
`conf_level`	Confidence level used in `marginaleffects::comparisons()`. The default is `NULL`, in which case `conf_level` is inferred from `probs`. With the default `probs`, this corresponds to `conf_level = 0.95`, which is also the default in `marginaleffects::comparisons()`. Note that if `probs = NULL`, then `conf_level` is used to determine the `probs` as well as the `alpha` and `ci` (see below).
`probs`	Probability levels for credible intervals. Defaults to `c(0.025, 0.975)` (95\ values for `alpha`, `ci`, and `conf_level` when those are `NULL` (see below).
`get_range_null_form`	Logical; if `TRUE`, return the expected structure of `range` (`bayestestR::equivalence_test()`) and `null` (`bayestestR::p_direction()`) values. User can use this format to set the `range` and `null` values that are then passed to the `bayestestR::equivalence_test()` and `bayestestR::equivalence_test()` via `comparison_range_null` and `hypothesis_range_null`.
`get_range_null_value`	Logical; if `TRUE`, return the default `range` (`bayestestR::equivalence_test()`) and `null` (`bayestestR::p_direction()`) values that are used to set up the `range` and `null` values for `bayestestR::equivalence_test()` and `bayestestR::equivalence_test(`
`plot`	Logical or a character string `"return"`; if `TRUE`, plot the hypothesis test results and return the data frame. Set `plot = "return"` to return the plot object itself instead of the data frame. Note that only `brms::hypothesis()` and `bayestestR::equivalence_test()` support plotting.
`digits`	Number of digits to use when printing numeric results. Default `2`.
`engine`	Optional engine selector specifying which backend(s) to use:

- `"brms"` for `brms::hypothesis()` non-linear hypothesis testing
- `"bayestestR"` for `bayestestR::equivalence_test()` (ROPE) and `bayestestR::p_direction(` (PD) testing
- `"marginaleffects"` for `marginaleffects::comparisons()` pairwise/contrast comparisons
- `"mbcombo"` for combined **marginaleffects** and **bayestestR** workflows
- `NULL` (default) automatically selects appropriate engine(s) based on `model` class and requested analyses

`usesavedfuns`	A logical value (default `NULL`) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the `expose_functions` argument from the `bsitar()` call. Manual specification of `usesavedfuns` is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.
`verbose`	Toggle off warnings.
`envir`	The environment used for function evaluation. The default is `NULL`, which sets the environment to `parent.frame()`. Since most post-processing functions rely on **brms**, it is recommended to set `envir = globalenv()` or `envir = .GlobalEnv`, especially for derivatives like velocity curves.
`...`	Additional arguments forwarded to `get_growthparameters()`

Details

This function collects arguments from multiple upstream packages:

- `hypothesis_test` arguments (e.g., `hypothesis`, `class`, `scope`, `alpha`, `robust`, `seed`) map to `brms::hypothesis()` and, in particular, `scope` controls where hypothesis components referenced in `hypothesis` are looked up. Note that to avoid conflicts with the `hypothesis` argument from `marginaleffects::comparisons()`, the `hypothesis` argument for the brms package is renamed as `hypothesis_str`. In other words, the `hypothesis_str` (see below) is used to setup the `hypothesis` argument when calling the `brms::hypothesis()`.
- `equivalence_test` arguments (e.g., `range`, `ci`, `remove_na`, `parameters`) map to `bayestestR::equivalence_test(` for ROPE-based practical equivalence decisions.
- `p_direction` arguments (e.g., `method`, `null`, `ci`, `remove_na`, `parameters`) map to `bayestestR::p_direction()` for probability of direction based practical decisions.
- Additional arguments in `...` are forwarded to `marginaleffects::comparisons()` via `get_growthparameters` to compute contrasts between predictions under different predictor values.

Users should supply either `hypothesis_str` (**brms** hypothesis), `parameters` (**bayestestR**), or `parameter` (growth-parameter selector used by this package), but not multiple overlapping selectors.

Supported inputs: `bgmfit` objects (via all three packages), posterior data frames (via **brms** and **bayestestR** methods), or `brmsfit` objects (via brms).

Value

An object (typically a list) containing some or all of the following components, depending on the requested analyses and available methods:

- A `brms::hypothesis()` result object for hypothesis tests.
- A `bayestestR::equivalence_test()` result for ROPE and practical equivalence.
- A `bayestestR::p_direction()` result for probability of direction.
- A `marginaleffects::comparisons()` result for contrasts between predictions.

Author(s)

Satpal Sandhu <`satpal.sandhu@bristol.ac.uk`>

See Also

- `brms::hypothesis()` for hypothesis testing.
- `bayestestR::equivalence_test()` for ROPE tests.
- `bayestestR::p_direction()` for probability of direction tests.
- `marginaleffects::comparisons()` for prediction contrasts.

Examples

```
# Fit Bayesian SITAR model

# To avoid mode estimation which takes time, the Bayesian SITAR model fit to
# the 'berkeley_exdata' has been saved as an example fit ('berkeley_exfit').
# See 'bsitar' function for details on 'berkeley_exdata' and 'berkeley_exfit'.

# Check and confirm whether the model fit object 'berkeley_exfit' exists
 berkeley_exfit <- getNsObject(berkeley_exfit)

model <- berkeley_exfit

# Speed up examples by using a subset of posterior draws
draw_ids <- 1:5

## Example 1: Hypothesis testing (brms-style)

set_hypothesis <- c("a_Intercept = 0", "b_Intercept > 1")

# brms reference
hyp_ex1 <- brms::hypothesis(model, hypothesis = set_hypothesis, draw_ids = draw_ids)

# hypothesis_test (auto-detects brms engine)
hyp_ex11 <- hypothesis_test(model,
                           draw_ids = draw_ids,
                           hypothesis_str = set_hypothesis)

# If see package is installed (install.packages("see")), then you can plot it
# plot(hyp_ex1)
# plot(hyp_ex11)

## Example 2: ROPE equivalence testing (bayestestR)
```

```
set_parameters <- c("b_a_Intercept", "b_b_Intercept")
set_range <- list(b_a_Intercept = c(100, 150), b_b_Intercept = c(-2, 2))

# bayestestR reference - If you installed bayestestR package
# hyp_ex1 <- bayestestR::equivalence_test(model,
#                                       parameters = set_parameters,
#                                       range = set_range,
#                                       draw_ids = draw_ids)

# hypothesis_test (auto-detects bayestestR engine)
# If see package is installed (install.packages("see")), then you can plot
# the results by setting plot = TRUE
hyp_ex11 <- hypothesis_test(model,
                           draw_ids = draw_ids,
                           parameters = set_parameters,
                           range = set_range,
                           plot = FALSE)

# print(hyp_ex1)
print(hyp_ex11)

# Note: bayestestR often ignores parameters/effects for nonlinear bsitar models.
# hypothesis_test correctly respects user-specified parameters.

## Example 3: p-direction testing (bayestestR)

set_parameters <- c("b_a_Intercept", "b_b_Intercept")
set_null <- list(b_a_Intercept = 0, b_b_Intercept = 0)

# bayestestR reference - If you installed bayestestR package
# hyp_ex1 <- bayestestR::p_direction(model,
#                                  parameters = set_parameters,
#                                  null = set_null,
#                                  draw_ids = draw_ids)

# hypothesis_test (auto-detects bayestestR engine)
# If see package is installed (install.packages("see")), then you can plot
# the results by setting plot = TRUE
hyp_ex11 <- hypothesis_test(model,
                           draw_ids = draw_ids,
                           parameters = set_parameters,
                           null = set_null,
                           plot = FALSE,
                           equivalence_test = FALSE,
                           p_direction = TRUE)

# print(hyp_ex1)
print(hyp_ex11)

## Example 4: Growth parameter hypothesis testing (marginaleffects)

set_parameter <- c("apgv", "pgv")
set_range <- list(apgv = 1, pgv = c(0, 0.5))
```

```
set_null <- list(apgv = 1, pgv = 0)

# Directly using get_growthparameters()
hyp_ex1 <- get_growthparameters(model,
                                    parameter = set_parameter,
                                    draw_ids = draw_ids)
# Uncomment for ROPE/p-direction tests:
# equivalence_test = list(range = set_range),
# p_direction = list(null = set_null)

## Example 5: hypothesis_test() with growth parameters

hyp_ex1 <- hypothesis_test(model,
                          parameter = set_parameter,
                          equivalence_test = list(range = set_range),
                          p_direction = list(null = set_null),
                          draw_ids = draw_ids)

## Example 6: Pass posterior draws from get_growthparameters()

# pdrawsp = TRUE returns draws for downstream hypothesis testing
hyp_draws <- get_growthparameters(model,
                                      parameter = set_parameter,
                                      pdrawsp = TRUE,
                                      equivalence_test = list(range = set_range),
                                      p_direction = list(null = set_null),
                                      draw_ids = draw_ids)

# Use draws directly in hypothesis_test()
hyp_ex1 <- hypothesis_test(hyp_draws,
                          parameter = set_parameter,
                          equivalence_test = list(range = set_range),
                          p_direction = list(null = set_null),
                          draw_ids = draw_ids)
```

`is.bgmfit` *Checks if argument is a* `bgmfit` *object*

Description

Checks if argument is a `bgmfit` object

Usage

```
is.bgmfit(x)
```

Arguments

x	An R object

loo_validation.bgmfit *Leave-one-out (LOO) cross-validation for the Bayesian SITAR model*

Description

The **loo_validation()** function is a wrapper around the `brms::loo()` function to perform approximate leave-one-out cross-validation based on the posterior likelihood. See `brms::loo()` for more details.

Usage

```
## S3 method for class 'bgmfit'
loo_validation(
  model,
  compare = TRUE,
  resp = NULL,
  dpar = 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,
  ndraws = NULL,
  draw_ids = NULL,
  cores = 1,
  model_deriv = NULL,
  verbose = FALSE,
  dummy_to_factor = NULL,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  envir = NULL,
  ...
)

loo_validation(model, ...)
```

Arguments

model	An object of class `bgmfit`.

`compare`	A logical flag indicating if the information criteria of the models should be compared using `loo::loo_compare()`.
`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.
`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`	A logical flag to indicate whether `loo::loo_moment_match()` should be applied to problematic observations. Defaults to `FALSE`. For most models, moment matching will only work if `save_pars = save_pars(all = TRUE)` was set when fitting the model with `brms::brm()`. See `brms::loo_moment_match()` for more details.
`reloo`	A logical flag indicating whether `brms::reloo()` should be applied to 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`	An optional `list` of additional arguments passed to `loo::loo_moment_match()`.
`reloo_args`	An optional `list` of additional arguments passed to `brms::reloo()`.
`model_names`	If `NULL` (the default) will use model names derived from deparsing the call. Otherwise will use the passed values as model names.
`ndraws`	A positive integer indicating the number of posterior draws to use in estimation. If `NULL` (default), all draws are used.
`draw_ids`	An integer specifying the specific posterior draw(s) to use in estimation (default `NULL`).
`cores`	An integer specifying the number of cores for parallel execution. • If `NULL` (default), the number of cores is set to `future::availableCores() - 1`. • On non-Windows systems, this can be controlled globally via the `mc.cores` option.
`model_deriv`	A logical value specifying whether to estimate the velocity curve from the derivative function or by differentiating the distance curve. Set `model_deriv = TRUE` for functions that require the velocity curve, such as `growthparameters()` and `plot_curves()`. Set it to `NULL` for functions that use the distance curve (i.e., fitted values), such as `loo_validation()` and `plot_ppc()`.
`verbose`	A logical argument (default `FALSE`) to specify whether to print information collected during the setup of the object(s).

`dummy_to_factor`
: A named list (default `NULL`) to convert dummy variables into a factor variable. The list must include the following elements:
  - `factor.dummy`: A character vector of dummy variables to be converted to factors.
  - `factor.name`: The name for the newly created factor variable (default is `'factor.var'` if `NULL`).
  - `factor.level`: A vector specifying the factor levels. If `NULL`, levels are taken from `factor.dummy`. If `factor.level` is provided, its length must match `factor.dummy`.

`expose_function`
: A logical value or `NULL` (default `FALSE`). Controls whether Stan functions are exposed for post-processing.
  - `TRUE`: Explicitly exposes Stan functions saved from the model fit. This is required when calculating `fit criteria` or `bayes_R2` during model optimization.
  - `FALSE` (default): Stan functions are not exposed.
  - `NULL`: The setting is inherited from the original `bsitar()` model fit configuration.

  Note: In the `optimize_model()` function, the default is `NULL` (inheriting behavior), whereas other post-processing functions default to `FALSE`.

`usesavedfuns`
: A logical value (default `NULL`) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the `expose_functions` argument from the `bsitar()` call. Manual specification of `usesavedfuns` is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.

`clearenvfuns`
: A logical value indicating whether to clear the exposed Stan functions from the environment (`TRUE`) or not (`FALSE`). If `NULL`, `clearenvfuns` is set based on the value of `usesavedfuns`: `TRUE` if `usesavedfuns = TRUE`, or `FALSE` if `usesavedfuns = FALSE`.

`envir`
: The environment used for function evaluation. The default is `NULL`, which sets the environment to `parent.frame()`. Since most post-processing functions rely on **brms**, it is recommended to set `envir = globalenv()` or `envir = .GlobalEnv`, especially for derivatives like velocity curves.

`...`
: Additional arguments passed to the `brms::loo()` function. Please see `brms::loo` for details on various options available.

Details

The function supports model comparisons using `loo::loo_compare()` for comparing information criteria across models. For `bgmfit` objects, `LOO` is simply an alias for `loo`. Additionally, you can use `brms::add_criterion()` to store information criteria in the fitted model object for later use.

Value

If only one model object is provided, an object of class `loo` is returned. If multiple objects are provided, an object of class `loolist` is returned.

Author(s)

Satpal Sandhu <satpal.sandhu@bristol.ac.uk>

See Also

brms::loo()

Examples

```
# Fit Bayesian SITAR model

# To avoid mode estimation which takes time, the Bayesian SITAR model fit to
# the 'berkeley_exdata' has been saved as an example fit ('berkeley_exfit').
# See 'bsitar' function for details on 'berkeley_exdata' and 'berkeley_exfit'.

# Check and confirm whether model fit object 'berkeley_exfit' exists
berkeley_exfit <- getNsObject(berkeley_exfit)

model <- berkeley_exfit

# Perform leave-one-out cross-validation
loo_validation(model, cores = 1)
```

modelbased_growthparameters.bgmfit

Estimate model-based growth parameters for the Bayesian SITAR model

Description

The **modelbased_growthparameters()** function estimates individual growth parameters by mapping population average estimate of age of interest (such as age at peak growth velocity or age at take off) on to the individual velocity curves defined by individual level random effects. Note that option `'dpar'` can not be used along with `'nlpar'` in brms::posterior_linpred().

Usage

```
## S3 method for class 'bgmfit'
modelbased_growthparameters(
  model,
  resp = NULL,
  dpar = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  newdata = NULL,
  datagrid = NULL,
  re_formula = NA,
  newdata2 = NULL,
  allow_new_levels = FALSE,
  sample_new_levels = "gaussian",
  parameter = NULL,
  xrange = 1,
  acg_velocity = 0.1,
  digits = 2,
  numeric_cov_at = NULL,
  aux_variables = NULL,
  levels_id = NULL,
  avg_reffects = NULL,
  idata_method = NULL,
  ipts = FALSE,
  seed = 123,
  future = FALSE,
  future_session = "multisession",
  future_splits = TRUE,
  future_method = "future",
  future_re_expose = NULL,
  usedtplyr = FALSE,
  usecollapse = TRUE,
  parallel = FALSE,
  cores = NULL,
  fullframe = FALSE,
  average = FALSE,
  plot = FALSE,
  showlegends = NULL,
  variables = NULL,
  deriv = NULL,
  model_deriv = NULL,
  method = "custom",
  marginals = NULL,
  preparms = NULL,
  pdraws = FALSE,
  pdrawso = FALSE,
  pdrawsp = FALSE,
  pdrawsh = FALSE,
  comparison = "difference",
  type = NULL,
  by = FALSE,
  bys = NULL,
  conf_level = 0.95,
  transform = NULL,
  transform_draws = NULL,
  cross = FALSE,
  wts = NULL,
  hypothesis = NULL,
  equivalence = NULL,
  eps = NULL,
  constrats_by = FALSE,
  constrats_at = FALSE,
  constrats_subset = FALSE,
  reformat = NULL,
  estimate_center = NULL,
  estimate_interval = NULL,
  dummy_to_factor = NULL,
  verbose = FALSE,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  funlist = NULL,
  xvar = NULL,
  idvar = NULL,
  difx = NULL,
  itransform = NULL,
  incl_autocor = TRUE,
  parameter_method = 1,
  subset_by = NULL,
  add_xtm = FALSE,
  call_function = "R",
  newdata_fixed = NULL,
  envir = NULL,
  ...
)

modelbased_growthparameters(model, ...)
```

Arguments

`model`	An object of class `bgmfit`.
`resp`	A character string (default `NULL`) to specify the response variable when processing posterior draws for `univariate_by` and `multivariate` models. See `bsitar()` for details on `univariate_by` and `multivariate` models.
`dpar`	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
`ndraws`	A positive integer indicating the number of posterior draws to use in estimation. If `NULL` (default), all draws are used.
`draw_ids`	An integer specifying the specific posterior draw(s) to use in estimation (default `NULL`).
`newdata`	An optional data frame for estimation. If `NULL` (default), `newdata` is retrieved from the `model`.
`datagrid`	A grid of user-specified values to be used in the `newdata` argument of various functions in the **marginaleffects** package. This allows you to define the regions of the predictor space where you want to evaluate the quantities of interest. See

marginaleffects::datagrid() for more details. By default, the datagrid is set to NULL, meaning no custom grid is constructed. To set a custom grid, the argument should either be a data frame created using marginaleffects::datagrid(), or a named list, which is internally used for constructing the grid. For convenience, you can also pass an empty list datagrid = list(), in which case essential arguments like model and newdata are inferred from the respective arguments specified elsewhere. Additionally, the first-level predictor (such as age) and any covariates included in the model (e.g., gender) are automatically inferred from the model object.

re_formula	Option to indicate whether or not to include individual/group-level effects in the estimation. When NA (default), individual-level effects are excluded, and population average growth parameters are computed. When NULL, individual-level effects are included in the computation, and the resulting growth parameters are individual-specific.
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.
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.
parameter	A single character string or a character vector specifying the growth parameter(s) to be estimated. Options include age at peak growth velocity ('apgv') and 'atgv' (age at takeoff growth velocity). The corresponding distance and velocity at 'apgv' and 'atgv' are computed by default.
xrange	An integer to set the predictor range (e.g., age) when executing the interpolation via ipts. By default, NULL sets the individual-specific predictor range. Setting xrange = 1 applies the same range for individuals within the same higher grouping variable (e.g., study). Setting xrange = 2 applies an identical range across the entire sample. Alternatively, a numeric vector (e.g., xrange = c(6, 20)) can be provided to set the range within the specified values.
acg_velocity	A real number specifying the percentage of peak growth velocity to be used as the cessation velocity when estimating the cgv and acgv growth parameters. The acg_velocity should be greater than 0 and less than 1. The default value

	of `acg_velocity = 0.10` indicates that 10 percent of the peak growth velocity will be used to calculate the cessation growth velocity and the corresponding age at cessation velocity. For example, if the peak growth velocity estimate is `10 mm/year`, then the cessation growth velocity will be `1 mm/year`.
`digits`	An integer (default `2`) to set the decimal places for rounding the results using the `base::round()` function.
`numeric_cov_at`	An optional (named list) argument to specify the value of continuous covariate(s). The default `NULL` option sets the continuous covariate(s) to their mean. Alternatively, a named list can be supplied to manually set these values. For example, `numeric_cov_at = list(xx = 2)` will set the continuous covariate variable 'xx' to 2. The argument `numeric_cov_at` is ignored when no continuous covariates are included in the model.
`aux_variables`	An optional argument to specify the variable(s) that can be passed to the `ipts` argument (see below). This is useful when fitting location-scale models and measurement error models. If post-processing functions throw an error such as `variable 'x' not found in either 'data' or 'data2'`, consider using `aux_variables`.
`levels_id`	An optional argument to specify the `ids` for the hierarchical model (default `NULL`). It is used only when the model is applied to data with three or more levels of hierarchy. For a two-level model, `levels_id` is automatically inferred from the model fit. For models with three or more levels, `levels_id` is inferred from the model fit under the assumption that hierarchy is specified from the lowest to the uppermost level, i.e., `id` followed by `study`, where `id` is nested within `study`. However, it is not guaranteed that `levels_id` is sorted correctly, so it is better to set it manually when fitting a model with three or more levels of hierarchy.
`avg_reffects`	An optional argument (default `NULL`) to calculate (marginal/average) curves and growth parameters, such as APGV and PGV. If specified, it must be a named list indicating the `over` (typically a level 1 predictor, such as age), `feby` (fixed effects, typically a factor variable), and `reby` (typically `NULL`, indicating that parameters are integrated over the random effects). For example, `avg_reffects = list(feby = 'study', reby = NULL, over = 'age')`.
`idata_method`	A character string specifying the interpolation method (default `NULL`). The number of interpolation points is controlled by the `ipts` argument. Available options: • `'m1'`: Adapted from the **iapvbs** package (documented here). This method internally constructs the data frame based on the model configuration. *Note:* This method may fail if the model includes covariates (especially in `univariate_by` models). • `'m2'`: Based on the **JMbayes** package (documented here). This method uses the exact data frame from the model fit (`fit$data`) and is generally more robust. If `idata_method = NULL`, `'m2'` is automatically selected. It is recommended to use `'m2'` if `'m1'` encounters errors with covariate-dependent models.
`ipts`	An integer specifying the number of points for interpolating the predictor variable (e.g., age) to generate smooth curves for predictions and plots. This value

is used as the `length.out` argument for `seq()`, controlling the smoothness of distance and velocity curves without altering the predictor range.

`NULL` **(the default)** Engages automatic behavior based on the `dpar` argument: it is internally set to `50` for the mean response (`dpar = 'mu'`), ensuring smooth curves, while for the distributional parameters (e.g., `dpar = 'sigma'`), no interpolation is performed.

An integer (e.g., `100`**)** Explicitly sets the number of interpolation points.

`FALSE` Disables all interpolation, forcing predictions to be made only at the original data points of the predictor variable.

This argument affects the following post-processing functions: `fitted_draws()`, `predict_draws()`, `growthparameters()`, `plot_curves()`, `get_predictions()`, `get_comparisons()`, and `get_growthparameters()`.

`seed` An integer (default `123`) that is passed to the estimation method to ensure reproducibility.

`future` A logical value (default `FALSE`) indicating whether to perform parallel computations. If `TRUE`, posterior summaries are computed in parallel using `future.apply::future_sapply()`.

`future_session` A character string or a named list specifying the parallel plan when `future = TRUE`.

- **Character string**: Defaults to `"multisession"`. Use `"multicore"` for forking (not supported on Windows).
- **Named list**: Useful for advanced plans like `future.mirai::mirai_cluster()`. The list must contain:
  - `future_session`: The planner function (e.g., `mirai_cluster`).
  - Additional named arguments passed to the planner (e.g., daemons for `mirai::daemons()`).

  Example: `list(future_session = mirai_cluster, daemons = list(...))`.

`future_splits` A list, numeric vector, or logical value (default `TRUE`). Controls how posterior draws are partitioned into smaller subsets for parallel computation. This helps manage memory and improve performance, particularly on Linux when using `future::plan("multicore")`.

Input options:

- **List**: (e.g., `list(1:6, 7:10)`). Each element is a sequence of numbers passed to `draw_ids` to process in a separate chunk.
- **Numeric vector of length 2**: (e.g., `c(100, 4)`). The first element is the total number of draws, and the second is the number of splits. Indices are generated via `parallel::splitIndices()`.
- **Numeric vector of length 1**: Specifies the total number of draws. Splits are calculated automatically.
- `TRUE`: Automatically creates splits based on `ndraws` and `cores`. If both `ndraws` and `draw_ids` are specified, `draw_ids` takes precedence. This is consistent with post-processing functions in the `bsitar` and `brms` packages.
- `FALSE`: Splitting is disabled.

Note: On Windows with `future::plan("multisession")`, background R processes may not free resources automatically. If needed, use `installr::kill_all_Rscript_s()` to terminate them.

`future_method`	A character string (default `'future'`) specifying the parallel computation backend. Options include: • `'future'`: Uses `future.apply::future_lapply()` for execution. • `'dofuture'`: Uses `foreach::foreach()` via the `doFuture` package.
`future_re_expose`	A logical value (default `NULL`) indicating whether to re-expose internal `Stan` functions when `future = TRUE`. This is critical when `future::plan()` is set to `"multisession"`, as compiled C++ functions cannot be exported across distinct R sessions. • If `NULL` (default), it is automatically set to `TRUE` when the plan is `"multisession"`. • Explicitly setting this to `TRUE` is recommended for improved performance during parallel execution.
`usedtplyr`	A logical (default `FALSE`) indicating whether to use the **dtplyr** package for summarizing the draws. This package uses **data.table** as a back-end. It is useful when the data has a large number of observations. For typical use cases, it does not make a significant performance difference. The `usedtplyr` argument is evaluated only when `method = 'custom'`.
`usecollapse`	A logical (default `FALSE`) to indicate whether to use the **collapse** package for summarizing the draws.
`parallel`	A logical (default `FALSE`) indicating whether to use parallel computation (via **doParallel** and **foreach**) when **usecollapse = TRUE**. When `parallel = TRUE`, `parallel::makeCluster()` sets the cluster type as `"PSOCK"`, which works on all operating systems, including `Windows`. If you want to use a faster option for Unix-based systems, you can set `parallel = "FORK"`, but note that it is not compatible with `Windows` systems.
`cores`	An integer specifying the number of cores for parallel execution. • If `NULL` (default), the number of cores is set to `future::availableCores() - 1`. • On non-Windows systems, this can be controlled globally via the `mc.cores` option.
`fullframe`	A logical value indicating whether to return a `fullframe` object in which `newdata` is bound to the summary estimates. Note that `fullframe` cannot be used with `summary = FALSE`, and it is only applicable when `idata_method = 'm2'`. A typical use case is when fitting a `univariate_by` model. This option is mainly for internal use.
`average`	A logical value indicating whether to internally call the `marginaleffects::comparisons()` or the `marginaleffects::avg_comparisons()` function. If `FALSE` (default), `marginaleffects::comparisons()` is called, otherwise `marginaleffects::avg_comparisons()` is used when `average = TRUE`.
`plot`	A logical value specifying whether to plot comparisons by calling the `marginaleffects::plot_compari` function (`TRUE`) or not (`FALSE`). If `FALSE` (default), then `marginaleffects::comparisons()` or `marginaleffects::avg_comparisons()` are called to compute predictions (see the `average` argument for details).
`showlegends`	A logical value to specify whether to show legends (`TRUE`) or not (`FALSE`). If `NULL` (default), the value of `showlegends` is internally set to `TRUE` if `re_formula = NA`, and `FALSE` if `re_formula = NULL`.

`variables`	A named list specifying the level 1 predictor, such as `age` or `time`, used for estimating growth parameters in the current use case. The `variables` list is set via the `esp` argument (default value is `1e-6`). If `variables` is `NULL`, the relevant information is retrieved internally from the `model`. Alternatively, users can define `variables` as a named list, e.g., `variables = list('x' = 1e-6)` where `'x'` is the level 1 predictor. By default, `variables = list('age' = 1e-6)` in the **marginaleffects** package, as velocity is usually computed by differentiating the distance curve using the `dydx` approach. When using this default, the argument `deriv` is automatically set to `0` and `model_deriv` to `FALSE`. If parameters are to be estimated based on the model's first derivative, `deriv` must be set to `1` and `variables` will be defined as `variables = list('age' = 0)`. Note that if the default behavior is used (`deriv = 0` and `variables = list('x' = 1e-6)`), additional arguments cannot be passed to `variables`. In contrast, when using an alternative approach (`deriv = 0` and `variables = list('x' = 0)`), additional options can be passed to the `marginaleffects::comparisons()` and `marginaleffects::avg_comparisons()` functions.
`deriv`	A numeric value specifying whether to estimate parameters based on the differentiation of the distance curve or the model's first derivative. Please refer to the `variables` argument for more details.
`model_deriv`	A logical value specifying whether to estimate the velocity curve from the derivative function or by differentiating the distance curve. Set `model_deriv = TRUE` for functions that require the velocity curve, such as `growthparameters()` and `plot_curves()`. Set it to `NULL` for functions that use the distance curve (i.e., fitted values), such as `loo_validation()` and `plot_ppc()`.
`method`	A character string indicating whether to compute estimates using the `'marginaleffects'` package (`method = 'pkg'`) or custom functions for efficiency and speed (`method = 'custom'`, default). The `method = 'pkg'` option is only suitable for simple cases and should be used with caution. `method = 'custom'` is the preferred option because it allows for simultaneous estimation of multiple parameters (e.g., `'apgv'` and `'pgv'`). This method works during the post-draw stage, supports multiple parameter comparisons via the `hypothesis` argument, and allows users to add or return draws (see `pdraws` for details). If `method = 'pkg'`, the `by` argument must not contain the predictor (e.g., `age`), and `variables` must either be `NULL` (which defaults to `list(age = 1e-6)`) or a list with factor variables like `variables = list(class = 'pairwise')` or `variables = list(age = 1e-6, class = 'pairwise')`. With `method = 'custom'`, the `by` argument can include predictors, which will be ignored, and `variables` should not contain predictors, but can accept factor variables as a vector (e.g., `variables = c('class')`). Using `method = 'custom'` is strongly recommended for better performance and flexibility.
`marginals`	A `list`, `data.frame`, or `tibble` of velocity returned by the **marginaleffects** functions (default `NULL`). This is only evaluated when `method = 'custom'`. The `marginals` can be the output from **marginaleffects** functions or posterior draws from `marginaleffects::posterior_draws()`. The `marginals` argument is primarily used for internal purposes only.
`preparms`	A `list`, `data.frame`, or `tibble` of pre computed parameters (default `NULL`). This is only evaluated when `method = 'custom'`. The `preparms` argument is primarily used for internal purposes only.

`pdraws`	A character string (default `FALSE`) that indicates whether to return the raw posterior draws. Options include: • `'return'`: returns the raw draws, • `'add'`: adds the raw draws to the final return object, • `'returns'`: returns the summary of the raw draws, • `'adds'`: adds the summary of raw draws to the final return object. The `pdraws` are the velocity estimates for each posterior sample. For more details, see `marginaleffects::posterior_draws()`.
`pdrawso`	A character string (default `FALSE`) to indicate whether to return the original posterior draws for parameters. Options include: • `'return'`: returns the original posterior draws, • `'add'`: adds the original posterior draws to the outcome. When `pdrawso = TRUE`, the default behavior is `pdrawso = 'return'`. Note that the posterior draws are returned before calling `marginaleffects::posterior_draws()`.
`pdrawsp`	A character string (default `FALSE`) to indicate whether to return the posterior draws for parameters. Options include: • `'return'`: returns the posterior draws for parameters, • `'add'`: adds the posterior draws to the outcome. When `pdrawsp = TRUE`, the default behavior is `pdrawsp = 'return'`. The `pdrawsp` represent the parameter estimates for each of the posterior samples, and the summary of these are the estimates returned.
`pdrawsh`	A character string (default `FALSE`) to indicate whether to return the posterior draws for parameter contrasts. Options include: • `'return'`: returns the posterior draws for contrasts. The summary of posterior draws for parameters is the default returned object. The `pdrawsh` represent the contrast estimates for each of the posterior samples, and the summary of these are the contrast returned.
`comparison`	A character string specifying the comparison type for growth parameter estimation. Options are `'difference'` and `'differenceavg'`. This argument sets up the internal function for estimating parameters using `sitar::getPeak()`, `sitar::getTakeoff()`, and `sitar::getTrough()` functions. These options are restructured according to the user-specified `hypothesis` argument.
`type`	string indicates the type (scale) of the predictions used to compute contrasts or slopes. This can differ based on the model type, but will typically be a string such as: "response", "link", "probs", or "zero". When an unsupported string is entered, the model-specific list of acceptable values is returned in an error message. When `type` is `NULL`, the first entry in the error message is used by default. See the Type section in the documentation below.
`by`	Aggregate unit-level estimates (aka, marginalize, average over). Valid inputs: • `FALSE`: return the original unit-level estimates. • `TRUE`: aggregate estimates for each term. • Character vector of column names in `newdata` or in the data frame produced by calling the function without the `by` argument.

- Data frame with a by column of group labels, and merging columns shared by `newdata` or the data frame produced by calling the same function without the by argument.
- See examples below.
- For more complex aggregations, you can use the `FUN` argument of the `hypotheses()` function. See that function's documentation and the Hypothesis Test vignettes on the `marginaleffects` website.

`bys` A character string (default `NULL`) specifying the variables over which the parameters are summarized. The summary statistics are computed for each unique combination of variables specified.

`conf_level` numeric value between 0 and 1. Confidence level to use to build a confidence interval.

`transform` string or function. Transformation applied to unit-level estimates and confidence intervals just before the function returns results. Functions must accept a vector and return a vector of the same length. Support string shortcuts: "exp", "ln"

`transform_draws`
A function applied to individual draws from the posterior distribution before computing summaries (default `NULL`). The argument `transform_draws` is derived from the `marginaleffects::predictions()` function and should not be confused with the `transform` argument from the deprecated `brms::posterior_predict()` function. It's important to note that for both `marginaleffects::predictions()` and `marginaleffects::avg_predictions()`, the `transform_draws` argument takes precedence over the `transform` argument. Note that when `transform_draws = NULL`, an attempt is made to automatically set `transform_draws = 'exp'` for `dpar = 'sigma'`. User can set `transform_draws = FALSE` to turn off this automatic assignment of `'exp'` to the `transform_draws`. It is also important to set `transform_draws = FALSE` when computing the first derivative (velocity) for `dpar = 'sigma'`.

`cross`
- `FALSE`: Contrasts represent the change in adjusted predictions when one predictor changes and all other variables are held constant.
- `TRUE`: Contrasts represent the changes in adjusted predictions when all the predictors specified in the `variables` argument are manipulated simultaneously (a "cross-contrast").

`wts` logical, string or numeric: weights to use when computing average predictions, contrasts or slopes. These weights only affect the averaging in `avg_*()` or with the by argument, and not unit-level estimates. See `?weighted.mean`
- string: column name of the weights variable in `newdata`. When supplying a column name to `wts`, it is recommended to supply the original data (including the weights variable) explicitly to `newdata`.
- numeric: vector of length equal to the number of rows in the original data or in `newdata` (if supplied).
- FALSE: Equal weights.
- TRUE: Extract weights from the fitted object with `insight::find_weights()` and use them when taking weighted averages of estimates. Warning: `newdata=datagrid()` returns a single average weight, which is equivalent to using `wts=FALSE`

`hypothesis` specify a hypothesis test or custom contrast using a number , formula, string equation, vector, matrix, or function.

- Number: The null hypothesis used in the computation of Z and p (before applying `transform`).
- String: Equation to specify linear or non-linear hypothesis tests. Two-tailed tests must include an equal = sign. One-tailed tests must start with < or >. If the terms in `coef(object)` uniquely identify estimates, they can be used in the formula. Otherwise, use `b1`, `b2`, etc. to identify the position of each parameter. The `b*` wildcard can be used to test hypotheses on all estimates. When the hypothesis string represents a two-sided equation, the `estimate` column holds the value of the left side minus the right side of the equation. If a named vector is used, the names are used as labels in the output. Examples:
  - `hp = drat`
  - `hp + drat = 12`
  - `b1 + b2 + b3 = 0`
  - `b* / b1 = 1`
  - `<= 0`
  - `>= -3.5`
  - `b1 >= 10`
- Formula: `lhs ~ rhs | group`
  - `lhs`
    * `ratio` (null = 1)
    * `difference` (null = 0)
    * Leave empty for default value
  - `rhs`
    * `pairwise` and `revpairwise`: pairwise differences between estimates in each row.
    * `reference`: differences between the estimates in each row and the estimate in the first row.
    * `sequential`: difference between an estimate and the estimate in the next row.
    * `meandev`: difference between an estimate and the mean of all estimates.
    * `meanotherdev`: difference between an estimate and the mean of all other estimates, excluding the current one.
    * `poly`: polynomial contrasts, as computed by the `stats::contr.poly()` function.
    * `helmert`: Helmert contrasts, as computed by the `stats::contr.helmert()` function. Contrast 2nd level to the first, 3rd to the average of the first two, and so on.
    * `trt_vs_ctrl`: difference between the mean of estimates (except the first) and the first estimate.
    * `I(fun(x))`: custom function to manipulate the vector of estimates `x`. The function `fun()` can return multiple (potentially named) estimates.

- `group` (optional)
  - Column name of `newdata`. Conduct hypothesis tests withing subsets of the data.
- Examples:
  - ~ `poly`
  - ~ `sequential` | `groupid`
  - ~ `reference`
  - `ratio` ~ `pairwise`
  - `difference` ~ `pairwise` | `groupid`
  - ~ `I(x - mean(x))` | `groupid`
  - `~ I(\(x) c(a = x[1], b = mean(x[2:3]))) | groupid`

- Matrix or Vector: Each column is a vector of weights. The the output is the dot product between these vectors of weights and the vector of estimates. The matrix can have column names to label the estimates.
- Function:
  - Accepts an argument `x`: object produced by a `marginaleffects` function or a data frame with column `rowid` and `estimate`
  - Returns a data frame with columns `term` and `estimate` (mandatory) and `rowid` (optional).
  - The function can also accept optional input arguments: `newdata`, `by`, `draws`.
  - This function approach will not work for Bayesian models or with bootstrapping. In those cases, it is easy to use `get_draws()` to extract and manipulate the draws directly.
- See the Examples section below and the vignette: https://marginaleffects.com/chapters/hypothesis.html
- Warning: When calling `predictions()` with `type="invlink(link)"` (the default in some models), `hypothesis` is tested and p values are computed on the link scale.

`equivalence` Numeric vector of length 2: bounds used for the two-one-sided test (TOST) of equivalence, and for the non-inferiority and non-superiority tests. For bayesian models, this report the proportion of posterior draws in the interval and the ROPE. See Details section below.

`eps` NULL or numeric value which determines the step size to use when calculating numerical derivatives: (f(x+eps)-f(x))/eps. When `eps` is `NULL`, the step size is 0.0001 multiplied by the difference between the maximum and minimum values of the variable with respect to which we are taking the derivative. Changing `eps` may be necessary to avoid numerical problems in certain models.

`constrats_by` A character vector (default `FALSE`) specifying the variable(s) by which estimates and contrasts (during the post-draw stage) should be computed using the `hypothesis` argument. The variable(s) in `constrats_by` should be a subset of those specified in the `by` argument. If `constrats_by = NULL`, it will copy all variables from `by`, except for the level-1 predictor (e.g., `age`). To disable this automatic behavior, use `constrats_by = FALSE`. This argument is evaluated only when `method = 'custom'` and `hypothesis` is not `NULL`.

`constrats_at`	A named list (default `FALSE`) to specify the values at which estimates and contrasts should be computed during the post-draw stage using the `hypothesis` argument. The values can be specified as `'max'`, `'min'`, `'unique'`, or `'range'` (e.g., `constrats_at = list(age = 'min')`) or as numeric vectors (e.g., `constrats_at = list(age = c(6, 7))`). If `constrats_at = NULL`, level-1 predictors (e.g., age) are automatically set to their unique values (i.e., `constrats_at = list(age = 'unique')`). To turn off this behavior, use `constrats_at = FALSE`. Note that `constrats_at` only affects data subsets prepared via `marginaleffects::datagrid()` or the `newdata` argument. The argument is evaluated only when `method = 'custom'` and `hypothesis` is not `NULL`.
`constrats_subset`	A named list (default `FALSE`) to subset draws for which contrasts should be computed via the `hypothesis` argument. This is similar to `constrats_at`, except that `constrats_subset` filters draws based on the character vector of variable names (e.g., `constrats_subset = list(id = c('id1', 'id2'))`) rather than numeric values. The argument is evaluated only when `method = 'custom'` and `hypothesis` is not `NULL`.
`reformat`	A logical (default `TRUE`) indicating whether to reformat the output returned by `marginaleffects` as a data frame. Column names are redefined as `conf.low` to `Q2.5` and `conf.high` to `Q97.5` (assuming `conf_int = 0.95`). Additionally, some columns (`term`, `contrast`, etc.) are dropped from the data frame.
`estimate_center`	A character string (default `NULL`) specifying how to center estimates: either `'mean'` or `'median'`. This option sets the global options as follows: `options("marginaleffects_poste` `= "mean")` or `options("marginaleffects_posterior_center" = "median")`. These global options are restored upon function exit using `base::on.exit()`.
`estimate_interval`	A character string (default `NULL`) to specify the type of credible intervals: `'eti'` for equal-tailed intervals or `'hdi'` for highest density intervals. This option sets the global options as follows: `options("marginaleffects_posterior_interval" = "eti")` or `options("marginaleffects_posterior_interval" = "hdi")`, and is restored on exit using `base::on.exit()`.
`dummy_to_factor`	A named list (default `NULL`) to convert dummy variables into a factor variable. The list must include the following elements: • `factor.dummy`: A character vector of dummy variables to be converted to factors. • `factor.name`: The name for the newly created factor variable (default is `'factor.var'` if `NULL`). • `factor.level`: A vector specifying the factor levels. If `NULL`, levels are taken from `factor.dummy`. If `factor.level` is provided, its length must match `factor.dummy`.
`verbose`	A logical argument (default `FALSE`) to specify whether to print information collected during the setup of the object(s).
`expose_function`	A logical value or `NULL` (default `FALSE`). Controls whether Stan functions are exposed for post-processing.

- `TRUE`: Explicitly exposes Stan functions saved from the model fit. This is required when calculating `fit criteria` or `bayes_R2` during model optimization.
- `FALSE` (default): Stan functions are not exposed.
- `NULL`: The setting is inherited from the original `bsitar()` model fit configuration.

Note: In the `optimize_model()` function, the default is `NULL` (inheriting behavior), whereas other post-processing functions default to `FALSE`.

`usesavedfuns`	A logical value (default `NULL`) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the `expose_functions` argument from the `bsitar()` call. Manual specification of `usesavedfuns` is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.
`clearenvfuns`	A logical value indicating whether to clear the exposed Stan functions from the environment (`TRUE`) or not (`FALSE`). If `NULL`, `clearenvfuns` is set based on the value of `usesavedfuns`: `TRUE` if `usesavedfuns = TRUE`, or `FALSE` if `usesavedfuns = FALSE`.
`funlist`	A list (default `NULL`) specifying function names. This is rarely needed, as required functions are typically retrieved automatically. A use case for `funlist` is when `sigma_formula`, `sigma_formula_gr`, or `sigma_formula_gr_str` use an external function (e.g., `poly(age)`). The `funlist` should include function names defined in the `globalenv()`. For functions needing both distance and velocity curves (e.g., `plot_curves(..., opt = 'dv')`), `funlist` must include two functions: one for the distance curve and one for the velocity curve.
`xvar`	A character string (default `NULL`) specifying the `'x'` variable. Rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.
`idvar`	A character string (default `NULL`) specifying the `'id'` variable. Rarely used because `idvar` is inferred internally.
`difx`	A character string (default `NULL`) specifying the `'x'` variable that should be used for manual differentiation of the distance curve. Internally, the `xvar` is set as `difx` if specified. The argument `difx` is evaluated only when `dpar = 'sigma'`, ignored otherwise. Note that argument `xvar` itself is rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.
`itransform`	A character string (default `NULL`) indicating the variables names that are reverse transformed. Options are `c("x", "y", "sigma")`. The `itransform` is primarily used to get the `xvar` variable at original scale i.e., `itransform = 'x'`. To turn of all transformations, use `itransform = ""`. when `itransform = NULL`, the appropriate transformation for `xvar` is selected automatically. Note that when no match for `xvar` is found in the `data,frame`, the `itransform` will be ignored within the calling function, `'prepare_transformations()'`.
`incl_autocor`	A flag indicating if correlation structures originally specified via `autocor` should be included in the predictions. Defaults to `TRUE`.

`parameter_method`	An integer (default = `1`) that specifies the method used to compute individual-level growth parameters from fitted spline models. Two methods are available: `1` **(Model-based differentiation)** This method estimates growth parameters by directly leveraging the structure of the fitted spline model. The spline curve is segmented into pieces of cubic polynomials. Each segment is then analytically differentiated to obtain first and second derivatives, which represent the growth velocity and acceleration, respectively. This approach is more faithful to the underlying model and is recommended when the goal is to derive precise, model-consistent growth characteristics. `2` **(Plug-in adjustment using random effects)** This method takes a two-step approach. First, it calculates the population-level average estimate of the age or time point of interest. Then, it adjusts this estimate for individual subjects by incorporating random effects from the model. This approach is computationally simpler and may be preferable when derivative estimation from the spline model is not feasible or when interpretation in terms of deviations from the population norm is of primary interest.
`subset_by`	A logical or character string (default = `NULL`) that determines how to subset the data to retain a single unique row per `id`. This parameter is only used when `parameter_method == 2` and is ignored if `add_xtm = TRUE`. When `subset_by = NULL`, the function automatically sets `subset_by` to the value of the `by` argument (i.e., `subset_by = by`). To override this default behavior and skip subsetting altogether, set `subset_by = ""` (an empty string). This option is useful when multiple rows per `id` are present and a reduction to one representative row per individual is needed for downstream calculations. For `parameter_method == 2` with `re_formula == NA`, the `subset_by = "one-row"` will provide one row per parameter.
`add_xtm`	A logical (default `FALSE`) to indicate whether to compute `x` and `y` adjusted to the mean. Ignored if `parameter_method == 1`. Note that `add_xtm` does not affect the estimation of parameters.
`call_function`	A character string indicating the source of the function used for computation—either a native `R` implementation or a compiled function exposed from `Stan`. Valid options are `"R"` and `"Stan"`. The `call_function` is ignored when `parameter_method == 2`. Though `"Stan"` is much faster than `R`, The native `R` implementation (`call_function = "R"`, default) is recommended when the number of posterior draws is small. This is because the `Stan` function needs compilation which takes time. The future plan is to allow integration with `Stan`-based computational back ends which then be exposed along with other functions from `Stan` function block.
`newdata_fixed`	An indicator to specify whether to check data format and structure for the user provided `newdata`, and apply needed `prepare_data2` and `prepare_transformations` (`newdata_fixed = NULL`, default), return user provided `newdata` (`newdata = TRUE`) as it is without checking for the data format or applying `prepare_data2` and `prepare_transformations` (`newdata_fixed = 0`), check for the data format and if needed, prepare data format using `prepare_data2` (`newdata_fixed = 1`), or apply `prepare_transformations` only assuming that data format is correct (`newdata_fixed = 2`). It is strongly recommended that user either leave

the `newdata = NULL` and `newdata_fixed = NULL` in which case data used in the model fitting is automatically retrieved and checked for the required data format and transformations, and if needed, `prepare_data2` and `prepare_transformations` are applied internally. The other flags provided for `newdata_fixed = 0, 1, 2` are mainly for the internal use during post-processing.

`envir`	The environment used for function evaluation. The default is `NULL`, which sets the environment to `parent.frame()`. Since most post-processing functions rely on **brms**, it is recommended to set `envir = globalenv()` or `envir = .GlobalEnv`, especially for derivatives like velocity curves.
`...`	Additional arguments passed to the function.

Details

Since SITAR is a shape-invariant model, each individual curve has a peak velocity point that can be mapped by knowing the population average age at peak velocity. This hold true even when a individual lacks measurements at the expected turning point.

Value

A data frame comprising growth parameter estimates for **age**, **distance** and **velocity**.

Author(s)

Satpal Sandhu <`satpal.sandhu@bristol.ac.uk`>

See Also

`marginaleffects::predictions()`

Examples

```
# Fit Bayesian SITAR model

# To avoid mode estimation which takes time, the Bayesian SITAR model fit to
# the 'berkeley_exdata' has been saved as an example fit ('berkeley_exfit').
# See 'bsitar' function for details on 'berkeley_exdata' and 'berkeley_exfit'.

# Check and confirm whether model fit object 'berkeley_exfit' exists
berkeley_exfit <- getNsObject(berkeley_exfit)

model <- berkeley_exfit

modelbased_growthparameters(model, ndraws = 2, parameter = 'apgv')
```

optimize_model.bgmfit *Optimize Bayesian SITAR Model*

Description

The optimization process for selecting the best-fitting SITAR model involves choosing the optimal degrees of freedom (df) for the natural cubic spline curve, as well as determining the appropriate transformations for the predictor (x) and/or outcome (y) variables.

Usage

```
## S3 method for class 'bgmfit'
optimize_model(
  model,
  newdata = NULL,
  optimize_df = NULL,
  optimize_x = list(NULL, log, sqrt),
  optimize_y = list(NULL, log, sqrt),
  transform_prior_class = NULL,
  transform_beta_coef = NULL,
  transform_sd_coef = NULL,
  exclude_default = TRUE,
  add_fit_criteria = c("loo"),
  byresp = FALSE,
  model_name = NULL,
  overwrite = FALSE,
  file = NULL,
  force_save = FALSE,
  save_each = FALSE,
  digits = 2,
  cores = 1,
  verbose = FALSE,
  expose_function = FALSE,
  usesavedfuns = FALSE,
  clearenvfuns = NULL,
  envir = NULL,
  ...
)

optimize_model(model, ...)
```

Arguments

model	An object of class bgmfit.
newdata	An optional data frame for estimation. If NULL (default), newdata is retrieved from the model.

`optimize_df` A list of integers specifying the degrees of freedom (`df`) values to be optimized. If `NULL` (default), the `df` is taken from the original model. To optimize over different `df` values, for example, `df` 4 and `df` 5, the corresponding code would be `optimize_df = list(4, 5)`. For `univariate_by` and `multivariate` models, `optimize_df` can be a single integer (e.g., `optimize_df = 4`), a list (e.g., `optimize_df = list(4, 5)`), or a list of lists. For instance, to optimize over `df` 4 and `df` 5 for the first submodel, and `df` 5 and `df` 6 for the second submodel, the corresponding code would be `optimize_df = list(list(4, 5), list(5, 6))`.

`optimize_x` A list specifying the transformations for the predictor variable (i.e., `x`). The available options are `NULL`, `'log'`, `'sqrt'`, or their combinations. Note that the user need not enclose these options in single or double quotes, as they are handled internally. The default setting explores all possible combinations, i.e., `optimize_x = list(NULL, 'log', 'sqrt')`. Similar to `optimize_df`, the user can specify different `optimize_x` values for `univariate_by` and `multivariate` submodels. Additionally, it is possible to pass any primitive function instead of fixed functions like `log` and `sqrt`. This greatly enhances the flexibility of model optimization by allowing the search for a wide range of `x` transformations, such as `optimize_x = list(function(x) log(x + 3/4))`.

`optimize_y` A list specifying the transformations of the response variable (i.e., `y`). The approach and available options for `optimize_y` are the same as described above for `optimize_x`.

`transform_prior_class`
A character vector (default `NULL`) specifying the parameter class for which transformations of user-specified priors should be performed. These options are `'beta'`, `'sd'`, `'rsd'`, `'sigma'`, and `'dpar'`, and they can be specified as follows:
`transform_prior_class = c('beta', 'sd', 'rsd', 'sigma', 'dpar')`. Note that transformations can only be applied to location-scale based priors such as `normal()`. Currently, transformation are supported only for the `'log'` transformed `'x'` variable. The `'log'` transformation of prior is performed as follows:
`log_location = log(location / sqrt(scale^2 / location^2 + 1))`,
`log_scale = sqrt(log(scale^2 / location^2 + 1))`,
where `location` and `scale` are the original parameters supplied by the user, and `log_location` and `log_scale` are the equivalent parameters on the log scale. Note that `transform_prior_class` is used on an experimental basis, and therefore the results may not be as intended. We recommend explicitly setting the desired prior for the `y` scale.

`transform_beta_coef`
A character vector (default `NULL`) specifying the regression coefficients for which transformations are applied. The coefficients that can be transformed are `'a'`, `'b'`, `'c'`, `'d'`, and `'s'`. The default is `transform_beta_coef = c('b', 'c', 'd')`, which implies that the parameters `'b'`, `'c'`, and `'d'` will be transformed, while parameter `'a'` will be left unchanged because the default prior for parameter `'a'` is based on the outcome `y` scale itself (e.g., `a_prior_beta = normal(ymean, ysd)`), which gets transformed automatically. Note that `transform_beta_coef` is ignored when `transform_prior_class = NULL`.

`transform_sd_coef`	A character vector (default `NULL`) specifying the `sd` parameters for which transformations are applied. The coefficients that can be transformed are `'a'`, `'b'`, `'c'`, `'d'`, and `'s'`. The default is `transform_sd_coef = c('b', 'c', 'd')`, which implies that the parameters `'b'`, `'c'`, and `'d'` will be transformed, while parameter `'a'` will be left unchanged because the default prior for parameter `'a'` is based on the outcome `y` scale itself (e.g., `a_prior_beta = normal(ymean, ysd)`), which gets transformed automatically. Note that `transform_sd_coef` is ignored when `transform_prior_class = NULL`.
`exclude_default`	A logical indicating whether transformations for (`x` and `y`) variables used in the original model fit should be excluded. If `TRUE` (default), the transformations specified for the `x` and `y` variables in the original model fit are excluded from `optimize_x` and `optimize_y`. For example, if the original model is fit with `xvar = log` and `yvar = NULL`, then `optimize_x` is translated into `optimize_x = list(NULL, sqrt)`, and `optimize_y` is reset as `optimize_y = list(log, sqrt)`.
`add_fit_criteria`	a
`byresp`	A logical (default `FALSE`) indicating whether response-wise fit criteria should be calculated. This argument is evaluated only for the `multivariate` model, where the user can select whether to get a joint calculation of point-wise log likelihood (`byresp = FALSE`) or response-specific calculations (`byresp = TRUE`). For the `univariate_by` model, the only available option is to calculate separate point-wise log likelihood for each submodel, i.e., `byresp = TRUE`.
`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`.
`save_each`	A logical (default `FALSE`) indicating whether to save each model (as a `.rds` file) when running the loop. Note that the user can also specify `save_each` as a named list to pass the following information when saving each model: `'prefix'` a character string (default `NULL`), `'suffix'` a character string (default `NULL`), `'extension'` a character string, either `.rds` or `.RData` (default `.rds`), `'compress'` a character string, either `'xz'`, `'gzip'`, or `'bzip2'` (default `'xz'`). These options are set as follows:

save_each = list(prefix = '', suffix = '', extension = 'rds', compress = 'xz').

digits	An integer (default 2) to set the decimal places for rounding the results using the base::round() function.
cores	The number of cores to use in parallel processing (default 1). The argument cores is passed to [brms::add_criterion()].
verbose	A logical argument (default FALSE) to specify whether to print information collected during the setup of the object(s).
expose_function	A logical value or NULL (default FALSE). Controls whether Stan functions are exposed for post-processing. • TRUE: Explicitly exposes Stan functions saved from the model fit. This is required when calculating fit criteria or bayes_R2 during model optimization. • FALSE (default): Stan functions are not exposed. • NULL: The setting is inherited from the original bsitar() model fit configuration. **Note**: In the optimize_model() function, the default is NULL (inheriting behavior), whereas other post-processing functions default to FALSE.
usesavedfuns	A logical value (default NULL) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the expose_functions argument from the bsitar() call. Manual specification of usesavedfuns is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.
clearenvfuns	A logical value indicating whether to clear the exposed Stan functions from the environment (TRUE) or not (FALSE). If NULL, clearenvfuns is set based on the value of usesavedfuns: TRUE if usesavedfuns = TRUE, or FALSE if usesavedfuns = FALSE.
envir	The environment used for function evaluation. The default is NULL, which sets the environment to parent.frame(). Since most post-processing functions rely on **brms**, it is recommended to set envir = globalenv() or envir = .GlobalEnv, especially for derivatives like velocity curves.
...	Other arguments passed to update_model.

Value

A list containing the optimized models of class bgmfit, and the the summary statistics if add_fit_criteria are specified.

Author(s)

Satpal Sandhu <satpal.sandhu@bristol.ac.uk>

See Also

brms::add_criterion()

Examples

```
# Fit Bayesian SITAR model

# To avoid model estimation, which takes time, the Bayesian SITAR model fit
# to the 'berkeley_exdata' has been saved as an example fit ('berkeley_exfit').
# See 'bsitar' function for details on 'berkeley_exdata' and 'berkeley_exfit'.

# Check and confirm whether the model fit object 'berkeley_exfit' exists
 berkeley_exfit <- getNsObject(berkeley_exfit)

model <- berkeley_exfit

# The following example shows a dummy call for optimization to save time.
# Note that if the degree of freedom, and both the \code{optimize_x} and
# \code{optimize_y} are \code{NULL} (i.e., nothing to optimize), the original
# model object is returned.
# To explicitly check whether the model is being optimized or not,
# the user can set \code{verbose = TRUE}. This is useful for getting
# information about what arguments have changed compared to the
# original model.

model2 <- optimize_model(model,
  optimize_df = NULL,
  optimize_x = NULL,
  optimize_y = NULL,
  verbose = TRUE)
```

plot_conditional_effects.bgmfit

Visualize conditional effects for the Bayesian SITAR model

Description

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

Usage

```
## S3 method for class 'bgmfit'
plot_conditional_effects(
  model,
  effects = NULL,
  conditions = NULL,
  int_conditions = NULL,
  re_formula = NA,
  spaghetti = FALSE,
  surface = FALSE,
  categorical = FALSE,
  ordinal = FALSE,
  method = NULL,
  allow_new_levels = FALSE,
  estimation_method = "fitted",
  transform = NULL,
  transform_draws = NULL,
  resolution = 100,
  select_points = 0,
  too_far = 0,
  probs = c(0.025, 0.975),
  robust = TRUE,
  newdata = NULL,
  ndraws = NULL,
  dpar = NULL,
  draw_ids = NULL,
  levels_id = NULL,
  resp = NULL,
  ipts = NULL,
  deriv = 0,
  summary = FALSE,
  model_deriv = NULL,
  idata_method = NULL,
  verbose = FALSE,
  label.x = NULL,
  label.y = NULL,
  label.title = NULL,
  label.subtitle = NULL,
  legendpos = NULL,
  dummy_to_factor = NULL,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  funlist = NULL,
  xvar = NULL,
  difx = NULL,
  idvar = NULL,
  itransform = NULL,
  newdata_fixed = NULL,
  envir = NULL,
  ...
)

plot_conditional_effects(model, ...)
```

Arguments

`model`	An object of class `bgmfit`.

`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.
`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`.
`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.
`allow_new_levels`	A flag indicating if new levels of group-level effects are allowed (defaults to `FALSE`). Only relevant if `newdata` is provided.

`estimation_method`	A character string specifying the estimation method when calculating the velocity from the posterior draws. The `'fitted'` method internally calls `fitted_draws()`, while the `'predict'` method calls `predict_draws()`. See `brms::fitted.brmsfit()` and `brms::predict.brmsfit()` for details.
`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"`.
`transform_draws`	A function applied to individual draws from the posterior distribution before computing summaries (default `NULL`). The argument `transform_draws` is derived from the `marginaleffects::predictions()` function and should not be confused with the `transform` argument from the deprecated `brms::posterior_predict()` function. It's important to note that for both `marginaleffects::predictions()` and `marginaleffects::avg_predictions()`, the `transform_draws` argument takes precedence over the `transform` argument. Note that when `transform_draws = NULL`, an attempt is made to automatically set `transform_draws = 'exp'` for `dpar = 'sigma'`. User can set `transform_draws = FALSE` to turn off this automatic assignment of `'exp'` to the `transform_draws`. It is also important to set `transform_draws = FALSE` when computing the first derivative (velocity) for `dpar = 'sigma'`.
`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.
`robust`	If `TRUE` (the default) the median is used as the measure of central tendency. If `FALSE` the mean is used instead.
`newdata`	An optional data frame for estimation. If `NULL` (default), `newdata` is retrieved from the `model`.
`ndraws`	A positive integer indicating the number of posterior draws to use in estimation. If `NULL` (default), all draws are used.
`dpar`	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.

`draw_ids`	An integer specifying the specific posterior draw(s) to use in estimation (default `NULL`).
`levels_id`	An optional argument to specify the `ids` for the hierarchical model (default `NULL`). It is used only when the model is applied to data with three or more levels of hierarchy. For a two-level model, `levels_id` is automatically inferred from the model fit. For models with three or more levels, `levels_id` is inferred from the model fit under the assumption that hierarchy is specified from the lowest to the uppermost level, i.e., `id` followed by `study`, where `id` is nested within `study`. However, it is not guaranteed that `levels_id` is sorted correctly, so it is better to set it manually when fitting a model with three or more levels of hierarchy.
`resp`	A character string (default `NULL`) to specify the response variable when processing posterior draws for `univariate_by` and `multivariate` models. See `bsitar()` for details on `univariate_by` and `multivariate` models.
`ipts`	An integer specifying the number of points for interpolating the predictor variable (e.g., age) to generate smooth curves for predictions and plots. This value is used as the `length.out` argument for `seq()`, controlling the smoothness of distance and velocity curves without altering the predictor range. `NULL` **(the default)** Engages automatic behavior based on the `dpar` argument: it is internally set to `50` for the mean response (`dpar = 'mu'`), ensuring smooth curves, while for the distributional parameters (e.g., `dpar = 'sigma'`), no interpolation is performed. **An integer (e.g.,** `100`**)** Explicitly sets the number of interpolation points. `FALSE` Disables all interpolation, forcing predictions to be made only at the original data points of the predictor variable. This argument affects the following post-processing functions: `fitted_draws()`, `predict_draws()`, `growthparameters()`, `plot_curves()`, `get_predictions()`, `get_comparisons()`, and `get_growthparameters()`.
`deriv`	An integer indicating whether to estimate the distance curve or its derivative (velocity curve). The default `deriv = 0` is for the distance curve, while `deriv = 1` is for the velocity curve.
`summary`	A logical value indicating whether only the estimate should be computed (`TRUE`), or whether the estimate along with SE and CI should be returned (`FALSE`, default). Setting `summary` to `FALSE` will increase computation time. Note that `summary = FALSE` is required to obtain correct estimates when `re_formula = NULL`.
`model_deriv`	A logical value specifying whether to estimate the velocity curve from the derivative function or by differentiating the distance curve. Set `model_deriv = TRUE` for functions that require the velocity curve, such as `growthparameters()` and `plot_curves()`. Set it to `NULL` for functions that use the distance curve (i.e., fitted values), such as `loo_validation()` and `plot_ppc()`.
`idata_method`	A character string specifying the interpolation method (default `NULL`). The number of interpolation points is controlled by the `ipts` argument. Available options:

- `'m1'`: Adapted from the **iapvbs** package (documented here). This method internally constructs the data frame based on the model configuration. *Note:* This method may fail if the model includes covariates (especially in `univariate_by` models).
- `'m2'`: Based on the **JMbayes** package (documented here). This method uses the exact data frame from the model fit (`fit$data`) and is generally more robust.

If `idata_method = NULL`, `'m2'` is automatically selected. It is recommended to use `'m2'` if `'m1'` encounters errors with covariate-dependent models.

`verbose` A logical argument (default `FALSE`) to specify whether to print information collected during the setup of the object(s).

`label.x` An optional character string to label the x-axis. If `NULL` (default), the x-axis label will be taken from the predictor (e.g., age).

`label.y` An optional character string to label the y-axis. If `NULL` (default), the y-axis label will be taken from the plot type (e.g., distance, velocity). When `layout = 'facet'`, the label is removed, and the same label is used as the title.

`label.title` An optional character string to label the title. Default `NULL`.

`label.subtitle` An optional character string to label the title. Default `NULL`

`legendpos` A character string to specify the position of the legend. If `NULL` (default), the legend position is set to 'bottom' for distance and velocity curves in the `'single'` layout. For individual-specific curves, the legend position is set to `'none'` to suppress the legend.

`dummy_to_factor`

A named list (default `NULL`) to convert dummy variables into a factor variable. The list must include the following elements:

- `factor.dummy`: A character vector of dummy variables to be converted to factors.
- `factor.name`: The name for the newly created factor variable (default is `'factor.var'` if `NULL`).
- `factor.level`: A vector specifying the factor levels. If `NULL`, levels are taken from `factor.dummy`. If `factor.level` is provided, its length must match `factor.dummy`.

`expose_function`

A logical value or `NULL` (default `FALSE`). Controls whether Stan functions are exposed for post-processing.

- `TRUE`: Explicitly exposes Stan functions saved from the model fit. This is required when calculating `fit criteria` or `bayes_R2` during model optimization.
- `FALSE` (default): Stan functions are not exposed.
- `NULL`: The setting is inherited from the original `bsitar()` model fit configuration.

Note: In the `optimize_model()` function, the default is `NULL` (inheriting behavior), whereas other post-processing functions default to `FALSE`.

`usesavedfuns`	A logical value (default `NULL`) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the `expose_functions` argument from the `bsitar()` call. Manual specification of `usesavedfuns` is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.
`clearenvfuns`	A logical value indicating whether to clear the exposed Stan functions from the environment (`TRUE`) or not (`FALSE`). If `NULL`, `clearenvfuns` is set based on the value of `usesavedfuns`: `TRUE` if `usesavedfuns = TRUE`, or `FALSE` if `usesavedfuns = FALSE`.
`funlist`	A list (default `NULL`) specifying function names. This is rarely needed, as required functions are typically retrieved automatically. A use case for `funlist` is when `sigma_formula`, `sigma_formula_gr`, or `sigma_formula_gr_str` use an external function (e.g., `poly(age)`). The `funlist` should include function names defined in the `globalenv()`. For functions needing both distance and velocity curves (e.g., `plot_curves(..., opt = 'dv')`), `funlist` must include two functions: one for the distance curve and one for the velocity curve.
`xvar`	A character string (default `NULL`) specifying the `'x'` variable. Rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.
`difx`	A character string (default `NULL`) specifying the `'x'` variable that should be used for manual differentiation of the distance curve. Internally, the `xvar` is set as `difx` if specified. The argument `difx` is evaluated only when `dpar = 'sigma'`, ignored otherwise. Note that argument `xvar` itself is rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.
`idvar`	A character string (default `NULL`) specifying the `'id'` variable. Rarely used because `idvar` is inferred internally.
`itransform`	A character string (default `NULL`) indicating the variables names that are reverse transformed. Options are `c("x", "y", "sigma")`. The `itransform` is primarily used to get the `xvar` variable at original scale i.e., `itransform = 'x'`. To turn of all transformations, use `itransform = ""`. when `itransform = NULL`, the appropriate transformation for `xvar` is selected automatically. Note that when no match for `xvar` is found in the `data,frame`, the `itransform` will be ignored within the calling function, `'prepare_transformations()'`.
`newdata_fixed`	An indicator to specify whether to check data format and structure for the user provided `newdata`, and apply needed `prepare_data2` and `prepare_transformations` (`newdata_fixed = NULL`, default), return user provided `newdata` (`newdata = TRUE`) as it is without checking for the data format or applying `prepare_data2` and `prepare_transformations` (`newdata_fixed = 0`), check for the data format and if needed, prepare data format using `prepare_data2` (`newdata_fixed = 1`), or apply `prepare_transformations` only assuming that data format is correct (`newdata_fixed = 2`). It is strongly recommended that user either leave the `newdata = NULL` and `newdata_fixed = NULL` in which case data used in the model fitting is automatically retrieved and checked for the required data format and transformations, and if needed, `prepare_data2` and `prepare_transformations` are applied internally. The other flags provided for `newdata_fixed = 0, 1, 2` are mainly for the internal use during post-processing.

`envir`	The environment used for function evaluation. The default is `NULL`, which sets the environment to `parent.frame()`. Since most post-processing functions rely on **brms**, it is recommended to set `envir = globalenv()` or `envir = .GlobalEnv`, especially for derivatives like velocity curves.
`...`	Additional arguments passed to the `brms::conditional_effects()` function. Please see `brms::conditional_effects()` for details.

Details

The **plot_conditional_effects()** is a wrapper around the `brms::conditional_effects()`. The `brms::conditional_effects()` function from the **brms** package can be used to plot the fitted (distance) curve when response (e.g., height) is not transformed. However, when the outcome is log or square root transformed, the `brms::conditional_effects()` will return the fitted curve on the log or square root scale, whereas the **plot_conditional_effects()** will return the fitted curve on the original scale. Furthermore, the **plot_conditional_effects()** also plots the velocity curve on the original scale after making the required back-transformation. Apart from these differences, both these functions (brms::conditional_effects and **plot_conditional_effects()**) work in the same manner. In other words, the user can specify all the arguments which are available in the `brms::conditional_effects()`. An alternative approach is to `get_predictions()` function (with `plot = TRUE`) which is based on the **marginaleffects**.

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. See `brms::conditional_effects()` for details.

Author(s)

Satpal Sandhu <`satpal.sandhu@bristol.ac.uk`>

See Also

`brms::conditional_effects()`

Examples

```
# Fit Bayesian SITAR model

# To avoid mode estimation which takes time, the Bayesian SITAR model fit to
# the 'berkeley_exdata' has been saved as an example fit ('berkeley_exfit').
# See 'bsitar' function for details on 'berkeley_exdata' and 'berkeley_exfit'.

# Check and confirm whether model fit object 'berkeley_exfit' exists
 berkeley_exfit <- getNsObject(berkeley_exfit)

model <- berkeley_exfit

# Population average distance curve
plot_conditional_effects(model, deriv = 0, re_formula = NA)
```

```
# Individual-specific distance curves
plot_conditional_effects(model, deriv = 0, re_formula = NULL)

# Population average velocity curve
plot_conditional_effects(model, deriv = 1, re_formula = NA)

# Individual-specific velocity curves
plot_conditional_effects(model, deriv = 1, re_formula = NULL)
```

---

plot_curves.bgmfit *Plot growth curves for the Bayesian SITAR model*

Description

The **plot_curves()** function visualizes six different types of growth curves using the **ggplot2** package. Additionally, it allows users to create customized plots from the data returned as a `data.frame`. For an alternative approach, the `get_predictions()` function can be used, which not only estimates adjusted curves but also enables comparison across groups using the `hypotheses` argument.

Usage

```
## S3 method for class 'bgmfit'
plot_curves(
  model,
  opt = "dv",
  apv = FALSE,
  bands = NULL,
  conf = 0.95,
  resp = NULL,
  dpar = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  newdata = NULL,
  summary = FALSE,
  digits = 2,
  re_formula = NULL,
  numeric_cov_at = NULL,
  aux_variables = NULL,
  grid_add = NULL,
  levels_id = NULL,
  avg_reffects = NULL,
  ipts = NULL,
  model_deriv = TRUE,
  xrange = NULL,
  xrange_search = NULL,
  takeoff = FALSE,
  trough = FALSE,
  acgv = FALSE,
  acgv_velocity = 0.1,
  seed = 123,
  estimation_method = "fitted",
  allow_new_levels = FALSE,
  sample_new_levels = "uncertainty",
  incl_autocor = TRUE,
  robust = FALSE,
  transform_draws = NULL,
  scale = c("response", "linear"),
  future = FALSE,
  future_session = "multisession",
  cores = NULL,
  trim = 0,
  layout = "single",
  linecolor = NULL,
  linecolor1 = NULL,
  linecolor2 = NULL,
  label.x = NULL,
  label.y = NULL,
  label.title = NULL,
  label.subtitle = NULL,
  legendpos = NULL,
  linetype.apv = NULL,
  linewidth.main = NULL,
  linewidth.apv = NULL,
  linetype.groupby = NA,
  color.groupby = NA,
  band.alpha = NULL,
  show_age_takeoff = TRUE,
  show_age_peak = TRUE,
  show_age_cessation = TRUE,
  show_vel_takeoff = FALSE,
  show_vel_peak = FALSE,
  show_vel_cessation = FALSE,
  returndata = FALSE,
  returndata_add_parms = FALSE,
  parms_eval = FALSE,
  idata_method = NULL,
  parms_method = "getPeak",
  verbose = FALSE,
  fullframe = NULL,
  dummy_to_factor = NULL,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  funlist = NULL,
  xvar = NULL,
  difx = NULL,
  idvar = NULL,
  itransform = NULL,
  newdata_fixed = NULL,
  envir = NULL,
  ...
)

plot_curves(model, ...)
```

Arguments

`model`	An object of class `bgmfit`.
`opt`	A character string containing one or more of the following plotting options: • 'd': Population average distance curve • 'v': Population average velocity curve • 'D': Individual-specific distance curves • 'V': Individual-specific velocity curves • 'u': Unadjusted individual-specific distance curves • 'a': Adjusted individual-specific distance curves (adjusted for random effects) Note that 'd' and 'D' cannot be specified simultaneously, nor can 'v' and 'V'. Other combinations are allowed, e.g., 'dvau', 'Dvau', 'dVau', etc.
`apv`	A logical value (default `FALSE`) indicating whether to calculate and plot the age at peak velocity (APGV) when `opt` includes 'v' or 'V'.
`bands`	A character string containing one or more of the following options, or `NULL` (default), indicating if CI bands should be plotted around the curves: • 'd': Band around the distance curve • 'v': Band around the velocity curve • 'p': Band around the vertical line denoting the APGV parameter The `'dvp'` option will include CI bands for distance and velocity curves, and the APGV.
`conf`	A numeric value (default `0.95`) specifying the confidence interval (CI) level for the bands. See `growthparameters()` for more details.
`resp`	A character string (default `NULL`) to specify the response variable when processing posterior draws for `univariate_by` and `multivariate` models. See `bsitar()` for details on `univariate_by` and `multivariate` models.
`dpar`	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
`ndraws`	A positive integer indicating the number of posterior draws to use in estimation. If `NULL` (default), all draws are used.
`draw_ids`	An integer specifying the specific posterior draw(s) to use in estimation (default `NULL`).

`newdata`	An optional data frame for estimation. If `NULL` (default), `newdata` is retrieved from the `model`.
`summary`	A logical value indicating whether only the estimate should be computed (`TRUE`), or whether the estimate along with SE and CI should be returned (`FALSE`, default). Setting `summary` to `FALSE` will increase computation time. Note that `summary = FALSE` is required to obtain correct estimates when `re_formula = NULL`.
`digits`	An integer (default 2) to set the decimal places for rounding the results using the `base::round()` function.
`re_formula`	Option to indicate whether or not to include individual/group-level effects in the estimation. When `NA` (default), individual-level effects are excluded, and population average growth parameters are computed. When `NULL`, individual-level effects are included in the computation, and the resulting growth parameters are individual-specific. In both cases (`NA` or `NULL`), continuous and factor covariates are appropriately included in the estimation. Continuous covariates are set to their means by default (see `numeric_cov_at` for details), while factor covariates remain unaltered, allowing for the estimation of covariate-specific population average and individual-specific growth parameters.
`numeric_cov_at`	An optional (named list) argument to specify the value of continuous covariate(s). The default `NULL` option sets the continuous covariate(s) to their mean. Alternatively, a named list can be supplied to manually set these values. For example, `numeric_cov_at = list(xx = 2)` will set the continuous covariate variable 'xx' to 2. The argument `numeric_cov_at` is ignored when no continuous covariates are included in the model.
`aux_variables`	An optional argument to specify variables passed to the `ipts` argument, useful when fitting location-scale or measurement error models.
`grid_add`	An optional argument to specify the variable(s) that can be passed to the `marginaleffects::datagrid()` This is useful when fitting location-scale models and measurement error models. If post-processing functions throw an error such as `variable 'x' not found in either 'data' or 'data2'`, consider using `grid_add`. Note that unlike `aux_variables` which are passed to the internal data functions such as `'get.newdata'`, the `grid_add` are passed to the `marginaleffects::datagrid()`.
`levels_id`	An optional argument to specify the `ids` for the hierarchical model (default `NULL`). It is used only when the model is applied to data with three or more levels of hierarchy. For a two-level model, `levels_id` is automatically inferred from the model fit. For models with three or more levels, `levels_id` is inferred from the model fit under the assumption that hierarchy is specified from the lowest to the uppermost level, i.e., `id` followed by `study`, where `id` is nested within `study`. However, it is not guaranteed that `levels_id` is sorted correctly, so it is better to set it manually when fitting a model with three or more levels of hierarchy.
`avg_reffects`	An optional argument (default `NULL`) to calculate (marginal/average) curves and growth parameters, such as APGV and PGV. If specified, it must be a named list indicating the `over` (typically a level 1 predictor, such as age), `feby` (fixed effects, typically a factor variable), and `reby` (typically `NULL`, indicating that parameters are integrated over the random effects). For example, `avg_reffects = list(feby = 'study', reby = NULL, over = 'age')`.

`ipts`	An integer specifying the number of points for interpolating the predictor variable (e.g., age) to generate smooth curves for predictions and plots. This value is used as the `length.out` argument for `seq()`, controlling the smoothness of distance and velocity curves without altering the predictor range. `NULL` **(the default)** Engages automatic behavior based on the `dpar` argument: it is internally set to `50` for the mean response (`dpar = 'mu'`), ensuring smooth curves, while for the distributional parameters (e.g., `dpar = 'sigma'`), no interpolation is performed. **An integer (e.g.,** `100`**)** Explicitly sets the number of interpolation points. `FALSE` Disables all interpolation, forcing predictions to be made only at the original data points of the predictor variable. This argument affects the following post-processing functions: `fitted_draws()`, `predict_draws()`, `growthparameters()`, `plot_curves()`, `get_predictions()`, `get_comparisons()`, and `get_growthparameters()`.
`model_deriv`	A logical value specifying whether to estimate the velocity curve from the derivative function or by differentiating the distance curve. Set `model_deriv = TRUE` for functions that require the velocity curve, such as `growthparameters()` and `plot_curves()`. Set it to `NULL` for functions that use the distance curve (i.e., fitted values), such as `loo_validation()` and `plot_ppc()`.
`xrange`	An integer to set the predictor range (e.g., age) when executing the interpolation via `ipts`. By default, `NULL` sets the individual-specific predictor range. Setting `xrange = 1` applies the same range for individuals within the same higher grouping variable (e.g., study). Setting `xrange = 2` applies an identical range across the entire sample. Alternatively, a numeric vector (e.g., `xrange = c(6, 20)`) can be provided to set the range within the specified values.
`xrange_search`	A vector of length two or a character string `'range'` to set the range of the predictor variable (`x`) within which growth parameters are searched. This is useful when there is more than one peak and the user wants to summarize the peak within a specified range of the `x` variable. The default value is `xrange_search = NULL`.
`takeoff`	A logical value (default `FALSE`) indicating whether to calculate the age at takeoff velocity (ATGV) and the takeoff growth velocity (TGV) parameters.
`trough`	A logical value (default `FALSE`) indicating whether to calculate the age at cessation of growth velocity (ACGV) and the cessation of growth velocity (CGV) parameters.
`acgv`	A logical value (default `FALSE`) indicating whether to calculate the age at cessation of growth velocity from the velocity curve. If `TRUE`, the age at cessation of growth velocity (ACGV) and the cessation growth velocity (CGV) are calculated based on the percentage of the peak growth velocity, as defined by the `acgv_velocity` argument (see below). The `acgv_velocity` is typically set at 10 percent of the peak growth velocity. ACGV and CGV are calculated along with the uncertainty (SE and CI) around the ACGV and CGV parameters.
`acgv_velocity`	The percentage of the peak growth velocity to use when estimating `acgv`. The default value is `0.10`, i.e., 10 percent of the peak growth velocity.
`seed`	An integer (default `123`) that is passed to the estimation method to ensure reproducibility.

`estimation_method`
: A character string specifying the estimation method when calculating the velocity from the posterior draws. The `'fitted'` method internally calls `fitted_draws()`, while the `'predict'` method calls `predict_draws()`. See `brms::fitted.brmsfit()` and `brms::predict.brmsfit()` for details.

`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`.

`robust`
: A logical value to specify the summary options. If `FALSE` (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 median absolute deviation (MAD) are applied instead. Ignored if `summary` is `FALSE`.

`transform_draws`
: A function applied to individual draws from the posterior distribution before computing summaries (default `NULL`). The argument `transform_draws` is derived from the `marginaleffects::predictions()` function and should not be confused with the `transform` argument from the deprecated `brms::posterior_predict()` function. It's important to note that for both `marginaleffects::predictions()` and `marginaleffects::avg_predictions()`, the `transform_draws` argument takes precedence over the `transform` argument. Note that when `transform_draws = NULL`, an attempt is made to automatically set `transform_draws = 'exp'` for `dpar = 'sigma'`. User can set `transform_draws = FALSE` to turn off this automatic assignment of `'exp'` to the `transform_draws`. It is also important to set `transform_draws = FALSE` when computing the first derivative (velocity) for `dpar = 'sigma'`.

`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.

`future`
: A logical value (default `FALSE`) indicating whether to perform parallel computations. If `TRUE`, posterior summaries are computed in parallel using `future.apply::future_sapply()`.

`future_session`	A character string or a named list specifying the parallel plan when `future = TRUE`. • **Character string**: Defaults to `"multisession"`. Use `"multicore"` for forking (not supported on Windows). • **Named list**: Useful for advanced plans like `future.mirai::mirai_cluster()`. The list must contain: – `future_session`: The planner function (e.g., `mirai_cluster`). – Additional named arguments passed to the planner (e.g., daemons for `mirai::daemons()`). Example: `list(future_session = mirai_cluster, daemons = list(...))`.
`cores`	An integer specifying the number of cores for parallel execution. • If `NULL` (default), the number of cores is set to `future::availableCores() - 1`. • On non-Windows systems, this can be controlled globally via the `mc.cores` option.
`trim`	A numeric value (default `0`) indicating the number of long line segments to be excluded from the plot when the option 'u' or 'a' is selected. See sitar::plot.sitar for further details.
`layout`	A character string defining the plot layout. The default `'single'` layout overlays distance and velocity curves on a single plot when `opt` includes combinations like `'dv'`, `'Dv'`, `'dV'`, or `'DV'`. The alternative layout option `'facet'` uses `facet_wrap` from **ggplot2** to map and draw plots when `opt` includes two or more letters.
`linecolor`	The color of the lines when the layout is `'facet'`. The default is `NULL`, which sets the line color to `'grey50'`.
`linecolor1`	The color of the first line when the layout is `'single'`. For example, in `opt = 'dv'`, the distance line is controlled by `linecolor1`. The default `NULL` sets `linecolor1` to `'orange2'`.
`linecolor2`	The color of the second line when the layout is `'single'`. For example, in `opt = 'dv'`, the velocity line is controlled by `linecolor2`. The default `NULL` sets `linecolor2` to `'green4'`.
`label.x`	An optional character string to label the x-axis. If `NULL` (default), the x-axis label will be taken from the predictor (e.g., age).
`label.y`	An optional character string to label the y-axis. If `NULL` (default), the y-axis label will be taken from the plot type (e.g., distance, velocity). When `layout = 'facet'`, the label is removed, and the same label is used as the title.
`label.title`	An optional character string to label the title. Default `NULL`.
`label.subtitle`	An optional character string to label the title. Default `NULL`
`legendpos`	A character string to specify the position of the legend. If `NULL` (default), the legend position is set to 'bottom' for distance and velocity curves in the `'single'` layout. For individual-specific curves, the legend position is set to `'none'` to suppress the legend.
`linetype.apv`	A character string to specify the type of the vertical line marking the APGV. Default `NULL` sets the linetype to `dotted`.

`linewidth.main`	A numeric value to specify the line width for distance and velocity curves. The default `NULL` sets the width to 0.35.
`linewidth.apv`	A numeric value to specify the width of the vertical line marking the APGV. The default `NULL` sets the width to 0.25.
`linetype.groupby`	A character string specifying the line type for distance and velocity curves when drawing plots for a model with factor covariates or individual-specific curves. The default is `NA`, which sets the line type to 'solid' and suppresses legends.
`color.groupby`	A character string specifying the line color for distance and velocity curves when drawing plots for a model with factor covariates or individual-specific curves. The default is `NA`, which suppresses legends.
`band.alpha`	A numeric value to specify the transparency of the CI bands around the curves. The default `NULL` sets the transparency to 0.4.
`show_age_takeoff`	A logical value (default `TRUE`) to indicate whether to display the ATGV line(s) on the plot.
`show_age_peak`	A logical value (default `TRUE`) to indicate whether to display the APGV line(s) on the plot.
`show_age_cessation`	A logical value (default `TRUE`) to indicate whether to display the ACGV line(s) on the plot.
`show_vel_takeoff`	A logical value (default `FALSE`) to indicate whether to display the TGV line(s) on the plot.
`show_vel_peak`	A logical value (default `FALSE`) to indicate whether to display the PGV line(s) on the plot.
`show_vel_cessation`	A logical value (default `FALSE`) to indicate whether to display the CGV line(s) on the plot.
`returndata`	A logical value (default `FALSE`) to indicate whether to plot the data or return it as a `data.frame`.
`returndata_add_parms`	A logical value (default `FALSE`) to specify whether to add growth parameters to the returned `data.frame`. Ignored when `returndata = FALSE`. Growth parameters are added when the `opt` argument includes 'v' or 'V' and `apv = TRUE`.
`parms_eval`	A logical value to specify whether or not to compute growth parameters on the fly. This is for internal use only and is mainly needed for compatibility across internal functions.
`idata_method`	A character string specifying the interpolation method (default `NULL`). The number of interpolation points is controlled by the `ipts` argument. Available options: • `'m1'`: Adapted from the **iapvbs** package (documented here). This method internally constructs the data frame based on the model configuration. *Note:* This method may fail if the model includes covariates (especially in `univariate_by` models).

- 'm2': Based on the **JMbayes** package (documented here). This method uses the exact data frame from the model fit (fit$data) and is generally more robust.

If idata_method = NULL, 'm2' is automatically selected. It is recommended to use 'm2' if 'm1' encounters errors with covariate-dependent models.

parms_method	A character string specifying the method used when evaluating parms_eval. The default method is getPeak, which uses the sitar::getPeak() function from the sitar package. Alternatively, findpeaks uses the findpeaks function from the pracma package. This parameter is for internal use and ensures compatibility across internal functions.
verbose	A logical argument (default FALSE) to specify whether to print information collected during the setup of the object(s).
fullframe	A logical value indicating whether to return a fullframe object in which newdata is bound to the summary estimates. Note that fullframe cannot be used with summary = FALSE, and it is only applicable when idata_method = 'm2'. A typical use case is when fitting a univariate_by model. This option is mainly for internal use.

dummy_to_factor

A named list (default NULL) to convert dummy variables into a factor variable. The list must include the following elements:

- factor.dummy: A character vector of dummy variables to be converted to factors.
- factor.name: The name for the newly created factor variable (default is 'factor.var' if NULL).
- factor.level: A vector specifying the factor levels. If NULL, levels are taken from factor.dummy. If factor.level is provided, its length must match factor.dummy.

expose_function

A logical value or NULL (default FALSE). Controls whether Stan functions are exposed for post-processing.

- TRUE: Explicitly exposes Stan functions saved from the model fit. This is required when calculating fit criteria or bayes_R2 during model optimization.
- FALSE (default): Stan functions are not exposed.
- NULL: The setting is inherited from the original bsitar() model fit configuration.

Note: In the optimize_model() function, the default is NULL (inheriting behavior), whereas other post-processing functions default to FALSE.

usesavedfuns	A logical value (default NULL) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the expose_functions argument from the bsitar() call. Manual specification of usesavedfuns is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.
clearenvfuns	A logical value indicating whether to clear the exposed Stan functions from the environment (TRUE) or not (FALSE). If NULL, clearenvfuns is set based

	on the value of `usesavedfuns`: `TRUE` if `usesavedfuns = TRUE`, or `FALSE` if `usesavedfuns = FALSE`.
`funlist`	A list (default `NULL`) specifying function names. This is rarely needed, as required functions are typically retrieved automatically. A use case for `funlist` is when `sigma_formula`, `sigma_formula_gr`, or `sigma_formula_gr_str` use an external function (e.g., `poly(age)`). The `funlist` should include function names defined in the `globalenv()`. For functions needing both distance and velocity curves (e.g., `plot_curves(..., opt = 'dv')`), `funlist` must include two functions: one for the distance curve and one for the velocity curve.
`xvar`	A character string (default `NULL`) specifying the `'x'` variable. Rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.
`difx`	A character string (default `NULL`) specifying the `'x'` variable that should be used for manual differentiation of the distance curve. Internally, the `xvar` is set as `difx` if specified. The argument `difx` is evaluated only when `dpar = 'sigma'`, ignored otherwise. Note that argument `xvar` itself is rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.
`idvar`	A character string (default `NULL`) specifying the `'id'` variable. Rarely used because `idvar` is inferred internally.
`itransform`	A character string (default `NULL`) indicating the variables names that are reverse transformed. Options are `c("x", "y", "sigma")`. The `itransform` is primarily used to get the `xvar` variable at original scale i.e., `itransform = 'x'`. To turn of all transformations, use `itransform = ""`. when `itransform = NULL`, the appropriate transformation for `xvar` is selected automatically. Note that when no match for `xvar` is found in the `data,frame`, the `itransform` will be ignored within the calling function, `'prepare_transformations()'`.
`newdata_fixed`	An indicator to specify whether to check data format and structure for the user provided `newdata`, and apply needed `prepare_data2` and `prepare_transformations` (`newdata_fixed = NULL`, default), return user provided `newdata` (`newdata = TRUE`) as it is without checking for the data format or applying `prepare_data2` and `prepare_transformations` (`newdata_fixed = 0`), check for the data format and if needed, prepare data format using `prepare_data2` (`newdata_fixed = 1`), or apply `prepare_transformations` only assuming that data format is correct (`newdata_fixed = 2`). It is strongly recommended that user either leave the `newdata = NULL` and `newdata_fixed = NULL` in which case data used in the model fitting is automatically retrieved and checked for the required data format and transformations, and if needed, `prepare_data2` and `prepare_transformations` are applied internally. The other flags provided for `newdata_fixed = 0, 1, 2` are mainly for the internal use during post-processing.
`envir`	The environment used for function evaluation. The default is `NULL`, which sets the environment to `parent.frame()`. Since most post-processing functions rely on **brms**, it is recommended to set `envir = globalenv()` or `envir = .GlobalEnv`, especially for derivatives like velocity curves.
`...`	Additional arguments passed to the `brms::fitted.brmsfit()` and `brms::predict()` functions.

Details

The **plot_curves()** function is a generic tool for visualizing the following six curves:

- Population average distance curve
- Population average velocity curve
- Individual-specific distance curves
- Individual-specific velocity curves
- Unadjusted individual growth curves (i.e., observed growth curves)
- Adjusted individual growth curves (adjusted for the model-estimated random effects)

Internally, **plot_curves()** calls the `growthparameters()` function to estimate and summarize the distance and velocity curves, as well as to compute growth parameters such as the age at peak growth velocity (APGV). The function also calls `fitted_draws()` or `predict_draws()` to make inferences based on posterior draws. As a result, **plot_curves()** can plot either fitted or predicted curves. For more details, see `fitted_draws()` and `predict_draws()` to understand the difference between fitted and predicted values.

Value

A plot object (default) or a `data.frame` when `returndata` = `TRUE`.

Author(s)

Satpal Sandhu <`satpal.sandhu@bristol.ac.uk`>

See Also

`growthparameters()` `fitted_draws()` `predict_draws()`

Examples

```
# Fit Bayesian SITAR model

# To avoid mode estimation which takes time, the Bayesian SITAR model is fit to
# the 'berkeley_exdata' and saved as an example fit ('berkeley_exfit').
# See 'bsitar' function for details on 'berkeley_exdata' and 'berkeley_exfit'.

# Check and confirm whether the model fit object 'berkeley_exfit' exists
berkeley_exfit <- getNsObject(berkeley_exfit)

model <- berkeley_exfit

# Population average distance and velocity curves with default options
plot_curves(model, opt = 'dv')

# Individual-specific distance and velocity curves with default options
# Note that \code{legendpos = 'none'} will suppress the legend positions.
# This suppression is useful when plotting individual-specific curves
```

```
plot_curves(model, opt = 'DV')

# Population average distance and velocity curves with APGV

plot_curves(model, opt = 'dv', apv = TRUE)

# Individual-specific distance and velocity curves with APGV

plot_curves(model, opt = 'DV', apv = TRUE)

# Population average distance curve, velocity curve, and APGV with CI bands
# To construct CI bands, growth parameters are first calculated for each
# posterior draw and then summarized across draws. Therefore,summary
# option must be set to FALSE

plot_curves(model, opt = 'dv', apv = TRUE, bands = 'dvp', summary = FALSE)

# Adjusted and unadjusted individual curves
# Note ipts = NULL (i.e., no interpolation of predictor (i.e., age) to plot a
# smooth curve). This is because it does not a make sense to interploate data
# when estimating adjusted curves. Also, layout = 'facet' (and not default
# layout = 'single') is used for the ease of visualizing the plotted
# adjusted and unadjusted individual curves. However, these lines can be
# superimposed on each other by setting the set layout = 'single'.
# For other plots shown above, layout can be set as 'single' or 'facet'

# Separate plots for adjusted and unadjusted curves (layout = 'facet')
plot_curves(model, opt = 'au', ipts = NULL, layout = 'facet')

# Superimposed adjusted and unadjusted curves (layout = 'single')
plot_curves(model, opt = 'au', ipts = NULL, layout = 'single')
```

`plot_ppc.bgmfit` *Perform posterior predictive checks for the Bayesian SITAR model*

Description

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

Usage

```
## S3 method for class 'bgmfit'
plot_ppc(
  model,
  type,
  ndraws = NULL,
  dpar = NULL,
  draw_ids = NULL,
  prefix = c("ppc", "ppd"),
  group = NULL,
  x = NULL,
  newdata = NULL,
  resp = NULL,
  size = 0.25,
  alpha = 0.7,
  trim = FALSE,
  bw = "nrd0",
  adjust = 1,
  kernel = "gaussian",
  n_dens = 1024,
  pad = TRUE,
  discrete = FALSE,
  binwidth = NULL,
  bins = NULL,
  breaks = NULL,
  freq = TRUE,
  y_draw = c("violin", "points", "both"),
  y_size = 1,
  y_alpha = 1,
  y_jitter = 0.1,
  verbose = FALSE,
  model_deriv = NULL,
  dummy_to_factor = NULL,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  newdata_fixed = NULL,
  envir = NULL,
  ...
)

plot_ppc(model, ...)
```

Arguments

`model`	An object of class `bgmfit`.
`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`	A positive integer indicating the number of posterior draws to use in estimation. If `NULL` (default), all draws are used.
`dpar`	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
`draw_ids`	An integer specifying the specific posterior draw(s) to use in estimation (default `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 estimation. If `NULL` (default), `newdata` is retrieved from the `model`.
`resp`	A character string (default `NULL`) to specify the response variable when processing posterior draws for `univariate_by` and `multivariate` models. See `bsitar()` for details on `univariate_by` and `multivariate` models.
`size`, `alpha`	Passed to the appropriate geom to control the appearance of the predictive distributions.
`trim`	A logical scalar passed to `ggplot2::geom_density()`.
`bw`, `adjust`, `kernel`, `n_dens`	Optional arguments passed to `stats::density()` to override default kernel density estimation parameters. `n_dens` defaults to `1024`.
`pad`	A logical scalar passed to `ggplot2::stat_ecdf()`.
`discrete`	For `ppc_ecdf_overlay()`, should the data be treated as discrete? The default is `FALSE`, in which case `geom="line"` is passed to `ggplot2::stat_ecdf()`. If `discrete` is set to `TRUE` then `geom="step"` is used.
`binwidth`	Passed to `ggplot2::geom_histogram()` to override the default binwidth.
`bins`	Passed to `ggplot2::geom_histogram()` to override the default binwidth.
`breaks`	Passed to `ggplot2::geom_histogram()` as an alternative to `binwidth`.
`freq`	For histograms, `freq=TRUE` (the default) puts count on the y-axis. Setting `freq=FALSE` puts density on the y-axis. (For many plots the y-axis text is off by default. To view the count or density labels on the y-axis see the `yaxis_text()` convenience function.)
`y_draw`	For `ppc_violin_grouped()`, a string specifying how to draw `y`: `"violin"` (default), `"points"` (jittered points), or `"both"`.
`y_jitter`, `y_size`, `y_alpha`	For `ppc_violin_grouped()`, if `y_draw` is `"points"` or `"both"` then `y_size`, `y_alpha`, and `y_jitter` are passed to to the `size`, `alpha`, and `width` arguments of `ggplot2::geom_jitter()` to control the appearance of `y` points. The default of `y_jitter=NULL` will let **ggplot2** determine the amount of jitter.
`verbose`	A logical argument (default `FALSE`) to specify whether to print information collected during the setup of the object(s).
`model_deriv`	A logical value specifying whether to estimate the velocity curve from the derivative function or by differentiating the distance curve. Set `model_deriv = TRUE` for functions that require the velocity curve, such as `growthparameters()` and `plot_curves()`. Set it to `NULL` for functions that use the distance curve (i.e., fitted values), such as `loo_validation()` and `plot_ppc()`.

`dummy_to_factor`
: A named list (default `NULL`) to convert dummy variables into a factor variable. The list must include the following elements:
  - `factor.dummy`: A character vector of dummy variables to be converted to factors.
  - `factor.name`: The name for the newly created factor variable (default is `'factor.var'` if `NULL`).
  - `factor.level`: A vector specifying the factor levels. If `NULL`, levels are taken from `factor.dummy`. If `factor.level` is provided, its length must match `factor.dummy`.

`expose_function`
: A logical value or `NULL` (default `FALSE`). Controls whether Stan functions are exposed for post-processing.
  - `TRUE`: Explicitly exposes Stan functions saved from the model fit. This is required when calculating `fit criteria` or `bayes_R2` during model optimization.
  - `FALSE` (default): Stan functions are not exposed.
  - `NULL`: The setting is inherited from the original `bsitar()` model fit configuration.

  Note: In the `optimize_model()` function, the default is `NULL` (inheriting behavior), whereas other post-processing functions default to `FALSE`.

`usesavedfuns`
: A logical value (default `NULL`) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the `expose_functions` argument from the `bsitar()` call. Manual specification of `usesavedfuns` is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.

`clearenvfuns`
: A logical value indicating whether to clear the exposed Stan functions from the environment (`TRUE`) or not (`FALSE`). If `NULL`, `clearenvfuns` is set based on the value of `usesavedfuns`: `TRUE` if `usesavedfuns = TRUE`, or `FALSE` if `usesavedfuns = FALSE`.

`newdata_fixed`
: An indicator to specify whether to check data format and structure for the user provided `newdata`, and apply needed `prepare_data2` and `prepare_transformations` (`newdata_fixed = NULL`, default), return user provided `newdata` (`newdata = TRUE`) as it is without checking for the data format or applying `prepare_data2` and `prepare_transformations` (`newdata_fixed = 0`), check for the data format and if needed, prepare data format using `prepare_data2` (`newdata_fixed = 1`), or apply `prepare_transformations` only assuming that data format is correct (`newdata_fixed = 2`). It is strongly recommended that user either leave the `newdata = NULL` and `newdata_fixed = NULL` in which case data used in the model fitting is automatically retrieved and checked for the required data format and transformations, and if needed, `prepare_data2` and `prepare_transformations` are applied internally. The other flags provided for `newdata_fixed = 0, 1, 2` are mainly for the internal use during post-processing.

`envir`
: The environment used for function evaluation. The default is `NULL`, which sets the environment to `parent.frame()`. Since most post-processing functions rely on **brms**, it is recommended to set `envir = globalenv()` or `envir = .GlobalEnv`, especially for derivatives like velocity curves.

`...` Additional arguments passed to the `brms::pp_check.brmsfit()` function. Please refer to `brms::pp_check.brmsfit()` for details.

Details

The **plot_ppc()** function is a wrapper around the `brms::pp_check()` function, which allows for the visualization of posterior predictive checks.

Value

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

Author(s)

Satpal Sandhu <`satpal.sandhu@bristol.ac.uk`>

Examples

```
# Fit Bayesian SITAR model

# To avoid mode estimation, which takes time, the Bayesian SITAR model is fit to
# the 'berkeley_exdata' and saved as an example fit ('berkeley_exfit').
# See the 'bsitar' function for details on 'berkeley_exdata' and 'berkeley_exfit'.

# Check and confirm whether the model fit object 'berkeley_exfit' exists
 berkeley_exfit <- getNsObject(berkeley_exfit)

model <- berkeley_exfit

plot_ppc(model, ndraws = NULL)
```

`predict_draws.bgmfit` *Estimate predicted values for the Bayesian SITAR model*

Description

The **predict_draws()** function is a wrapper around the `brms::predict.brmsfit()` function, which obtains predicted values (and their summary) from the posterior distribution. See `brms::predict.brmsfit()` for details. An alternative approach is to `get_predictions()` function which is based on the **marginaleffects**.

Usage

```
## S3 method for class 'bgmfit'
predict_draws(
  model,
  newdata = NULL,
  resp = NULL,
  dpar = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  re_formula = NA,
  allow_new_levels = FALSE,
  sample_new_levels = "uncertainty",
  incl_autocor = TRUE,
  numeric_cov_at = NULL,
  levels_id = NULL,
  avg_reffects = NULL,
  aux_variables = NULL,
  grid_add = NULL,
  ipts = NULL,
  deriv = 0,
  model_deriv = TRUE,
  summary = TRUE,
  robust = FALSE,
  transform_draws = NULL,
  probs = c(0.025, 0.975),
  xrange = NULL,
  xrange_search = NULL,
  parms_eval = FALSE,
  parms_method = "getPeak",
  idata_method = NULL,
  verbose = FALSE,
  fullframe = NULL,
  dummy_to_factor = NULL,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  funlist = NULL,
  xvar = NULL,
  difx = NULL,
  idvar = NULL,
  itransform = NULL,
  newdata_fixed = NULL,
  envir = NULL,
  ...
)

predict_draws(model, ...)
```

Arguments

`model`	An object of class `bgmfit`.
`newdata`	An optional data frame for estimation. If `NULL` (default), `newdata` is retrieved from the `model`.

resp	A character string (default NULL) to specify the response variable when processing posterior draws for univariate_by and multivariate models. See bsitar() for details on univariate_by and multivariate models.
dpar	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
ndraws	A positive integer indicating the number of posterior draws to use in estimation. If NULL (default), all draws are used.
draw_ids	An integer specifying the specific posterior draw(s) to use in estimation (default NULL).
re_formula	Option to indicate whether or not to include individual/group-level effects in the estimation. When NA (default), individual-level effects are excluded, and population average growth parameters are computed. When NULL, individual-level effects are included in the computation, and the resulting growth parameters are individual-specific. In both cases (NA or NULL), continuous and factor covariates are appropriately included in the estimation. Continuous covariates are set to their means by default (see numeric_cov_at for details), while factor covariates remain unaltered, allowing for the estimation of covariate-specific population average and individual-specific growth parameters.
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.
numeric_cov_at	An optional (named list) argument to specify the value of continuous covariate(s). The default NULL option sets the continuous covariate(s) to their mean. Alternatively, a named list can be supplied to manually set these values. For example, numeric_cov_at = list(xx = 2) will set the continuous covariate variable 'xx' to 2. The argument numeric_cov_at is ignored when no continuous covariates are included in the model.
levels_id	An optional argument to specify the ids for the hierarchical model (default NULL). It is used only when the model is applied to data with three or more levels of hierarchy. For a two-level model, levels_id is automatically inferred

from the model fit. For models with three or more levels, `levels_id` is inferred from the model fit under the assumption that hierarchy is specified from the lowest to the uppermost level, i.e., `id` followed by `study`, where `id` is nested within `study`. However, it is not guaranteed that `levels_id` is sorted correctly, so it is better to set it manually when fitting a model with three or more levels of hierarchy.

`avg_reffects`	An optional argument (default `NULL`) to calculate (marginal/average) curves and growth parameters, such as APGV and PGV. If specified, it must be a named list indicating the `over` (typically a level 1 predictor, such as age), `feby` (fixed effects, typically a factor variable), and `reby` (typically `NULL`, indicating that parameters are integrated over the random effects). For example, `avg_reffects = list(feby = 'study', reby = NULL, over = 'age')`.
`aux_variables`	An optional argument to specify the variable(s) that can be passed to the `ipts` argument (see below). This is useful when fitting location-scale models and measurement error models. If post-processing functions throw an error such as `variable 'x' not found in either 'data' or 'data2'`, consider using `aux_variables`.
`grid_add`	An optional argument to specify the variable(s) that can be passed to the `marginaleffects::datagrid()` This is useful when fitting location-scale models and measurement error models. If post-processing functions throw an error such as `variable 'x' not found in either 'data' or 'data2'`, consider using `grid_add`. Note that unlike `aux_variables` which are passed to the internal data functions such as `'get.newdata'`, the `grid_add` are passed to the `marginaleffects::datagrid()`.
`ipts`	An integer specifying the number of points for interpolating the predictor variable (e.g., age) to generate smooth curves for predictions and plots. This value is used as the `length.out` argument for `seq()`, controlling the smoothness of distance and velocity curves without altering the predictor range. `NULL` **(the default)** Engages automatic behavior based on the `dpar` argument: it is internally set to `50` for the mean response (`dpar = 'mu'`), ensuring smooth curves, while for the distributional parameters (e.g., `dpar = 'sigma'`), no interpolation is performed. **An integer (e.g.,** `100`**)** Explicitly sets the number of interpolation points. `FALSE` Disables all interpolation, forcing predictions to be made only at the original data points of the predictor variable. This argument affects the following post-processing functions: `fitted_draws()`, `predict_draws()`, `growthparameters()`, `plot_curves()`, `get_predictions()`, `get_comparisons()`, and `get_growthparameters()`.
`deriv`	An integer indicating whether to estimate the distance curve or its derivative (velocity curve). The default `deriv = 0` is for the distance curve, while `deriv = 1` is for the velocity curve.
`model_deriv`	A logical value specifying whether to estimate the velocity curve from the derivative function or by differentiating the distance curve. Set `model_deriv = TRUE` for functions that require the velocity curve, such as `growthparameters()` and `plot_curves()`. Set it to `NULL` for functions that use the distance curve (i.e., fitted values), such as `loo_validation()` and `plot_ppc()`.
`summary`	A logical value indicating whether only the estimate should be computed (`TRUE`), or whether the estimate along with SE and CI should be returned (`FALSE`, default). Setting `summary` to `FALSE` will increase computation time. Note that

	`summary = FALSE` is required to obtain correct estimates when `re_formula = NULL`.
`robust`	A logical value to specify the summary options. If `FALSE` (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 median absolute deviation (MAD) are applied instead. Ignored if `summary` is `FALSE`.
`transform_draws`	A function applied to individual draws from the posterior distribution before computing summaries (default `NULL`). The argument `transform_draws` is derived from the `marginaleffects::predictions()` function and should not be confused with the `transform` argument from the deprecated `brms::posterior_predict()` function. It's important to note that for both `marginaleffects::predictions()` and `marginaleffects::avg_predictions()`, the `transform_draws` argument takes precedence over the `transform` argument. Note that when `transform_draws = NULL`, an attempt is made to automatically set `transform_draws = 'exp'` for `dpar = 'sigma'`. User can set `transform_draws = FALSE` to turn off this automatic assignment of `'exp'` to the `transform_draws`. It is also important to set `transform_draws = FALSE` when computing the first derivative (velocity) for `dpar = 'sigma'`.
`probs`	The percentiles to be computed by the `quantile` function. Only used if `summary` is `TRUE`.
`xrange`	An integer to set the predictor range (e.g., age) when executing the interpolation via `ipts`. By default, `NULL` sets the individual-specific predictor range. Setting `xrange = 1` applies the same range for individuals within the same higher grouping variable (e.g., study). Setting `xrange = 2` applies an identical range across the entire sample. Alternatively, a numeric vector (e.g., `xrange = c(6, 20)`) can be provided to set the range within the specified values.
`xrange_search`	A vector of length two or a character string `'range'` to set the range of the predictor variable (`x`) within which growth parameters are searched. This is useful when there is more than one peak and the user wants to summarize the peak within a specified range of the `x` variable. The default value is `xrange_search = NULL`.
`parms_eval`	A logical value to specify whether or not to compute growth parameters on the fly. This is for internal use only and is mainly needed for compatibility across internal functions.
`parms_method`	A character string specifying the method used when evaluating `parms_eval`. The default method is `getPeak`, which uses the `sitar::getPeak()` function from the `sitar` package. Alternatively, `findpeaks` uses the `findpeaks` function from the `pracma` package. This parameter is for internal use and ensures compatibility across internal functions.
`idata_method`	A character string specifying the interpolation method (default `NULL`). The number of interpolation points is controlled by the `ipts` argument. Available options: • `'m1'`: Adapted from the **iapvbs** package (documented here). This method internally constructs the data frame based on the model configuration. *Note:* This method may fail if the model includes covariates (especially in `univariate_by` models).

- `'m2'`: Based on the **JMbayes** package (documented here). This method uses the exact data frame from the model fit (`fit$data`) and is generally more robust.

If `idata_method = NULL`, `'m2'` is automatically selected. It is recommended to use `'m2'` if `'m1'` encounters errors with covariate-dependent models.

`verbose` A logical argument (default `FALSE`) to specify whether to print information collected during the setup of the object(s).

`fullframe` A logical value indicating whether to return a `fullframe` object in which `newdata` is bound to the summary estimates. Note that `fullframe` cannot be used with `summary = FALSE`, and it is only applicable when `idata_method = 'm2'`. A typical use case is when fitting a `univariate_by` model. This option is mainly for internal use.

`dummy_to_factor`
A named list (default `NULL`) to convert dummy variables into a factor variable. The list must include the following elements:

- `factor.dummy`: A character vector of dummy variables to be converted to factors.
- `factor.name`: The name for the newly created factor variable (default is `'factor.var'` if `NULL`).
- `factor.level`: A vector specifying the factor levels. If `NULL`, levels are taken from `factor.dummy`. If `factor.level` is provided, its length must match `factor.dummy`.

`expose_function`
A logical value or `NULL` (default `FALSE`). Controls whether Stan functions are exposed for post-processing.

- `TRUE`: Explicitly exposes Stan functions saved from the model fit. This is required when calculating `fit criteria` or `bayes_R2` during model optimization.
- `FALSE` (default): Stan functions are not exposed.
- `NULL`: The setting is inherited from the original `bsitar()` model fit configuration.

Note: In the `optimize_model()` function, the default is `NULL` (inheriting behavior), whereas other post-processing functions default to `FALSE`.

`usesavedfuns` A logical value (default `NULL`) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the `expose_functions` argument from the `bsitar()` call. Manual specification of `usesavedfuns` is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.

`clearenvfuns` A logical value indicating whether to clear the exposed Stan functions from the environment (`TRUE`) or not (`FALSE`). If `NULL`, `clearenvfuns` is set based on the value of `usesavedfuns`: `TRUE` if `usesavedfuns = TRUE`, or `FALSE` if `usesavedfuns = FALSE`.

`funlist` A list (default `NULL`) specifying function names. This is rarely needed, as required functions are typically retrieved automatically. A use case for `funlist` is when `sigma_formula`, `sigma_formula_gr`, or `sigma_formula_gr_str` use an external function (e.g., `poly(age)`). The `funlist` should include function names defined in the `globalenv()`. For functions needing both distance and velocity curves (e.g., `plot_curves(..., opt = 'dv')`), `funlist` must include two functions: one for the distance curve and one for the velocity curve.

`xvar`	A character string (default `NULL`) specifying the `'x'` variable. Rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.
`difx`	A character string (default `NULL`) specifying the `'x'` variable that should be used for manual differentiation of the distance curve. Internally, the `xvar` is set as `difx` if specified. The argument `difx` is evaluated only when `dpar = 'sigma'`, ignored otherwise. Note that argument `xvar` itself is rarely used because `xvar` is inferred internally. A use case is when conflicting variables exist (e.g., `sigma_formula`) and user wants to set a specific variable as `'x'`.
`idvar`	A character string (default `NULL`) specifying the `'id'` variable. Rarely used because `idvar` is inferred internally.
`itransform`	A character string (default `NULL`) indicating the variables names that are reverse transformed. Options are `c("x", "y", "sigma")`. The `itransform` is primarily used to get the `xvar` variable at original scale i.e., `itransform = 'x'`. To turn of all transformations, use `itransform = ""`. when `itransform = NULL`, the appropriate transformation for `xvar` is selected automatically. Note that when no match for `xvar` is found in the `data,frame`, the `itransform` will be ignored within the calling function, `'prepare_transformations()'`.
`newdata_fixed`	An indicator to specify whether to check data format and structure for the user provided `newdata`, and apply needed `prepare_data2` and `prepare_transformations` (`newdata_fixed = NULL`, default), return user provided `newdata` (`newdata = TRUE`) as it is without checking for the data format or applying `prepare_data2` and `prepare_transformations` (`newdata_fixed = 0`), check for the data format and if needed, prepare data format using `prepare_data2` (`newdata_fixed = 1`), or apply `prepare_transformations` only assuming that data format is correct (`newdata_fixed = 2`). It is strongly recommended that user either leave the `newdata = NULL` and `newdata_fixed = NULL` in which case data used in the model fitting is automatically retrieved and checked for the required data format and transformations, and if needed, `prepare_data2` and `prepare_transformations` are applied internally. The other flags provided for `newdata_fixed = 0, 1, 2` are mainly for the internal use during post-processing.
`envir`	The environment used for function evaluation. The default is `NULL`, which sets the environment to `parent.frame()`. Since most post-processing functions rely on **brms**, it is recommended to set `envir = globalenv()` or `envir = .GlobalEnv`, especially for derivatives like velocity curves.
`...`	Additional arguments passed to the `brms::predict.brmsfit()` function. Please see `brms::predict.brmsfit()` for details on the various options available.

Details

The **predict_draws()** function computes the expected values from the posterior distribution. The `brms::predict.brmsfit()` function from the **brms** package can be used to obtain predicted (distance) values when the outcome (e.g., height) is untransformed. However, when the outcome is log or square root transformed, the `brms::predict.brmsfit()` function will return the expected curve on the log or square root scale. In contrast, the **predict_draws()** function returns the expected values on the original scale. Furthermore, **predict_draws()** also computes the first derivative (velocity), again on the original scale, after making the necessary back-transformation. Aside from these differences, both functions (`brms::predict.brmsfit()` and **predict_draws()**) work similarly. In other words, the user can specify all the options available in `brms::predict.brmsfit()`.

Value

An array of predicted response values. See `brms::predict.brmsfit()` for details.

Author(s)

Satpal Sandhu <`satpal.sandhu@bristol.ac.uk`>

See Also

`brms::predict.brmsfit()`

Examples

```
# Fit Bayesian SITAR model

# To avoid mode estimation, which takes time, the Bayesian SITAR model is fit
# to the 'berkeley_exdata' and saved as an example fit ('berkeley_exfit').
# See the 'bsitar' function for details on 'berkeley_exdata' and
# berkeley_exfit'.

# Check and confirm whether the model fit object 'berkeley_exfit' exists
 berkeley_exfit <- getNsObject(berkeley_exfit)

model <- berkeley_exfit

# Population average distance curve
predict_draws(model, deriv = 0, re_formula = NA)

# Individual-specific distance curves
predict_draws(model, deriv = 0, re_formula = NULL)

# Population average velocity curve
predict_draws(model, deriv = 1, re_formula = NA)

# Individual-specific velocity curves
predict_draws(model, deriv = 1, re_formula = NULL)
```

`update_model.bgmfit` *Update the Bayesian SITAR model*

Description

The **update_model()** function is a wrapper around the `update()` function from the **brms** package, which refits the model based on the user-specified updated arguments.

Usage

```
## S3 method for class 'bgmfit'
update_model(
  model,
  newdata = NULL,
  recompile = NULL,
  expose_function = FALSE,
  verbose = FALSE,
  check_newargs = FALSE,
  envir = NULL,
  ...
)

update_model(model, ...)
```

Arguments

`model`	An object of class `bgmfit`.
`newdata`	An optional `data.frame` to be used when updating the model. If `NULL` (default), the data used in the original model fit is reused. Note that data-dependent default priors are not automatically updated.
`recompile`	A logical value indicating whether the Stan model should be recompiled. When `NULL` (default), **update_model()** tries to internally determine whether recompilation is required. Setting `recompile` to `FALSE` will ignore any changes in the Stan code.
`expose_function`	A logical value or `NULL` (default `FALSE`). Controls whether Stan functions are exposed for post-processing. • `TRUE`: Explicitly exposes Stan functions saved from the model fit. This is required when calculating `fit criteria` or `bayes_R2` during model optimization. • `FALSE` (default): Stan functions are not exposed. • `NULL`: The setting is inherited from the original `bsitar()` model fit configuration. **Note**: In the `optimize_model()` function, the default is `NULL` (inheriting behavior), whereas other post-processing functions default to `FALSE`.

verbose	A logical argument (default FALSE) to specify whether to print information collected during the setup of the object(s).
check_newargs	A logical value (default FALSE) indicating whether to check if the arguments in the original model fit and the update_model are identical. When check_newargs = TRUE and the arguments are identical, it indicates that an update is unnecessary. In this case, the original model object is returned, along with a message if verbose = TRUE.
envir	The environment used for function evaluation. The default is NULL, which sets the environment to parent.frame(). Since most post-processing functions rely on **brms**, it is recommended to set envir = globalenv() or envir = .GlobalEnv, especially for derivatives like velocity curves.
...	Other arguments passed to [brms::brm()].

Details

This function is an adapted version of the **update()** function from the **brms** package.

Value

An updated object of class brmsfit.

Author(s)

Satpal Sandhu <satpal.sandhu@bristol.ac.uk>

Examples

```
# Fit Bayesian SITAR model

# To avoid mode estimation which takes time, the Bayesian SITAR model fit to
# the 'berkeley_exdata' has been saved as an example fit ('berkeley_exfit').
# See 'bsitar' function for details on 'berkeley_exdata' and 'berkeley_exfit'.

# Check and confirm whether model fit object 'berkeley_exfit' exists
 berkeley_exfit <- getNsObject(berkeley_exfit)

model <- berkeley_exfit

# Update model
# Note that in case all arguments supplied to the update_model() call are
# same as the original model fit (checked via check_newargs = TRUE), then
# original model object is returned.
# To explicitly get this information whether model is being updated or not,
# user can set verbose = TRUE. The verbose = TRUE also useful in getting the
# information regarding what all arguments have been changed as compared to
# the original model.

model2 <- update_model(model, df = 5, check_newargs = TRUE, verbose = TRUE)
```

Index

∗ **datasets**
berkeley, 6
berkeley_exdata, 8
berkeley_exfit, 9

add_model_criterion (add_model_criterion.bgmfit), 3
add_model_criterion.bgmfit, 3

base::jitter(), *39*
base::on.exit(), *72*, *90*, *105*, *125*, *145*
base::round(), *64*, *79*, *97*, *112*, *137*, *152*, *164*
base::Vectorize(), *51*
bayestestR::equivalence_test(), *88*, *116*, *117*, *121*, *122*, *124–127*
bayestestR::p_direction(), *88*, *116*, *117*, *122*, *124–127*
berkeley, 6, *8*
berkeley_exdata, 8, *9*
berkeley_exfit, 9
brms::add_criterion(), *3*, *132*, *152*
brms::add_ic(), *6*
brms::add_loo, *3*, *6*
brms::add_waic(), *6*
brms::bayes_R2(), *3*, *6*
brms::bridge_sampler(), *42*
brms::brm(), *20*, *21*, *27*, *39*, *41–45*, *49*, *131*
brms::brmsformula(), *18*, *19*, *27*, *43*, *49*
brms::conditional_effects, *160*
brms::conditional_effects(), *160*
brms::custom_family(), *27*
brms::fitted.brmsfit, *59*, *107*
brms::fitted.brmsfit(), *53*, *59*, *91*, *107*, *111*, *156*, *166*
brms::hypothesis(), *43*, *116*, *120*, *125–127*
brms::lf(), *21*, *22*
brms::loo(), *130*, *132*, *133*
brms::loo_moment_match(), *131*
brms::nlf(), *21*, *22*
brms::opencl(), *42*
brms::posterior_linpred(), *133*
brms::posterior_predict(), *56*, *69*, *85*, *102*, *110*, *142*, *156*, *166*, *180*
brms::pp_check(), *176*
brms::pp_check.brmsfit(), *176*
brms::predict.brmsfit(), *107*, *111*, *156*, *166*, *176*, *182*, *183*
brms::prior(), *27*, *29*, *49*
brms::reloo(), *131*
brms::save_pars(), *43*
brms::set_prior(), *43*
bsitar, 9
bsitar(), *5*, *15*, *16*, *24*, *25*, *54*, *58*, *63*, *73*, *77*, *90*, *95*, *106*, *109*, *114*, *126*, *132*, *135*, *146*, *152*, *157*, *159*, *163*, *169*, *174*, *175*, *178*, *181*

coef.brmsfit, *121*

estimate_density(), *123*
expose_model_functions (expose_model_functions.bgmfit), 51
expose_model_functions.bgmfit, 50

fitted_draws (fitted_draws.bgmfit), 53
fitted_draws(), *56*, *59*, *65*, *81*, *93*, *98*, *110*, *112*, *115*, *138*, *156*, *157*, *165*, *166*, *171*, *179*
fitted_draws.bgmfit, 53
foreach::foreach(), *66*, *82*, *99*, *139*
future, *45*
future.apply::future_lapply(), *66*, *82*, *99*, *139*
future.apply::future_sapply(), *65*, *81*, *98*, *112*, *138*, *166*
future.mirai::mirai_cluster(), *66*, *81*, *98*, *113*, *138*, *167*
future::plan(), *66*, *82*, *99*, *139*

get_comparisons
(get_comparisons.bgmfit), 61
get_comparisons(), *56*, *65*, *81*, *98*, *112*, *138*, *157*, *165*, *179*
get_comparisons.bgmfit, 61
get_gp (get_growthparameters.bgmfit), 75
get_growthparameters
(get_growthparameters.bgmfit), 75
get_growthparameters(), *56*, *65*, *81*, *98*, *108*, *112*, *124*, *126*, *138*, *157*, *165*, *179*
get_growthparameters.bgmfit, 75
get_predictions
(get_predictions.bgmfit), 93
get_predictions(), *53*, *56*, *65*, *81*, *93*, *98*, *112*, *138*, *157*, *160*, *161*, *165*, *176*, *179*
get_predictions.bgmfit, 93
getNsObject, 60
ggplot2::geom_density(), *174*
ggplot2::geom_histogram(), *174*
ggplot2::geom_jitter(), *174*
ggplot2::stat_ecdf(), *174*
growthparameters
(growthparameters.bgmfit), 108
growthparameters(), *56*, *65*, *81*, *98*, *112*, *138*, *157*, *163*, *165*, *171*, *179*
growthparameters.bgmfit, 108
growthparameters_comparison
(get_growthparameters.bgmfit), 75

Hmisc::rcspline.eval(), *15*, *18*
hypothesis_test
(hypothesis_test.bgmfit), 116
hypothesis_test(), *122*
hypothesis_test.bgmfit, 116

installr::kill_all_Rscript_s(), *66*, *81*, *99*, *138*
is.bgmfit, 129

loo, *131*
loo::loo_compare(), *131, 132*
loo::loo_moment_match(), *131*
loo_compare, *4*
loo_moment_match, *131*
loo_validation (loo_validation.bgmfit), 130
loo_validation.bgmfit, 130

make_conditions, *155*
marginal_comparison
(get_comparisons.bgmfit), 61
marginal_comparisons
(get_comparisons.bgmfit), 61
marginal_draws
(get_predictions.bgmfit), 93
marginal_growthparameters
(get_growthparameters.bgmfit), 75
marginaleffects::avg_comparisons(), *61*, *67*, *74*, *75*, *82*, *83*, *92*, *100*, *139*, *140*
marginaleffects::avg_predictions(), *56*, *69*, *85*, *93*, *99*, *100*, *102*, *107*, *110*, *142*, *156*, *166*, *180*
marginaleffects::comparisons(), *61*, *67*, *68*, *72*, *74*, *75*, *82*, *83*, *88*, *92*, *100*, *101*, *117*, *120*, *122*, *124–127*, *139*, *140*
marginaleffects::datagrid(), *55*, *63*, *64*, *78*, *80*, *89*, *95*, *97*, *112*, *136*, *145*, *164*, *179*
marginaleffects::plot_comparisons(), *67*, *74*, *82*, *92*, *139*
marginaleffects::plot_predictions(), *93*, *100*, *107*
marginaleffects::posterior_draws(), *68*, *84*, *101*, *141*
marginaleffects::predictions(), *56*, *68*, *69*, *85*, *93*, *99–102*, *107*, *110*, *142*, *148*, *156*, *166*, *180*
mirai::daemons(), *66*, *81*, *98*, *113*, *138*, *167*
modelbased_growthparameters
(modelbased_growthparameters.bgmfit), 133
modelbased_growthparameters.bgmfit, 133

nlme::varConstPower(), *23*
nlme::varExp(), *23*
nlme::varPower(), *23*

optimize_model (optimize_model.bgmfit), 149

optimize_model(), *5*, *58*, *73*, *90*, *105*, *114*, *132*, *146*, *152*, *158*, *169*, *175*, *181*, *184*
optimize_model.bgmfit, 149
options, *42*, *44*

p_direction(), *123*
parallel::makeCluster(), *82*, *139*
parallel::splitIndices(), *66*, *81*, *99*, *138*
pareto_k_ids, *131*
pd_to_p(), *123*
plan, *45*
plot_conditional_effects *(*plot_conditional_effects.bgmfit*)*, 153
plot_conditional_effects.bgmfit, 153
plot_curves *(*plot_curves.bgmfit*)*, 161
plot_curves(), *56*, *65*, *81*, *98*, *112*, *138*, *157*, *165*, *179*
plot_curves.bgmfit, 161
plot_ppc *(*plot_ppc.bgmfit*)*, 172
plot_ppc.bgmfit, 172
PPC, *173*
predict_draws *(*predict_draws.bgmfit*)*, 176
predict_draws(), *56*, *65*, *81*, *93*, *98*, *111*, *112*, *115*, *138*, *156*, *157*, *165*, *166*, *171*, *179*
predict_draws.bgmfit, 176

ranef.brmsfit, *121*
reloo, *131*
rope_range(), *123*
rstan::expose_stan_functions(), *51*, *52*
rstan::rstan(), *45*
rstan::stan_model, *44*
rstan::stan_model(), *52*
rstan::stanc(), *52*

saveRDS, *4*, *44*, *151*
seq(), *55*, *65*, *80*, *98*, *112*, *138*, *157*, *165*, *179*
set.seed, *121*
sitar::getPeak(), *57*, *69*, *85*, *113*, *115*, *141*, *169*, *180*
sitar::getTakeoff(), *69*, *85*, *115*, *141*
sitar::getTrough(), *69*, *85*, *115*, *141*
sitar::plot.sitar, *167*
sitar::sitar(), *21*, *46*
splines2::nsk(), *18*, *22*, *46*
splines2::nsp(), *18*, *46*
splines::ns(), *46*
stanvars, *64*, *78*, *95*, *136*
stats::density(), *174*
stats::model.matrix(), *19*

update, *44*
update_model, *152*
update_model *(*update_model.bgmfit*)*, 184
update_model.bgmfit, 184

variables, *121*

yaxis_text(), *174*