

Package ‘bsitar’

August 28, 2026

Type Package

Title Bayesian Super Imposition by Translation and Rotation Growth Curve Analysis

Version 0.4.0

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](https://doi.org/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 (>= 2.10.1), dplyr (>= 1.2.1), rlang (>= 1.1.2), Rdpack (>= 2.6.6), insight (>= 1.5.3), data.table (>= 1.18.6.1), collapse (>= 2.1.7), margaleffects (>= 0.32.0), magrittr, methods, utils

Suggests sitar (>= 1.5.0), ggplot2 (>= 4.0.3), tidybayes (>= 3.0.7), bayesplot (>= 1.16.0), posterior (>= 1.7.0), testthat (>= 3.3.2), ggthemes (>= 0.5.6), jtools (>= 2.3.0), splines2 (>= 0.5.4), scales (>= 1.3.0), kableExtra (>= 1.4.0), knitr (>= 1.51), future (>= 1.75.0), future.callr (>= 1.0.0), future.apply (>= 1.20.2), future.mirai (>= 1.0.0), mirai (>= 2.7.2), doParallel (>= 1.0.17), foreach (>= 1.5.2), doFuture (>= 1.3.0), fastplyr (>= 0.9.92), cheapr (>= 1.5.1), dtplyr (>= 1.3.3), checkmate (>= 2.3.3), installr (>= 0.23.4), bayestestR (>= 0.18.1), rmarkdown (>= 2.31.0), modelsummary (>= 2.6.0), performance (>= 0.17.1), reformulas (>= 0.4.4), growthcleanr (>= 2.2.3), R.rsp (>= 0.46.0), openxlsx2 (>= 1.29), priorsense (>= 1.2.0), distributional (>= 0.8.1), extraDistr (>= 1.10.0.5), ggdist (>= 3.3.3), flextable (>= 0.10.0), flexlsx (>= 1.0.0), Hmisc, R.utils, parallel, MASS, Matrix, tidyr, nlme, purrr, forcats, patchwork, tibble, pracma, bookdown, spelling, boot, 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 8.0.0

Language en-US

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

Repository CRAN

Date/Publication 2026-08-28 15:40:08 UTC

Contents

add_model_criterion.bgmfit	3
berkeley	7
berkeley_exdata	8
berkeley_exfit	9
bsitar	9
compare_models.bgmfit	50

expose_model_functions.bgmfit	52
fitted_draws.bgmfit	54
getNsObject	62
get_comparisons.bgmfit	63
get_growthparameters.bgmfit	77
get_model_criterion.bgmfit	95
get_predictions.bgmfit	98
growthparameters.bgmfit	113
hypothesis_test.bgmfit	122
is.bgmfit	135
loo_validation.bgmfit	136
modelbased_growthparameters.bgmfit	139
optimize_model.bgmfit	155
plot_caterpillar.bgmfit	159
plot_conditional_effects.bgmfit	165
plot_curves.bgmfit	173
plot_diagnostics.bgmfit	186
plot_ppc.bgmfit	193
predict_draws.bgmfit	197
prior_conflict.bgmfit	205
prior_sensitivity.bgmfit	207
prior_table.bgmfit	216
update_model.bgmfit	220

Index**223**

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,
    return_criteria = FALSE,
    verbose = FALSE,
    expose_function = FALSE,
    usesavedfuns = NULL,
    clearenvfuns = NULL,
    envir = NULL,
    ...
)

add_model_criterion(model, ...)

```

Arguments

<code>model</code>	An object of class <code>bgmfit</code> representing the model to which the fit criteria will be added.
<code>criterion</code>	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).
<code>ndraws</code>	A positive integer indicating the number of posterior draws to use in estimation. If NULL (default), all draws are used.
<code>draw_ids</code>	An integer specifying the specific posterior draw(s) to use in estimation (default NULL).
<code>compare</code>	A flag indicating if the information criteria of the models should be compared to each other via loo_compare .
<code>pointwise</code>	A flag indicating whether to compute the full log-likelihood matrix at once or separately for each observation. The latter approach is usually considerably slower but requires much less working memory. Accordingly, if one runs into memory issues, <code>pointwise = TRUE</code> is the way to go.
<code>model_name</code>	Optional name of the model. If NULL (the default) the name is taken from the call to <code>x</code> .
<code>overwrite</code>	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.
<code>force_save</code>	Logical; only relevant if <code>file</code> is specified and ignored otherwise. If TRUE, the fitted model object will be saved regardless of whether new criteria were added via <code>add_criterion</code> .

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 <code>future::availableCores() - 1</code>. • On non-Windows systems, this can be controlled globally via the <code>mc.cores</code> option.
model_deriv	Ignored.
return_model	A logical (default TRUE) to indicate whether to return the model object or just assign the criteria to the model. Mainly for internal use.
return_criteria	A logical (default FALSE) to indicate whether to return the criterion. Mainly for internal use.
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 <code>bsitar()</code> model fit configuration.

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

<code>usesavedfuns</code>	A logical value (default <code>NULL</code>) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the <code>expose_functions</code> argument from the <code>bsitar()</code> call. Manual specification of <code>usesavedfuns</code> is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.
<code>clearenvfun</code>	A logical value indicating whether to clear the exposed Stan functions from the environment (<code>TRUE</code>) or not (<code>FALSE</code>). If <code>NULL</code> , <code>clearenvfun</code> is set based on the value of <code>usesavedfuns</code> : <code>TRUE</code> if <code>usesavedfuns = TRUE</code> , or <code>FALSE</code> if <code>usesavedfuns = FALSE</code> .
<code>envir</code>	The environment used for function evaluation. The default is <code>NULL</code> , which sets the environment to <code>parent.frame()</code> . Since most post-processing functions rely on brms , it is recommended to set <code>envir = globalenv()</code> or <code>envir = .GlobalEnv</code> , especially for derivatives like velocity curves.
<code>...</code>	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.ilial Bi-ilial 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-ilial 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.

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**

A Bayesian implementation of the Super Imposition by Translation and Rotation (*SITAR*) model. The *SITAR* model is a nonlinear mixed-effects framework widely used to summarize individual growth trajectories (e.g., height or weight) from early childhood through adulthood.

The frequentist implementation of the *SITAR* model is available in the **sitar** package (Cole 2022). In contrast, **bsitar** extends this framework within a Bayesian paradigm, offering increased flexibility in model specification, improved uncertainty quantification, and richer inferential capabilities.

Beyond standard *Univariate analysis* (i.e., modeling a single outcome), **bsitar** supports *Univariate-by-subgroup analysis* and *Multivariate analysis* as described below.

- Univariate analysis: Modeling a single outcome, consistent with the traditional use of the **sitar** package.

- Univariate-by-subgroup analysis: Simultaneous modeling of a single outcome across subgroups defined by a factor variable (e.g., sex). Posterior draws for all subgroups are obtained within a single model object, facilitating direct comparison of parameters and formal hypothesis testing across groups.
- Multivariate analysis: Joint modeling of two or more outcomes, allowing for a more comprehensive representation of correlated growth processes.

Overall, the Bayesian implementation in **bsitar** provides a flexible and robust framework for growth curve modeling, extending the capabilities of the traditional 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 = "rcs",
  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", rescoring = 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, 50, autoscale = FALSE),
b_cov_prior_beta = normal(0, 2, autoscale = FALSE),
c_cov_prior_beta = normal(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, 0.2, autoscale = FALSE),
sigma_cov_prior_sd = normal(0, 0.1, 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, 0.5, 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 = "lm",
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 = 0.95, 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

<code>x</code>	Predictor variable (typically age in years). For a univariate model, <code>x</code> is a single variable. For <code>univariate_by</code> and <code>multivariate</code> models, <code>x</code> can either be the same for all sub-models, or different for each sub-model. For example, when fitting a bivariate model, <code>x = list(x1, x2)</code> specifies that <code>x1</code> is the predictor variable for the first sub-model, and <code>x2</code> is the predictor for the second sub-model. To use <code>x1</code> as a common predictor variable for both sub-models, user can specify <code>x = list(x1)</code> or simply <code>x = x1</code> .
<code>y</code>	Response variable (e.g., repeated height measurements). For <code>univariate</code> and <code>univariate_by</code> models, <code>y</code> is specified as a single variable. In the case of a <code>univariate_by</code> 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 <code>univariate_by = sex</code> creates response vectors <code>Female</code> and <code>Male</code> when <code>Female</code> is the first level and <code>Male</code> is the second level of the <code>sex</code> variable. In a <code>multivariate</code> model, the response variables are provided as a list, such as <code>y = list(y1, y2)</code> , where <code>y1</code> is the response variable for the first sub-model, and <code>y2</code> is the response for the second sub-model. Note that for the <code>multivariate</code> model, the data are not stacked; instead, response vectors are separate variables in the data and must be of equal length.
<code>id</code>	A factor variable uniquely identifying the groups (e.g., individuals) in the data frame. For <code>univariate_by</code> and <code>multivariate</code> models, the <code>id</code> can be the same (typically) for all sub-models, or different for each sub-model (see the <code>x</code> argument for details on setting different arguments for sub-models).
<code>data</code>	A data frame containing variables such as <code>x</code> , <code>y</code> , <code>id</code> , etc.
<code>df</code>	Degrees of freedom for the natural cubic spline design matrix (default 4). The <code>df</code> is used internally to construct knots (quantiles of the <code>x</code> distribution) for the spline design matrix. For <code>univariate_by</code> and <code>multivariate</code> models, the <code>df</code> can be the same across sub-models (e.g., <code>df = 4</code>) or different for each sub-model, such as <code>df = list(4, 5)</code> , where <code>df = 4</code> applies to the first sub-model and <code>df = 5</code> applies to the second sub-model.

knots	A numeric vector specifying the knots for the natural cubic spline design matrix (default NULL). Note that user 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., $x - x_{\text{offset}}$)

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, user can specify different fixed effect structures for each sub-model. For example, <code>fixed = list('a+b+c', 'a+b')</code> 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., $\text{mean}(x)$), • 'max': The maximum value of x (i.e., $\text{max}(x)$), • 'min': The minimum value of x (i.e., $\text{min}(x)$),

- '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), as 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 is used in fitting `univariate_by` models.

<code>a_formula</code>	Formula for the fixed effect parameter, <code>a</code> (default <code>~ 1</code>). Users can specify different formulas when fitting <code>univariate_by</code> and <code>multivariate</code> models. For example, <code>a_formula = list(~1, ~1 + cov)</code> specifies that the <code>a_formula</code> for the first sub-model includes only an intercept, while the second sub-model includes both an intercept and a covariate <code>cov</code>. The covariate(s) can be either continuous or factor variables. For factor covariates, dummy variables are created internally using <code>stats::model.matrix()</code>. The formula can include a combination of continuous and factor variables, as well as their interactions.
<code>b_formula</code>	Formula for the fixed effect parameter, <code>b</code> (default <code>~ 1</code>). See <code>a_formula</code> for details on how to specify the formula. The behavior and structure of <code>b_formula</code> are similar to <code>a_formula</code>.
<code>c_formula</code>	Formula for the fixed effect parameter, <code>c</code> (default <code>~ 1</code>). See <code>a_formula</code> for details on how to specify the formula. The behavior and structure of <code>c_formula</code> are similar to <code>a_formula</code>.
<code>d_formula</code>	Formula for the fixed effect parameter, <code>d</code> (default <code>~ 1</code>). See <code>a_formula</code> for details on how to specify the formula. The behavior and structure of <code>d_formula</code> are similar to <code>a_formula</code>.
<code>s_formula</code>	Formula for the fixed effect parameter, <code>s</code> (default <code>~ 1</code>). The <code>s_formula</code> sets up the spline design matrix. Typically, covariates are not included in the <code>s_formula</code> 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 <code>s_formula</code>. 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.
<code>a_formula_gr</code>	Formula for the random effect parameter <code>a</code> (default <code>~ 1</code>). As with <code>a_formula</code>, users may specify different formulas when fitting <code>univariate_by</code> and <code>multivariate</code> models. The formula can include continuous and/or factor covariates, as well as their interactions (see <code>a_formula</code> for details). In addition to defining the design matrix for the random effect parameter <code>a</code>, this argument allows specification of the grouping structure and correlation among random effects using vertical bar (<code> </code>) notation. For example, to include only random intercepts for parameters <code>a</code>, <code>b</code>, and <code>c</code>, one can specify: <pre>a_formula_gr = ~1, b_formula_gr = ~1, c_formula_gr = ~1.</pre> To include a grouping variable (e.g., <code>id</code>) and specify an unstructured correlation among the random effects, use:

```

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

```

Here, *i* is a shared identifier indicating that the random effects for *a*, *b*, and *c* are modeled as jointly correlated with an unstructured covariance matrix. See `[brms::brm()]` for further details on this notation.

Alternatively, the grouping structure and correlation can be specified using the `group_by` argument. The equivalent specification to the above can be written as:

```

a_formula_gr = ~1, b_formula_gr = ~1, c_formula_gr = ~1,
together with group_by = list(groupvar = id, cor = un), where id defines
the grouping variable and un specifies an unstructured correlation. See group_by
for additional details.

```

b_formula_gr Formula for the random effect parameter *b* (default ~ 1). See `a_formula_gr` for details.

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

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

a_formula_gr_str

Formula for the random effect parameter *a* (default NULL), used for specifying hierarchical models with three or more levels. For example, this may arise in data with repeated measurements (level 1) nested within individuals (level 2), which are further nested within studies (level 3).

For `a_formula_gr_str`, only the vertical bar (`|`) notation (see `a_formula_gr`) can be used to define grouping structures and correlation patterns. An example specification for a three-level model with random effect parameters *a*, *b*, and *c* is:

```

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 setup, `|i|` and `|i2|` define unstructured correlation among random effects at the individual and study levels, respectively. These identifiers must be distinct, as random effects are not correlated across different hierarchical levels.

More generally, users can specify models with additional hierarchical levels and include covariates within the random effect formulas as needed.

b_formula_gr_str

Formula for the random effect parameter *b* (default NULL), used for specifying hierarchical models with three or more levels. See `a_formula_gr_str` for details.

c_formula_gr_str

Formula for the random effect parameter *c* (default NULL), used for specifying hierarchical models with three or more levels. See `a_formula_gr_str` for details.

- d_formula_gr_str** Formula for the random effect parameter d (default NULL), used for specifying hierarchical models with three or more levels. See `a_formula_gr_str` for details.
- d_adjusted** Logical indicator for adjusting the scale of the predictor variable x when including the random effect parameter d. The coefficient for d is modeled as a linear function of x, i.e., $d * x$.
If FALSE (default), the original x is used. If TRUE, x is adjusted for the timing (b) and intensity (c) parameters as $(x - b) * \exp(c)$, so that the model includes $d * ((x - b) * \exp(c))$.
This adjusted scale represents individual developmental age rather than chronological age, making the parameter d more sensitive to differences in growth timing (e.g., pubertal timing) across individuals. See `[sitar::sitar()]` for further details.
- sigma_formula** Formula for the distributional (fixed effect) parameter sigma. This argument defines the fixed-effect design matrix for sigma, allowing inclusion of continuous and/or factor covariates, as well as their interactions (see `a_formula` for details). The parameter sigma is modeled on the log scale.
By default, `sigma_formula = NULL`, in which case `[brms::brm()]` estimates sigma as a residual standard deviation (RSD) parameter on the link scale. When specified, `sigma_formula`, together with `sigma_formula_gr` and `sigma_formula_gr_str`, enables modeling sigma with both fixed and random effects, analogous to other model parameters such as a, b, and c.
An alternative approach is to use the `dpar_formula` argument to specify the design matrix for sigma. The advantage of `dpar_formula` is its flexibility in supporting both linear and nonlinear specifications via `[brms::lf()]` and `[brms::nlf()]`. This allows additional control, such as centering predictors or removing the intercept using a $0 +$ formulation. However, a limitation of `dpar_formula` is that it does not support random effects for sigma.
The arguments `sigma_formula` and `dpar_formula` are mutually exclusive. When either is specified, the default estimation of sigma as an RSD parameter in `[brms::brm()]` is disabled.
External functions (e.g., `poly`, `splines::ns`) can be used within `sigma_formula`. Two common approaches are:
- Direct use of standard functions, e.g., `sigma_formula = ~ 1 + splines::ns(age, df = 3)`.
 - Use of a custom wrapper function with a single argument (the predictor), e.g., `sigma_formula = mypoly(age)`. Additional arguments can be set by modifying the function definition, for example: `mypoly <- poly; formals(mypoly)[['degree']] <- 3; mypoly(age)`. This approach can produce more concise coefficient names.
- sigma_formula_gr** Formula for the random effect parameter sigma (default NULL). See `a_formula_gr` for details on specifying grouping structures and correlation using vertical bar notation. As with `sigma_formula`, external functions (e.g., `poly`, `splines::ns`) may be included. See `sigma_formula` for further details.

`sigma_formula_gr_str`

Formula for the random effect parameter sigma, used when fitting hierarchical models with three or more levels. See `a_formula_gr_str` for details on specifying multi-level structures.

As with `sigma_formula`, external functions can be incorporated. For example: `sigma_formula_gr_str = (1 + splines::ns(age, df = 3) | i | id:study)`.

See `sigma_formula` for additional details.

`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 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 demonstrating how to use 'sigmavarfun' via the method sub-argument to specify each of the six variance models. To minimize the risk of typographical errors, we recommend using the shorthand form 'vf' instead of the full name 'sigmavarfun'.

All variance model specifications are provided through the `sigma_formula_manual` argument in `bsitar()`. In general, the syntax takes the form:

```
sigma_formula_manual = nlf(sigma ~ vf(...), method = ...) + lf(...)
```

Note that the predictor used within these variance functions should not be identical to `x` (see argument `x`, typically `age`). This is because the predictors for `mu` and `sigma` may require different transformations, which cannot be accommodated using a single shared variable.

Instead, define a separate predictor variable (even if it contains identical values). For example:

```
data$age_vf <- data$age
```

and then use `age_vf` within the variance function. This ensures that transformations applied to `sigma` do not interfere with those used for `mu`.

For `method = 're'`, outcome in the `nlf(sigma ~ vf())` is a placeholder that is internally replaced by the actual response variable such as `height` (See `y` for the response variables).

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(), outcome), 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 $\log(\text{abs}(\text{predictor}))$.
- 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, user 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 `SITARFun`, then the name for 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` 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 `sigmax` and `sigmarandom` argument. In other words, either `sigmax` or `sigmarandom` must include the form for the corresponding growth parameter defined in the `ls` function. For example, when either or both `sigmax` 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 subset of parameters are specified via the `sigmax` 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, or '0' depending on whether the `init = 'random'` or `init = '0'`.

<code>sigmax</code>	Predictor for the distributional parameter sigma. See <code>x</code> for details. Ignored if <code>sigma_formula_manual = NULL</code> .
<code>sigmaid</code>	A factor variable uniquely identifying the groups (e.g., individuals) in the data frame. If <code>NULL</code> (default), then <code>sigmaid</code> is set same as <code>id</code> . For univariate_by and multivariate models, the <code>sigmaid</code> can be the same (typically) for all sub-models, or different for each sub-model (see the <code>id</code> argument for details on setting different arguments for sub-models). Ignored if <code>sigma_formula_manual = NULL</code> .
<code>sigmadf</code>	Degree of freedom for the spline function used for sigma. See <code>df</code> for details. Ignored if <code>sigma_formula_manual = NULL</code> .
<code>sigmaknots</code>	Knots for the spline function used for sigma. See <code>knots</code> for details. Ignored if <code>sigma_formula_manual = NULL</code> .
<code>sigmaxfixed</code>	Fixed effect formula for the sigma structure. See <code>fixed</code> for details. Ignored if <code>sigma_formula_manual = NULL</code> .
<code>sigmarandom</code>	Random effect formula for the sigma structure. See <code>random</code> for details. Ignored if <code>sigma_formula_manual = NULL</code> . Currently not used even when <code>sigma_formula_manual</code> is specified.
<code>sigmaxoffset</code>	Offset for the <code>x</code> in the sigma structure. See <code>xoffset</code> for details. Ignored if <code>sigma_formula_manual = NULL</code> .
<code>sigmaxbstart</code>	Starting value for the <code>b</code> parameter in the sigma structure. See <code>bstart</code> for details. Ignored if <code>sigma_formula_manual = NULL</code> . Currently not used even when <code>sigma_formula_manual</code> is specified.
<code>sigmaxcstart</code>	Starting value for the <code>c</code> parameter in the sigma structure. See <code>cstart</code> for details. Ignored if <code>sigma_formula_manual = NULL</code> . Currently not used even when <code>sigma_formula_manual</code> is specified.
<code>sigmaxfun</code>	Transformation of <code>sigmax</code> variable for sigma structure. See <code>xfun</code> for details. Ignored if <code>sigma_formula_manual = NULL</code> .
<code>sigmaxbound</code>	Bounds for the <code>x</code> in the sigma structure. See <code>bound</code> for details. Ignored if <code>sigma_formula_manual = NULL</code> .
<code>sigmaxfunxoffset</code>	Transformation applied to <code>sigmaxoffset</code> for <code>sigmax</code> variable (default <code>TRUE</code>). See <code>sigmaxfun</code> for details. The default <code>sigmaxfunxoffset = TRUE</code> sets its value to same as <code>sigmaxfun</code> . Users rarely need to specify <code>sigmaxfunxoffset</code> themselves. One potential application is when a user wants to turn it off by setting <code>sigmaxfunxoffset = FALSE</code> .

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: • auto regressive moving average (arma) of order p and q, specified as autocor_formula = ~arma(p = 1, q = 1). • auto regressive (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 of residuals. Note that for unstr autocorrelation structure, time is a placeholder which could be any name. Internally, a new variable time, or time_response for multivariate or univariate_by models, is created from the predictor variable x e.g., age. This internal variable is required because the unstructured autocorrelation covariance matrix is indexed by time points treated as discrete, ordered indices. The underlying Stan implementation expects time to be represented as unique positive integers in increasing order (e.g., 1, 2, 3, ...) for each individual so that: • Each observation can be mapped to a unique row/column in the N_time x N_time unstructured covariance matrix, where N_time is the number of distinct time points per individual. • The autocorrelation structure is defined over a regular, ordered sequence of time indices, which is essential for constructing the covariance matrix in Stan. • Individuals with different original time scales (e.g., age in years, months, or days) can share a common autocorrelation structure, since time is standardized to integer indices within each individual. The predictor variable x is therefore transformed into this internal time variable such that it contains unique positive integers in increasing order, such as 1, 2, 3, for each individual identified by id. When the original x values do not satisfy this condition (e.g., they are non-integer ages like 0.5, 1.2, 3.7), the transformation automatically transforms them to meet this requirement during the creation of the internal time variable. No pre-processing is needed before using unstr autocorrelation.
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).

<code>custom_family</code>	Specifies custom families (i.e., response distribution). Default is NULL. For details, see <code>brms::custom_family()</code> . Note that user-defined Stan functions must be exposed by setting <code>expose_functions = TRUE</code> .
<code>custom_formula</code>	Specifies custom formula. Default is NULL. For details, see <code>brms::brmsformula()</code> . Currently ignored.
<code>custom_prior</code>	Specifies custom prior. Default is NULL. For details, see <code>brms::prior()</code> . Currently ignored. It is primarily designed to support setting custom prior for <code>custom_formula</code> . Note that <code>custom_prior</code> is different from the <code>set_self_priors</code> which is evaluated throughout the call.
<code>custom_stanvars</code>	Allows the passing of user-defined variables to be added to Stan's program blocks (default NULL). This is primarily useful when defining a <code>custom_family</code> . For more details on specifying <code>stanvars</code> , see <code>brms::custom_family()</code> . Note that <code>custom_stanvars</code> are passed directly without conducting any sanity checks.
<code>group_arg</code>	Specify the grouping variable for group-level random effects. The <code>group_arg</code> should be a named list that may include <code>groupvar</code>, <code>dist</code>, <code>cor</code>, and <code>by</code>, as described below: • <code>groupvar</code> specifies the grouping (subject) identifier. If NULL (default), it is automatically inferred from the <code>id</code> argument. • <code>dist</code> specifies the distribution of the random effects (default <code>gaussian</code>). Currently, only the Gaussian distribution is supported (see <code>[brms::brm()]</code>). • <code>by</code> allows estimation of separate variance-covariance structures (i.e., standard deviations and correlations) across levels of a factor variable (default NULL). The <code>by</code> variable must be a factor. For example, <code>by = "sex"</code> estimates separate variance-covariance structures for males and females. • <code>cor</code> specifies the covariance (correlation) structure of the random effects. The default is unstructured (<code>cor = un</code>) for all model types (univariate, <code>univariate_by</code>, and <code>multivariate</code>). For univariate and <code>univariate_by</code> models, the alternative option is <code>diagonal</code>, which estimates only variance parameters while fixing correlations to zero. For multivariate models, available options are: – <code>un</code>: A fully unstructured covariance, where random effects across outcomes are jointly modeled with shared correlations. – <code>diagonal</code>: Variances are estimated separately for each outcome, with correlations fixed to zero. – <code>un_s</code>: Separate unstructured variance-covariance matrices are estimated for each outcome.

All components of `group_arg` are optional and default values are used when unspecified.

For `univariate_by` and `multivariate` models, only `groupvar` from `group_arg` is used, as additional model-specific options are handled via the corresponding arguments.

The `group_arg` argument is ignored when random effects are specified using vertical bar notation (see `a_formula_gr`) or when fitting hierarchical models with three or more levels (see `a_formula_gr_str`).

<code>sigma_group_arg</code>	Specify arguments for modeling distributional-level random effects for sigma. The setup for <code>sigma_group_arg</code> follows the same approach as described for group-level random effects (see <code>group_arg</code> for details).
<code>univariate_by</code>	Set up the univariate-by-subgroup model fitting (default NULL) via a named list with the following elements: • <code>by</code> (optional, character string): Specifies the factor variable used to define the sub-models (default NA). • <code>cor</code> (optional, character string): Defines the correlation structure. Options include <code>un</code> (default) for a full unstructured variance-covariance structure and <code>diagonal</code> for a structure with only variance parameters (i.e., standard deviations) and no covariance (i.e., correlations set to zero). • <code>terms</code> (optional, character string): Specifies the method for setting up the sub-models. Options are <code>'subset'</code> (default) and <code>'weights'</code>. See <code>brms::`addition-terms`</code> for more details.
<code>multivariate</code>	Set up the multivariate model fitting (default NULL) using a named list with the following elements: • <code>mvar</code> (logical, default FALSE): Indicates whether to fit a multivariate model. • <code>cor</code> (optional, character string): Specifies the correlation structure for group-level random effects. Available options are: – <code>"un"</code> (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. – <code>"diagonal"</code>: Estimates only the variance parameters for each sub-model, with the correlation parameters set to zero. – <code>"un_s"</code>: Estimates unstructured variance-covariance parameters separately for each response variable. • <code>rescor</code> (logical, default TRUE): Indicates whether to estimate the residual correlation between response variables. • <code>rcorr_by</code> (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 <code>rescor = FALSE</code>. • <code>rcorr_gr</code> (character string, default NULL): The grouping variable (typically a level 2 variable like <code>id</code>) for which residual correlations between response variables are estimated. This must be a factor variable present in the data. Ignored when <code>rescor = FALSE</code>. • <code>rcorr_method</code> (character string, default NULL): Specifies the method for estimating residual correlations: <code>"lkj"</code> (uses an LKJ distribution) or <code>"cde"</code> (uses the Cholesky decomposition of the covariance matrix with a uniform prior of -1, 1 for each off-diagonal element). If NULL, <code>"lkj"</code> is set as the default. Ignored when <code>rescor = FALSE</code>. • <code>rcorr_prior</code> (numeric vector, default NULL): Used to specify the LKJ prior for each level of <code>rcorr_by</code> when <code>rcorr_method = "lkj"</code>. The length of <code>rcorr_prior</code> should be either one (to use the same prior for all levels) or equal to the number of levels in <code>rcorr_by</code>. For <code>rcorr_method = 'cde'</code>, uniform priors -1, 1 are assigned to all correlation parameters. The argument <code>rcorr_prior</code> is ignored when <code>rescor = FALSE</code>.

a_prior_beta	Prior specification for the fixed effect parameter a (default: <code>normal(ymean, ysd, autoscale = FALSE)</code>). A robust alternative to the default setting is to use <code>lm</code> rather than <code>ymean</code> . This is because even when the predictor <code>xvar</code> is mean-centered, the intercept parameter a does not always correspond exactly to the overall sample mean. The <code>@details</code> section provides full details on prior specifications for the bsitar package.
b_prior_beta	Prior specification for the fixed effect parameter b (default: <code>normal(0, 2.0, autoscale = FALSE)</code>). The <code>@details</code> section provides full details on prior specifications for the bsitar package.
c_prior_beta	Prior specification for the fixed effect parameter c (default: <code>normal(0, 1.0, autoscale = FALSE)</code>). The <code>@details</code> section provides full details on prior specifications for the bsitar package.
d_prior_beta	Prior specification for the fixed effect parameter d (default: <code>normal(0, 1.0, autoscale = FALSE)</code>). For full details on prior specifications, please refer to <code>a_prior_beta</code> . <i>Note:</i> To set the scale for location-scale priors, users can specify the standard deviation (<code>xsd</code>) or the median absolute deviation (<code>xmad</code>) of the predictor variable <code>x</code> .
s_prior_beta	Prior specification for the fixed effect parameter s (spline coefficients) (default: <code>normal(lm, lm, autoscale = FALSE)</code>). For general guidelines on prior specifications, please refer to <code>a_prior_beta</code>. Key points specific to <code>s</code> include: • Data-driven lm priors: When using <code>lm</code> for location-scale priors (e.g., <code>normal(lm, lm)</code>), the location parameter is derived from spline coefficients obtained via a simple linear model fit. The scale parameter is derived from the standard deviation of the spline design matrix. • Hierarchical priors (<code>sethp</code>): The <code>sethp</code> argument (default: <code>FALSE</code>) enables hierarchical prior specifications. Setting <code>sethp = TRUE</code> modifies the prior from $s \sim \text{normal}(0, \text{lm})$ to $s \sim \text{normal}(0, \text{'hp'})$, where $\text{hp} \sim \text{normal}(0, \text{lm})$. The scale for the hierarchical prior is inherited from <code>s</code>, but user can also explicitly pass a distribution. For example, <code>normal(0, lm, sethp = cauchy)</code> translates to $s \sim \text{normal}(0, \text{lm})$ and $\text{hp} \sim \text{cauchy}(0, \text{lm})$. • Uniform distributions: The <code>addrange</code> argument can be used to symmetrically expand the prior range. Scale Parameter Variants for <code>lm</code> When the scale parameter is set to <code>lm</code>, it defaults to <code>lm1</code>. The available variants calculate the scale differently (where <code>y</code> is the outcome, <code>X</code> is the design matrix, and <code>lm_se</code> is the standard error vector from the linear model fit): • <code>lm</code> or <code>lm1</code>: $\text{sd}(y) / \text{sd}(X)$ • <code>lm2</code>: $\text{lm_se} / \text{sd}(X)$ • <code>lm3</code>: <code>lm_se</code> <i>Note:</i> Location-scale priors (such as <code>normal</code>, <code>student_t</code>, and <code>cauchy</code>) generally perform well for spline coefficients. In most cases, the default <code>normal(lm, lm)</code> is highly effective and can be further refined by setting <code>autoscale = TRUE</code> depending on the specific data structure.
a_cov_prior_beta	Prior specification for covariate effects included in the fixed effect parameter a (default: <code>normal(0, 5.0, autoscale = FALSE)</code>). The general approach to prior specification is similar to <code>a_prior_beta</code> , with the following differences:

- **Restricted data-driven options:** The options 'ymean', 'ymedian', 'ysd', and 'ymad' are not available for `a_cov_prior_beta`.
- **lm for location:** The `lm` option can be used for the location parameter, allowing covariate coefficients to be derived from a simple linear model fitted to the data. This option is available only for `a_cov_prior_beta` and not for covariates in other fixed or random effect parameters.
- **Submodel-specific priors:** Separate priors can be specified for individual sub models when fitting `univariate_by` or `multivariate` models; see `a_prior_beta` for details.

`b_cov_prior_beta`

Prior specification for covariate effects 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`

Prior specification for covariate effects 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`

Prior specification for covariate effects 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`

Prior specification for covariate effects included in the fixed effect parameter `splines` (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`

Prior specification 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 sub model (see `a_prior_beta`).

`b_prior_sd`

Prior specification for the random effect parameter, `b`. (default: `normal(0, 2.0, autoscale = FALSE)`). See `a_prior_sd` for details.

`c_prior_sd`

Prior specification for the random effect parameter, `c`. (default: `normal(0, 1.0, autoscale = FALSE)`). See `a_prior_sd` for details.

`d_prior_sd`

Prior specification for the random effect parameter, `d`. (default: `normal(0, 1.0, autoscale = FALSE)`). See `a_prior_sd` for details.

`a_cov_prior_sd`

Prior specification for covariate effects 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** Prior specification for covariate effects 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** Prior specification for covariate effects 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** Prior specification for covariate effects 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** Prior specification for the random effect parameter *a* when fitting a hierarchical model with three or more levels of hierarchy. (default `NULL`). See `a_prior_sd` for details on setting priors when needed.
- b_prior_sd_str** Prior specification for the random effect parameter *b* when fitting a hierarchical model with three or more levels of hierarchy. (default `NULL`). See `b_prior_sd` for details on setting priors when needed.
- c_prior_sd_str** Prior specification for the random effect parameter *c* when fitting a hierarchical model with three or more levels of hierarchy. (default `NULL`). See `c_prior_sd` for details on setting priors when needed.
- d_prior_sd_str** Prior specification for the random effect parameter *d* when fitting a hierarchical model with three or more levels of hierarchy (default `NULL`). See `a_prior_sd` for details on setting priors when needed.
- a_cov_prior_sd_str**
Prior specification for covariate effects in the random effect parameter *a* when fitting a hierarchical model with three or more levels of hierarchy (default `NULL`). The approach follows `a_cov_prior_sd`.
- b_cov_prior_sd_str**
Prior specification for covariate effects in the random effect parameter *b* when fitting a hierarchical model with three or more levels of hierarchy (default `NULL`). The approach follows `a_cov_prior_sd_str`.
- c_cov_prior_sd_str**
Prior specification for covariate effects in the random effect parameter *c* when fitting a hierarchical model with three or more levels of hierarchy (default `NULL`). The approach follows `a_cov_prior_sd_str`.
- d_cov_prior_sd_str**
Prior specification for covariate effects in the random effect parameter *d* when fitting a hierarchical model with three or more levels of hierarchy (default `NULL`). The approach follows `a_cov_prior_sd_str`.
- sigma_prior_beta**
Prior specification for the fixed effect distributional parameter *sigma* (default `normal(0, 1.0, autoscale = FALSE)`). The general approach mirrors `a_prior_beta`.
- sigma_cov_prior_beta**
Prior specification for covariate effects in the fixed effect distributional parameter *sigma* (default `normal(0, 0.5, autoscale = FALSE)`). The general approach mirrors `a_cov_prior_beta`.
- sigma_prior_sd** Prior specification for the random effect distributional parameter *sigma* (default `normal(0, 0.25, autoscale = FALSE)`). The general approach mirrors `a_prior_sd`.

<code>sigma_cov_prior_sd</code>	Prior specification for covariate effects in the random effect distributional parameter <code>sigma</code> (default <code>normal(0, 0.15, autoscale = FALSE)</code>). The general approach mirrors <code>a_cov_prior_sd</code> .
<code>sigma_prior_sd_str</code>	Prior specification for the random effect distributional parameter <code>sigma</code> when fitting a hierarchical model with three or more levels of hierarchy (default <code>NULL</code>). The general approach mirrors <code>a_prior_sd_str</code> .
<code>sigma_cov_prior_sd_str</code>	Prior specification for covariate effects in the random effect distributional parameter <code>sigma</code> when fitting a hierarchical model with three or more levels of hierarchy (default <code>NULL</code>). The general approach mirrors <code>a_cov_prior_sd_str</code> .
<code>rsd_prior_sigma</code>	Prior specification for the residual standard deviation parameter <code>sigma</code> (default <code>normal(0, 'ysd', autoscale = FALSE)</code>). Evaluated when both <code>dpar_formula</code> and <code>sigma_formula</code> are <code>NULL</code> . For location-scale distributions, the scale parameter can be set to the standard deviation (<code>ysd</code>) or median absolute deviation (<code>ymad</code>) of the outcome, or to the residual standard deviation from a linear mixed model (<code>'lme_rsd'</code>) or linear model (<code>'lm_rsd'</code>). If <code>nlme::lme()</code> fails to converge, <code>'lm_rsd'</code> is used automatically.
<code>dpar_prior_sigma</code>	Prior specification for the fixed effect distributional parameter <code>sigma</code> when using <code>dpar_formula</code> (default <code>normal(0, 'ysd', autoscale = FALSE)</code>). Evaluated when <code>sigma_formula</code> is <code>NULL</code> . See <code>rsd_prior_sigma</code> for details.
<code>dpar_cov_prior_sigma</code>	Prior specification for covariate effects in the fixed effect distributional parameter <code>sigma</code> when using <code>dpar_formula</code> (default <code>normal(0, 1.0, autoscale = FALSE)</code>). Evaluated when <code>sigma_formula</code> is <code>NULL</code> .
<code>autocor_prior_acor</code>	Prior specification for autocorrelation parameters when fitting models with "arma", "ar", or "ma" autocorrelation structures (see <code>autocor_formula</code>). The only allowed distribution is uniform, bounded between -1 and 1 (default <code>uniform(-1, 1, autoscale = FALSE)</code>). For unstructured residual correlation structures, use <code>autocor_prior_unstr_acor</code> .
<code>autocor_prior_unstr_acor</code>	Prior specification for autocorrelation parameters when fitting models with the unstructured ("un") autocorrelation structure (see <code>autocor_formula</code>). The only allowed distribution is <code>lkj</code> (default <code>lkj(1)</code>). See <code>gr_prior_cor</code> for details on the <code>lkj</code> prior.
<code>gr_prior_cor</code>	Prior specification for correlation parameters of group-level random effects (default <code>lkj(1)</code>). The only allowed distribution is <code>lkj</code> , specified via a single parameter <code>eta</code> .
<code>gr_prior_cor_str</code>	Prior specification for correlation parameters of group-level random effects when fitting a hierarchical model with three or more levels of hierarchy (default <code>lkj(1)</code>). The general approach mirrors <code>gr_prior_cor</code> .

<code>sigma_prior_cor</code>	Prior specification for correlation parameters of distributional random effects sigma (default <code>lkj(1)</code>). The only allowed distribution is <code>lkj</code> (see <code>gr_prior_cor</code>). Note that <code>brms::brm()</code> does not currently allow different <code>lkj</code> priors for group-level and distributional random effects sharing the same group identifier.
<code>sigma_prior_cor_str</code>	Prior specification for correlation parameters of distributional random effects sigma when fitting a hierarchical model with three or more levels of hierarchy (default <code>lkj(1)</code>). The general approach mirrors <code>sigma_prior_cor</code> .
<code>mvr_prior_rescor</code>	Prior specification for the residual correlation matrix in multivariate models (default <code>lkj(1)</code>). The only allowed distribution is <code>lkj</code> (see <code>gr_prior_cor</code> for details).
<code>init</code>	Initial values for the sampler. Options include: • "random" (default): Stan randomly generates initial values for each parameter within a range defined by <code>init_r</code> (see below), or between -2 and 2 on the unconstrained scale if <code>init_r = NULL</code>. • "0": All parameters are initialized to zero. • "prior": Parameters are initialized from their specified priors. • NULL: Initial values are taken from the parameter-specific <code>init</code> arguments defined below. When <code>init = NULL</code>, fixed effects and variance-covariance parameters (with <code>vcov_init_0 = FALSE</code>) are initialized according to their individual settings. To force all parameters to use random initial values, set <code>init = "random"</code>, which is internally translated to <code>init = NULL</code> with fully random initialization.
<code>init_r</code>	Positive real value specifying the range for random initial values (default 0.5). This argument is used only when <code>init = "random"</code> . Note that the default in Stan is 2.0, which generates random initials between -2.0 and 2.0 on the unconstrained parameter space.
<code>a_init_beta</code>	Initial specification 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 coefficients from a simple linear model fitted to the data. Note that "ymean", "ymedian", and "lm" are available only for the fixed effect parameter a. For univariate_by and multivariate models, initial values can be the same across sub-models (e.g., <code>a_init_beta = "0"</code>) or different for each sub-model (e.g., <code>a_init_beta = list("0", "lm")</code>).
<code>b_init_beta</code>	Initial specification for the fixed effect parameter b (default "random"). See <code>a_init_beta</code> for details on available options.

c_init_beta	Initial specification for the fixed effect parameter c (default "random"). See a_init_beta for details on available options.
d_init_beta	Initial specification for the fixed effect parameter d (default "random"). See a_init_beta for details on available options.
s_init_beta	Initial specification 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 coefficients from a simple linear model fitted to the data.
a_cov_init_beta	Initial specification for covariate effects in the fixed effect parameter a (default "random"). Available options include: • "0": Initializes covariate effects to zero. • "random": Initializes with random values within a specified range. • "prior": Uses values drawn from the prior distribution. • "lm": Initializes with coefficients from a simple linear model fitted to the data. Note that "lm" is available only for a_cov_init_beta and not for covariates in other fixed effect parameters such as b, c, or d.
b_cov_init_beta	Initial specification for covariate effects in the fixed effect parameter b (default "random"). See a_cov_init_beta for details.
c_cov_init_beta	Initial specification for covariate effects in the fixed effect parameter c (default "random"). See a_cov_init_beta for details.
d_cov_init_beta	Initial specification for covariate effects in the fixed effect parameter d (default "random"). See a_cov_init_beta for details.
s_cov_init_beta	Initial specification for covariate effects in the fixed effect parameter s (default "lm"). See a_cov_init_beta for details. The option "lm" corresponds to spline coefficients obtained from a simple linear model fitted to the data. However, s_cov_init_beta acts only as a placeholder and is not evaluated, because covariates are not allowed for s; see s_formula for details.
a_init_sd	Initial specification 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": Uses the standard deviation (sd) of the response variable as the initial value. • "ymad": Uses the median absolute deviation (mad) of the response variable as the initial value.

- "lme_sd_a": Uses the standard deviation of the random intercept from a linear mixed model (`nlme::lme()`) fitted to the data. If `nlme::lme()` fails to converge, "lm_sd_a" is used automatically.
- "lm_sd_a": Uses the square root of the residual variance from a simple linear model fitted to the data.

Note that "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.

When fitting univariate_by and multivariate models, initial values can be shared across all sub-models or specified separately for each sub-model.

b_init_sd	Initial specification for the standard deviation of the group-level random effect parameter b (default "random"). See a_init_sd for options and details.
c_init_sd	Initial specification for the standard deviation of the group-level random effect parameter c (default "random"). See a_init_sd for options and details.
d_init_sd	Initial specification for the standard deviation of the group-level random effect parameter d (default "random"). See a_init_sd for options and details.
a_cov_init_sd	Initial specification for covariate effects in the group-level random effect parameter a (default "random"). Available options include: • "random": Random initialization. • "prior": Uses values drawn from the prior distribution.
b_cov_init_sd	Initial specification for covariate effects in the group-level random effect parameter b (default "random"). See a_cov_init_sd for options and details.
c_cov_init_sd	Initial specification for covariate effects in the group-level random effect parameter c (default "random"). See a_cov_init_sd for options and details.
d_cov_init_sd	Initial specification for covariate effects in the group-level random effect parameter d (default "random"). See a_cov_init_sd for options and details.
sigma_init_beta	Initial specification for the fixed effect distributional parameter sigma (default "random"). Available options include: • "random": Random initialization. • "prior": Uses values drawn from the prior distribution.
sigma_cov_init_beta	Initial specification for covariate effects in the fixed effect distributional parameter sigma (default "random"). See sigma_init_beta for options and details.
sigma_init_sd	Initial specification for the standard deviation of the distributional random effect parameter sigma (default "random"). The approach is the same as for group-level random effects such as a (see a_init_sd for details).
sigma_cov_init_sd	Initial specification for covariate effects in the distributional random effect parameter sigma (default "random"). The approach mirrors a_cov_init_sd.
gr_init_cor	Initial specification for the correlation parameters of group-level random effects (default "random"). Allowed options are: • "random": Random initialization. • "prior": Uses values drawn from the prior distribution.

- Numeric vector: A vector of length equal to the number of distinct correlations (lower-triangular elements). For example, for three random effect parameters, one may use `gr_init_cor = list(c(0.5, 0.5, 0.5))`.

Note that when `vcov_init_0 = TRUE`, `gr_init_cor` is set to `"0"`.

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

- `"random"`: Random initialization.
- `"prior"`: Uses values drawn from the prior distribution.

`rsd_init_sigma` Initial specification for the residual standard deviation parameter sigma (default `"random"`). Available options are:

- `"0"`: Initializes the residual standard deviation to zero.
- `"random"`: Random initialization of the residual standard deviation.
- `"prior"`: Uses values drawn from the prior distribution.
- `"lme_rsd"`: Uses the residual standard deviation from a linear mixed model (`nlme::lme()`) fitted to the data.
- `"lm_rsd"`: Uses the square root of the residual variance from a simple linear model fitted to the data.

If `nlme::lme()` fails to converge, `"lm_rsd"` is used automatically. `rsd_init_sigma` is evaluated only when both `dpar_formula` and `sigma_formula` are `NULL`.

`dpar_init_sigma`

Initial specification for the distributional parameter sigma when using `dpar_formula` (default `"random"`). The approach and options mirror `rsd_init_sigma`. This argument is evaluated only when `dpar_formula` is not `NULL`.

`dpar_cov_init_sigma`

Initial specification for covariate effects in the distributional parameter sigma (default `"random"`). Allowed options are `"0"`, `"random"`, and `"prior"`.

`autocor_init_acor`

Initial specification for autocorrelation parameters (see `autocor_formula` for details; default `"random"`). Allowed options are `"0"`, `"random"`, and `"prior"`.

`autocor_init_unstr_acor`

Initial specification for unstructured residual autocorrelation parameters (default `"random"`). Allowed options are `"0"`, `"random"`, and `"prior"`. The approach mirrors `gr_init_cor`.

`mvr_init_rescor`

Initial specification for the residual correlation matrix in multivariate models (default `"random"`). Allowed options are `"0"`, `"random"`, and `"prior"`.

`r_init_z`

Initial specification for standardized group-level random effect parameters (default `"random"`). These parameters arise from the non-centered parameterization used internally by `brms::brm()`.

`vcov_init_0`

Logical indicator controlling initial values for variance (standard deviation) and covariance (correlation) parameters. When `vcov_init_0 = TRUE`, all variance–covariance parameters are initialized to zero, allowing custom initial specifications for fixed effects while keeping variance–covariance parameters at zero. When `vcov_init_0 = FALSE` (default), variance–covariance parameters are assigned random initial

values unless a parameter-specific initial specification is provided (e.g., `a_init_sd = "0"`). This setting is ignored when global initial values are assigned via the `init` argument.

`jitter_init_beta`

Named list or numeric value used to add a small amount of noise to initial values for population-level (fixed effect) parameters.

When specified as a numeric value, `jitter_init_beta` is interpreted as a percentage and must lie between 0 and 100. Internally, this percentage is converted to a proportion (percentage / 100) and passed as the `amount` argument to `base::jitter()`, with `factor` fixed at its default 1.

The default `jitter_init_beta = NULL` means no perturbation is applied, so identical initial values are used across chains. For mild perturbation, a value such as `jitter_init_beta = 10` is often adequate.

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 = 10` yields perturbed values in the range 90–110, whereas an initial value of 10 yields values in the range 9–11.

When `jitter_init_beta` is provided as a named list, its elements are passed directly to `base::jitter()`. In addition to `factor` and `amount`, a `percent` argument may be supplied and is treated in the same way as a single numeric value. If both `percent` and `factor` are supplied, the effective perturbation is their product.

To use the defaults of `base::jitter()`, supply an empty list `list()`, which is then completed internally with `factor = 1`, `amount = NULL`. See `base::jitter()` for further details on `factor` and `amount`.

`jitter_init_sd` Named list or numeric value used to add a small amount of noise to initial values for the standard deviations of random-effect parameters. A pragmatic choice that has worked well in early testing is `jitter_init_sd = 1`, corresponding to a 1% perturbation. See `jitter_init_beta` for details on the available options and how perturbations are applied.

`jitter_init_cor`

Named list or numeric value used to add a small amount of noise to initial values for the correlations of random-effect parameters. A pragmatic choice that has worked well in early testing is `jitter_init_cor = 0.1`, corresponding to a 0.1 (10%) perturbation. See `jitter_init_beta` for details on available options and how perturbations are applied.

`prior_data`

Optional named list (default `NULL`) used to pass external objects into prior specifications for individual parameters (e.g., `a_prior_beta`). This is particularly useful when supplying long vectors or matrices as priors, which can be created in the R environment and then referenced via `prior_data`. For example, to pass vectors of location and scale parameters for 10 dummy-variable coefficients in the fixed effect parameter `a`, one can:

- Create named objects in R: `prior_a_cov_location <- rnorm(10, mean = 0, sd = 1)` `prior_a_cov_scale <- rep(5, 10)`
- Add them to `prior_data`: `prior_data = list(prior_a_cov_location = prior_a_cov_location, prior_a_cov_scale = prior_a_cov_scale)`

	• Use these objects in the prior specification: <code>a_cov_prior_beta = normal(prior_a_cov_location, prior_a_cov_scale)</code>
<code>init_data</code>	Optional named list (default NULL) used to pass external objects into initialization arguments. The handling of <code>init_data</code> is identical to <code>prior_data</code> (see above).
<code>init_custom</code>	Custom initialization object supplied as a named list. The list is passed directly to the underlying <code>init</code> argument without checking dimensions or name matching. If some parameters already have parameter-specific initial specifications (e.g., <code>a_init_beta = "0"</code>), <code>init_custom</code> is applied only to parameters without their own initial settings. To override this behavior and use <code>init_custom</code> for all parameters regardless of parameter-specific initials, set <code>init = "custom"</code> .
<code>expose_function</code>	Logical (default FALSE) indicating whether to expose Stan functions used in model fitting.
<code>get_stancode</code>	Logical (default FALSE) indicating whether to return the Stan code (see <code>brms::stancode()</code> for details).
<code>get_standata</code>	Logical (default FALSE) indicating whether to return the Stan data (see <code>brms::standata()</code> for details).
<code>get_formula</code>	Logical (default FALSE) indicating whether to return the model formula (see <code>brms::brmsformula()</code> for details).
<code>get_stanvars</code>	Logical (default FALSE) indicating whether to return the Stan variables (see <code>brms::stanvar()</code> for details).
<code>get_priors</code>	Logical or character (default FALSE) controlling prior retrieval (see <code>brms::get_prior()</code> for details). If <code>get_priors = TRUE</code> , priors corresponding to the final compiled model code are returned. To obtain basic default priors instead, use <code>get_priors = "default"</code> .
<code>get_priors_eval</code>	Logical (default FALSE) indicating whether to return the priors as specified by the user.
<code>get_init_eval</code>	Logical (default FALSE) indicating whether to return the initial values as specified by the user.
<code>validate_priors</code>	Logical (default FALSE) indicating whether to validate specified priors (see <code>brms::validate_prior()</code> for details).
<code>set_self_priors</code>	Optional object (default NULL) used to manually specify priors. <code>set_self_priors</code> is passed directly to <code>brms::brm()</code> without additional checks.
<code>add_self_priors</code>	Optional object (default NULL) used to append elements to an existing prior object. Intended for internal use only.
<code>set_replace_priors</code>	Optional object (default NULL) used to replace elements of an existing prior object. Intended for internal use only.
<code>set_same_priors_hierarchy</code>	Optional object (default NULL) used to enforce the same prior structure across hierarchical levels. Intended for internal use only.

outliers	Optional specification (default NULL) for removing outliers. This should be a named list passed directly to <code>sitar::velout()</code> and <code>sitar::zapvelout()</code> . Intended for internal use only.
unused	Optional formula defining variables that are unused in the model but should still be stored in the model frame, for example when they are required for post-processing.
chains	Number of Markov chains (default 4).
iter	Total number of iterations per chain, including warmup (default 2000).
warmup	Positive integer specifying the number of warmup (burn-in) iterations. Warmup iterations are also used for step-size adaptation and should not be used for inference. The number of warmup iterations must not exceed <code>iter</code> , and the default is <code>iter / 2</code> .
thin	Positive integer specifying the thinning interval. Set <code>thin > 1</code> to reduce memory usage and computation time when <code>iter</code> is large. Thinning is sometimes used when MCMC draws show high autocorrelation, evidenced by poor chain mixing (e.g., high <code>rhat</code>) despite good parameter recovery. Autocorrelation can be inspected using functions such as <code>mcmc_acf()</code> from the bayesplot package.
cores	Number of cores used to execute chains in parallel. See <code>brms::brm()</code> for further details. Unlike <code>brms::brm()</code>, which defaults <code>cores</code> to <code>getOption("mc.cores", 1)</code>, bsitar defaults <code>cores</code> to <code>getOption("mc.cores", "optimize")</code>, which attempts to optimize system resource usage. In this mode, the maximum number of cores used is the number of available cores minus one. If the available cores exceed the number of chains (see <code>chains</code>), <code>cores</code> is set equal to the number of chains. Alternatively, <code>cores</code> may be set to <code>getOption("mc.cores", "maximise")</code>, which uses all available cores regardless of the number of chains, or specified as in <code>brms::brm()</code> via <code>getOption("mc.cores", 1)</code>. This behavior can be set globally using <code>options(mc.cores = x)</code>, where <code>x</code> is one of "optimize", "maximise", or 1. The <code>cores</code> argument can also be set directly as an integer (e.g., <code>cores = 4</code>).
backend	Character string specifying the interface used to execute the Stan model. Available options are "rstan" (default) and "cmdstanr". The backend can also be set globally for the current R session via the "brms.backend" option. See <code>brms::brm()</code> for more details.
threads	Number of threads used for within-chain parallelization. Unlike <code>brms::brm()</code>, which defaults <code>threads</code> to <code>getOption("brms.threads", NULL)</code> (no within-chain parallelization), bsitar defaults <code>threads</code> to <code>getOption("brms.threads", "optimize")</code> to better utilize modern multi-core systems. In this mode, the number of threads per chain is set to the number of available cores minus one. An alternative is <code>getOption("brms.threads", "maximise")</code>, which sets the number of threads per chain equal to the total number of available cores. Users can also emulate the brms default via <code>getOption("brms.threads", NULL)</code>. These behaviors can be set globally using <code>options(brms.threads = x)</code>, where <code>x</code> is one of "optimize", "maximise", or NULL. Alternatively, <code>threads</code> can be specified directly using <code>threads = threading(x)</code>, where <code>x</code> is an integer. Additional arguments, such as <code>grainsize</code> and <code>static</code>,

	can also be passed via <code>threading()</code> . See <code>brms::brm()</code> for further details on within-chain parallelization.
<code>opengl</code>	Numeric vector of length two specifying the platform and device IDs for OpenCL-based GPU support during model fitting. If unsure, <code>c(0, 0)</code> usually corresponds to the default device. See <code>brms::opengl()</code> for guidance on identifying valid platform and device IDs. This parameter can also be set globally via the <code>"brms.opengl"</code> option.
<code>normalize</code>	Logical indicating whether normalization constants should be included in the Stan code (default TRUE). If FALSE, normalization constants are omitted, which may improve sampling efficiency, but requires Stan version 2.25 or higher. Note that <code>normalize = FALSE</code> disables some post-processing functionality (e.g., <code>brms::bridge_sampler()</code>). This behavior can be controlled globally via the <code>"brms.normalize"</code> option.
<code>algorithm</code>	Character string specifying the estimation method. Available options are: • "sampling" (default): MCMC sampling. • "meanfield": Variational inference with independent normal approximations. • "fullrank": Variational inference with a full-rank multivariate normal approximation. • "fixed_param": Sampling with parameters fixed (no adaptation). This argument can be set globally via the <code>"brms.algorithm"</code> option (see options).
<code>control</code>	Named list of sampler control parameters. Defaults are as in <code>brms::brm()</code> , except that <code>max_treedepth</code> is increased from 10 to 12 to better explore challenging posterior geometries common in nonlinear models. In contrast, <code>adapt_delta</code> retains the default value of 0.8 to avoid unnecessarily long sampling times. See <code>brms::brm()</code> for a complete description of available control parameters.
<code>empty</code>	Logical. If TRUE, the Stan model is not created and compiled and the corresponding 'fit' slot of the <code>brmsfit</code> object will be empty. This is useful if you have estimated a <code>brms</code> -created Stan model outside of brms and want to feed it back into the package.
<code>rename</code>	For internal use only.
<code>pathfinder_args</code>	Named list of arguments passed to the "pathfinder" algorithm, used to obtain pathfinder-based initial values for subsequent MCMC sampling. Currently, <code>pathfinder_args</code> is supported only when <code>backend = "cmdstanr"</code> . If <code>pathfinder_args</code> is not NULL and <code>backend = "rstan"</code> is requested, the backend is automatically switched to "cmdstanr".
<code>pathfinder_init</code>	Logical (default FALSE) indicating whether to use initial values obtained from the "pathfinder" algorithm when fitting the final model via MCMC sampling. <code>pathfinder_args</code> is currently supported only when <code>backend = "cmdstanr"</code> . If <code>pathfinder_args</code> is not NULL and <code>backend = "rstan"</code> is requested, the backend is automatically switched to "cmdstanr". The arguments passed to the "pathfinder" algorithm are supplied via <code>pathfinder_args</code> ; if <code>pathfinder_args</code> is NULL, the default settings from cmdstanr are used.

<code>data2</code>	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.
<code>data_custom</code>	A <code>data.frame</code> object (default <code>NULL</code>). This is mainly for internal testing and not be used for routine model fitting.
<code>genquant_xyadj</code>	A logical (default <code>NULL</code>) indicating whether to generate <code>xyadj</code> in the generated quantities block of the Stan.
<code>sample_prior</code>	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 <code>brms::hypothesis()</code> . Note that improper priors (including the default improper priors used by <code>brm</code>) are not sampled. For proper priors, see <code>brms::set_prior()</code> . Also, prior draws for the overall intercept are not obtained by default for technical reasons. See <code>brms::brmsformula()</code> for instructions on obtaining prior draws for the intercept. If <code>sample_prior</code> is set to "only", draws will be taken solely from the priors, ignoring the likelihood, which allows user to generate draws from the prior predictive distribution. In this case, all parameters must have proper priors.
<code>save_pars</code>	An object generated by <code>brms::save_pars()</code> 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.
<code>drop_unused_levels</code>	A logical value indicating whether unused factor levels in the data should be dropped. The default is <code>TRUE</code> .
<code>stan_model_args</code>	A list of additional arguments passed to <code>rstan::stan_model</code> when using the <code>backend = "rstan"</code> or <code>backend = "cmdstanr"</code> . This allows customization of how models are compiled.
<code>refresh</code>	An integer specifying the frequency of printing every <code>nth</code> iteration. By default, <code>NULL</code> indicates that the refresh rate will be automatically set by <code>brms::brm()</code> . Setting <code>refresh</code> is especially useful when <code>thin</code> is greater than 1, in which case the refresh rate is recalculated as $(\text{refresh} * \text{thin}) / \text{thin}$.
<code>silent</code>	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 <code>refresh = 0</code> . Additionally, when using <code>backend = "rstan"</code> , The user can prevent the opening of additional progress bars by setting <code>open_progress = FALSE</code> .
<code>seed</code>	An integer or <code>NA</code> (default) specifying the seed for random number generation, ensuring reproducibility of results. If set to <code>NA</code> , Stan will randomly select the seed.
<code>save_model</code>	A character string or <code>NULL</code> (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 <code>save_model</code> .

<code>fit</code>	An instance of class <code>brmsfit</code> from a previous fit (default is NA). If a <code>brmsfit</code> 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 <code>fit</code> argument.
<code>file</code>	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 <code>file</code> . The <code>.rds</code> 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, user must manually remove the file or specify the <code>file_refit</code> argument. The file name is stored within the <code>brmsfit</code> object for later use.
<code>file_compress</code>	Logical or a character string, specifying one of the compression algorithms supported by saveRDS . If the <code>file</code> argument is provided, this compression will be used when saving the fitted model object.
<code>file_refit</code>	Modifies when the fit stored via the <code>file</code> argument is re-used. This can be set globally for the current R session via the <code>"brms.file_refit"</code> option (see options). The possible options are: • <code>"never"</code> (default): The fit is always loaded if it exists, and fitting is skipped. • <code>"always"</code>: The model is always refitted, regardless of existing fits. • <code>"on_change"</code>: The model is refitted only if the model, data, algorithm, priors, <code>sample_prior</code>, <code>stanvars</code>, covariance structure, or similar parameters have changed. If user believe a false positive occurred, the <code>[brms::brmsfit_needs_refit()]</code> can be used 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.
<code>future</code>	Logical; If TRUE, the future package is used for parallel execution of the chains. In this case, the <code>cores</code> 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 <code>"future"</code> option.
<code>sum_zero</code>	Currently ignored. Placeholder for future development.
<code>global_args</code>	Currently ignored. Placeholder for future development.
<code>parameterization</code>	A character string specifying the type of parameterization to use for drawing group-level random effects. Options are: <code>'ncp'</code> for Non-Centered Parameterization (NCP), and <code>'cp'</code> 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 `stpye` 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 **rns**. 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 **rns** 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**, (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 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 provides full flexibility in specifying predictors, degrees of freedom for design matrices, priors, and initial values. It also supports unquoted argument evaluation for convenience (e.g., `univariate_by = sex` is treated equivalently to `univariate_by = 'sex'`).

Priors The following guidelines apply to all prior specifications. For full details, see `brms::prior()`:

- **Allowed distributions:** normal, student_t, cauchy, lognormal, uniform, exponential, gamma, and inv_gamma.
- **Bounds:** Upper and lower bounds can be defined for any distribution using `lb` and `ub` (default: NA). Strictly positive distributions (exponential, gamma, inv_gamma) automatically set their lower bound to zero.
- **Scaling:** Location-scale distributions (normal, student_t, cauchy, lognormal) support an `autoscale` argument (default: FALSE) that multiplies the scale parameter by a numeric value. While **brms** typically restricts scaling factors to 1.0 or 2.5, **bsitar** accepts any real number (e.g., `autoscale = 5.0`).
- **Transformations:** For location-scale distributions, `fxl` (function location) and `fxs` (function scale) apply transformations to their respective parameters. For example, `normal(2, 5, fxl = 'log', fxs = 'sqrt')` translates to `normal(log(2), sqrt(5))`. Alternatively, `fxls` (function location scale) simultaneously transforms both parameters when they are interdependent (e.g., log-normal transformations). This can be passed as a character string or list of functions.
- **Uniform distributions:** The `addrange` argument symmetrically widens the prior range. For instance, `uniform(a, b, addrange = 5)` adjusts the bounds to `uniform(a - 5, b + 5)`.
- **Exponential distributions:** The specified rate parameter is internally inverted. Therefore, `exponential(10.0)` is evaluated as `exponential(1.0 / 10.0) = exponential(0.1)`.
- **Syntax defaults:** Missing arguments automatically default to their standard positional values. For example, `normal(location = 5, scale = 1)` simplifies to `normal(5, 1)`.
- **Multivariate/Univariate-by models:** Priors can be applied globally across all submodels (e.g., `normal(5, 1)`) or specified individually via a list (e.g., `list(normal(5, 1), normal(10, 5))`).
- **Data-driven parameters:** For location-scale distributions, parameters can be derived directly from the response variable. The location can be set to the mean (`ymean`) or median (`ymedian`), and the scale to the standard deviation (`ysd`) or median absolute deviation (`ymad`). Alternatively, coefficients from a simple linear model (`lm(y ~ age)`) can be used. Examples include `normal(ymean, ysd)` or `normal(lm, ysd)`.

Note: Data-driven options (`ymean`, `ymedian`, `ysd`, `ymad`, and `lm`) are exclusive to the fixed effect parameter `a` and cannot be used for `b`, `c`, or `d`.

Initials 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': Draws initial values directly from the prior distribution.
- 'ymean': Initializes using the mean of the response variable.
- 'ymedian': Initializes using the median of the response variable.
- 'lm': Initializes using coefficients from a simple linear model fitted to the data.

Note: Data-driven initial values ('ymean', 'ymedian', and 'lm') are exclusive to the fixed effect parameter α . For univariate_by and multivariate models, initial values can be applied globally across all submodels (e.g., `a_init_beta = '0'`) or specified individually via a list (e.g., `a_init_beta = list('0', 'lm')`).

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 JJ, 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.

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.
#
# Description of Berkley height dataset is provided in the package
# documentation (help file: ?bsitar::berkeley). A detailed comparison of
# the bsitar and sitar models is provided in the bsitar package vignette
# "Bayesian_SITAR_model_fit"

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

```

# 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 documentation for prior details.
  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 = NULL,
    chains = 2, cores = 2, iter = 1000, thin = 6)
}

# Model summary
summary(model)

# Evaluate model fit using the posterior predictive checks (PPC) plot.
# plot_ppc() is a wrapper for the pp_check() from 'brms' package.
plot_ppc(model, ndraws = NULL)

# Plot distance and velocity curves using plot_conditional_effects.
# plot_conditional_effects() is a wrapper for conditional_effects()
# from 'brms' package. However, unlike conditional_effects() which plots
# only the distance curve, plot_conditional_effects() plot velocity curve
# in addition to the distance curve

# Distance curve
plot_conditional_effects(model, deriv = 0)

# Velocity curve
plot_conditional_effects(model, deriv = 1)

```

```
# A custom function plot_curves() can also be used to plot distance and
# velocity curves along with the growth parameter such as APGV
# plot_curves() is similar to plot() from the sitar package.
plot_curves(model, apv = TRUE)
```

compare_models.bgmfit *Model comparison with the loo package*

Description

A wrapper around `brms::loo_compare()` to compare models using fit criteria.

Usage

```
## S3 method for class 'bgmfit'
compare_models(
  model,
  ...,
  criterion = "loo",
  model_name = NULL,
  check_criterion = TRUE,
  add_criterion_args = list(),
  expose_model_functions_args = list(),
  expose_function = FALSE,
  verbose = FALSE
)

compare_models(model, ...)

## S3 method for class 'list'
compare_models(model, ...)

## S3 method for class 'character'
compare_models(model, ...)

## Default S3 method:
compare_models(model, ...)

compare_model(model, ...)
```

Arguments

<code>model</code>	An R object of class <code>bsitar</code> or a list of models.
<code>...</code>	More <code>brmsfit</code> objects.

criterion	The name of the criterion to be extracted from brmsfit objects.
model_name	Optional name of the model. If NULL (the default) the name is taken from the call to x.
check_criterion	A logical (default FALSE) indicating whether to check and add fit criterion if not already added to model objects
add_criterion_args	A named list to set up the arguments for the add_model_criterion() function. Ignored when check_criterion = FALSE or when each model objects has pre computed fit criteria.
expose_model_functions_args	A named list to set up the arguments for the expose_model_functions() function. Ignored when check_criterion = FALSE or when each model objects has pre computed fit criteria.
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 (default FALSE) to print some useful information.

Details

All models should have pre computed fit criterion See [add_model_criterion\(\)](#) for more help.

Value

An object of class "compare.loo".

Author(s)

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

See Also

[brms::loo_compare\(\)](#)

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)

# For illustration purposes, we compare model with itself
# In the example below, compare_models() should indicate no difference
# between model_1 and model_2 as both these models are exactly identical

# Add model fit criteria (e.g., WAIC).
model_1 <- add_model_criterion(model, criterion = c("waic"))
model_2 <- add_model_criterion(model, criterion = c("waic"))

# compare models model_1 and model_2
compare_model_12 <- compare_models(model_1, model_2, criterion = c("waic"))

# compare models model_1 and model_2 - model names: "mods[[1L]]" "mods[[2L]]"
mods <- list(model_1, model_2)
compare_model_12 <- compare_models(mods)

# compare models model_1 and model_2 - model names extracted as such
compare_model_12 <- compare_models(list(model_1, model_2))
compare_model_12 <- compare_models("list(model_1, model_2)")
```

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 <code>bgmfit</code> .
scode	A character string containing the user-defined Stan function(s) in Stan code. If NULL (the default), the <code>scode</code> will be retrieved from the <code>model</code> .
expose	A logical (default TRUE) to indicate whether to expose the functions and add them as an attribute to the <code>model</code> .
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 <code>expose = TRUE</code> , it is advisable to set <code>returnobj = TRUE</code> .
vectorize	A logical (default FALSE) to indicate whether the exposed functions should be vectorized using <code>base::Vectorize()</code> . Note that currently, <code>vectorize</code> 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 (' <code>rstan</code> ' or ' <code>cmdstanr</code> ') for compiling and exposing stan functions. If NULL, then the backend is same as the backend method used for model fitting. Note that when <code>backend = FALSE</code> , then default backend will be set as ' <code>rstan</code> '. This is particularly useful because <code>backend = 'cmdstanr'</code> will fail on WSL system.
path	A character string to set up the path of installed <code>CmdStan</code> . If NULL (default)
envir	The environment used for function evaluation. The default is NULL, which sets the environment to <code>parent.frame()</code> . Since most post-processing functions rely on brms , it is recommended to set <code>envir = globalenv()</code> or <code>envir = .GlobalEnv</code> , especially for derivatives like velocity curves.
...	Additional arguments passed to the <code>rstan::expose_stan_functions()</code> function. The " <code>...</code> " can be used to set the compiler, which can be either <code>rstan::stanc()</code> or <code>rstan::stan_model()</code> . You can also pass other compiler-specific arguments such as <code>save_dso</code> for <code>rstan::stan_model()</code> . Note that while both <code>rstan::stanc()</code> and <code>rstan::stan_model()</code> 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	<i>Estimate fitted (Expected) values for the Bayesian SITAR model</i>
---------------------	---

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

<code>model</code>	An object of class <code>bgmfit</code> .
<code>newdata</code>	An optional data frame for estimation. If <code>NULL</code> (default), <code>newdata</code> is retrieved from the <code>model</code> .
<code>resp</code>	A character string (default <code>NULL</code>) to specify the response variable when processing posterior draws for <code>univariate_by</code> and <code>multivariate</code> models. See bsitar() for details on <code>univariate_by</code> and <code>multivariate</code> models.
<code>dpar</code>	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
<code>ndraws</code>	A positive integer indicating the number of posterior draws to use in estimation. If <code>NULL</code> (default), all draws are used.
<code>draw_ids</code>	An integer specifying the specific posterior draw(s) to use in estimation (default <code>NULL</code>).
<code>re_formula</code>	Option to indicate whether or not to include individual/group-level effects in the estimation. When <code>NA</code> (default), individual-level effects are excluded, and population average growth parameters are computed. When <code>NULL</code> , individual-level effects are included in the computation, and the resulting growth parameters are individual-specific. In both cases (<code>NA</code> or <code>NULL</code>), continuous and factor covariates are appropriately included in the estimation. Continuous covariates are set to their means by default (see <code>numeric_cov_at</code> for details), while factor covariates remain unaltered, allowing for the estimation of covariate-specific population average and individual-specific growth parameters.
<code>allow_new_levels</code>	A flag indicating if new levels of group-level effects are allowed (defaults to <code>FALSE</code>). Only relevant if <code>newdata</code> is provided.
<code>sample_new_levels</code>	Indicates how to sample new levels for grouping factors specified in <code>re_formula</code> . This argument is only relevant if <code>newdata</code> is provided and <code>allow_new_levels</code> is set to <code>TRUE</code> . If <code>"uncertainty"</code> (default), 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 <code>"gaussian"</code> , sample new levels from the (multivariate) normal distribution implied by the group-level standard deviations and correlations. This options may be useful for conducting Bayesian power analysis or predicting new levels in situations where relatively few levels were observed in the <code>old_data</code> . If <code>"old_levels"</code> , 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.
<code>incl_autocor</code>	A flag indicating if correlation structures originally specified via <code>autocor</code> should be included in the predictions. Defaults to <code>TRUE</code> .
<code>numeric_cov_at</code>	An optional (named list) argument to specify the value of continuous covariate(s). The default <code>NULL</code> option sets the continuous covariate(s) to their mean. Alternatively, a named list can be supplied to manually set these values. For example, <code>numeric_cov_at = list(xx = 2)</code> will set the continuous covariate vari-

	able 'xx' to 2. The argument <code>numeric_cov_at</code> is ignored when no continuous covariates are included in the model.
<code>levels_id</code>	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, <code>levels_id</code> is automatically inferred from the model fit. For models with three or more levels, <code>levels_id</code> is inferred from the model fit under the assumption that hierarchy is specified from the lowest to the uppermost level, i.e., <code>id</code> followed by <code>study</code> , where <code>id</code> is nested within <code>study</code> . However, it is not guaranteed that <code>levels_id</code> is sorted correctly, so it is better to set it manually when fitting a model with three or more levels of hierarchy.
<code>avg_reffects</code>	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, <code>avg_reffects = list(feby = 'study', reby = NULL, over = 'age')</code> .
<code>aux_variables</code>	An optional argument to specify the variable(s) that can be passed to the <code>ipts</code> 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 <code>aux_variables</code> .
<code>grid_add</code>	An optional argument to specify the variable(s) that can be passed to the <code>marginalEffects::datagrid()</code> . 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 <code>grid_add</code> . Note that unlike <code>aux_variables</code> which are passed to the internal data functions such as <code>'get.newdata'</code> , the <code>grid_add</code> are passed to the <code>marginalEffects::datagrid()</code> .
<code>ipts</code>	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 <code>length.out</code> argument for <code>seq()</code>, controlling the smoothness of distance and velocity curves without altering the predictor range. NULL (the default) Engages automatic behavior based on the <code>dpar</code> argument: it is internally set to 50 for the mean response (<code>dpar = 'mu'</code>), ensuring smooth curves, while for the distributional parameters (e.g., <code>dpar = 'sigma'</code>), 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: <code>fitted_draws()</code>, <code>predict_draws()</code>, <code>growthparameters()</code>, <code>plot_curves()</code>, <code>get_predictions()</code>, <code>get_comparisons()</code>, and <code>get_growthparameters()</code>.
<code>deriv</code>	An integer indicating whether to estimate the distance curve or its derivative (velocity curve). The default <code>deriv = 0</code> is for the distance curve, while <code>deriv = 1</code> is for the velocity curve.
<code>model_deriv</code>	A logical value specifying whether to estimate the velocity curve from the derivative function or by differentiating the distance curve. Set <code>model_deriv = TRUE</code>

	for functions that require the velocity curve, such as <code>growthparameters()</code> and <code>plot_curves()</code> . Set it to <code>NULL</code> for functions that use the distance curve (i.e., fitted values), such as <code>loo_validation()</code> and <code>plot_ppc()</code> .
summary	A logical value indicating whether only the estimate should be computed (<code>TRUE</code>), or whether the estimate along with SE and CI should be returned (<code>FALSE</code> , default). Setting <code>summary</code> to <code>FALSE</code> will increase computation time. Note that <code>summary = FALSE</code> is required to obtain correct estimates when <code>re_formula = NULL</code> .
robust	A logical value to specify the summary options. If <code>FALSE</code> (default), the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If <code>TRUE</code> , the median and median absolute deviation (MAD) are applied instead. Ignored if <code>summary</code> is <code>FALSE</code> .
transform_draws	A function applied to individual draws from the posterior distribution before computing summaries (default <code>NULL</code>). The argument <code>transform_draws</code> is derived from the <code>marginaleffects::predictions()</code> function and should not be confused with the <code>transform</code> argument from the deprecated <code>brms::posterior_predict()</code> function. It's important to note that for both <code>marginaleffects::predictions()</code> and <code>marginaleffects::avg_predictions()</code> , the <code>transform_draws</code> argument takes precedence over the <code>transform</code> argument. Note that when <code>transform_draws = NULL</code> , an attempt is made to automatically set <code>transform_draws = 'exp'</code> for <code>dpar = 'sigma'</code> . User can set <code>transform_draws = FALSE</code> to turn off this automatic assignment of 'exp' to the <code>transform_draws</code> . It is also important to set <code>transform_draws = FALSE</code> when computing the first derivative (velocity) for <code>dpar = 'sigma'</code> .
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 <code>quantile</code> function. Only used if <code>summary</code> is <code>TRUE</code> .
xrange	An integer to set the predictor range (e.g., age) when executing the interpolation via <code>ipts</code> . By default, <code>NULL</code> sets the individual-specific predictor range. Setting <code>xrange = 1</code> applies the same range for individuals within the same higher grouping variable (e.g., study). Setting <code>xrange = 2</code> applies an identical range across the entire sample. Alternatively, a numeric vector (e.g., <code>xrange = c(6, 20)</code>) 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 <code>xrange_search = NULL</code> .
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 <code>sitar::getPeak()</code> 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. <i>Note:</i> 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 <code>optimize_model()</code> 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 <code>expose_functions</code> argument from the <code>bsitar()</code> call. Manual specification of <code>usesavedfuns</code> 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, <code>clearenvfuns</code> is set based on the value of <code>usesavedfuns</code> : TRUE if <code>usesavedfuns</code> = TRUE, or FALSE if <code>usesavedfuns</code> = FALSE.
funlist	A list (default NULL) specifying function names. This is rarely needed, as required functions are typically retrieved automatically. A use case for <code>funlist</code> is when <code>sigma_formula</code> , <code>sigma_formula_gr</code> , or <code>sigma_formula_gr_str</code> use an external function (e.g., <code>poly(age)</code>). The <code>funlist</code> should include function names defined in the <code>globalenv()</code> . For functions needing both distance and velocity curves (e.g., <code>plot_curves(..., opt = 'dv')</code>), <code>funlist</code> 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 <code>xvar</code> is inferred internally. A use case is when conflicting variables exist (e.g., <code>sigma_formula</code>) 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 <code>xvar</code> is set as <code>difx</code> if specified. The argument <code>difx</code> is evaluated only when <code>dpar</code> = 'sigma', ignored otherwise. Note that argument <code>xvar</code> itself is rarely used because <code>xvar</code> is inferred internally. A use case is when conflicting variables exist (e.g., <code>sigma_formula</code>) and user wants to set a specific variable as 'x'.
idvar	A character string (default NULL) specifying the 'id' variable. Rarely used because <code>idvar</code> is inferred internally.
itransform	A character string (default NULL) indicating the variables names that are reverse transformed. Options are <code>c("x", "y", "sigma")</code> . The <code>itransform</code> is primarily used to get the <code>xvar</code> variable at original scale i.e., <code>itransform</code> = 'x'. To turn off all transformations, use <code>itransform</code> = "". When <code>itransform</code> = NULL, the appropriate transformation for <code>xvar</code> is selected automatically. Note that when no match for <code>xvar</code> is found in the data frame, the <code>itransform</code> will be ignored within the calling function, <code>'prepare_transformations()'</code> .
newdata_fixed	An indicator specifying how to handle the format and structure of the user-provided <code>newdata</code> before use. The available options are: • NULL (default): Checks the data format and structure of <code>newdata</code>, and applies <code>prepare_data2</code> and <code>prepare_transformations</code> as needed. • TRUE: Returns <code>newdata</code> as-is, without any format checking or transformation. • 0: Same as TRUE — returns <code>newdata</code> without checking the data format or applying <code>prepare_data2</code> and <code>prepare_transformations</code>. • 1: Checks the data format and, if needed, prepares it using <code>prepare_data2</code>, but does not apply <code>prepare_transformations</code>. • 2: Applies <code>prepare_transformations</code> only, assuming the data format is already correct.

	It is strongly recommended to leave both <code>newdata = NULL</code> and <code>newdata_fixed = NULL</code> (the defaults). In this case, the data used during model fitting is automatically retrieved, checked for the required format, and transformed as needed via <code>prepare_data2</code> and <code>prepare_transformations</code> . The flags <code>newdata_fixed = 0</code> , <code>1</code> , and <code>2</code> are intended primarily for internal use.
<code>envir</code>	The environment used for function evaluation. The default is <code>NULL</code> , which sets the environment to <code>parent.frame()</code> . Since most post-processing functions rely on brms , it is recommended to set <code>envir = globalenv()</code> or <code>envir = .GlobalEnv</code> , especially for derivatives like velocity curves.
<code>...</code>	Additional arguments passed to the <code>brms::fitted.brmsfit()</code> function. For details on available options, please refer to <code>brms::fitted.brmsfit()</code> .

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 <code>bgmfit</code> .
resp	A character string (default <code>NULL</code>) to specify the response variable when processing posterior draws for <code>univariate_by</code> and <code>multivariate</code> models. See bsitar() for details on <code>univariate_by</code> and <code>multivariate</code> 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 <code>NULL</code> (default), all draws are used.
draw_ids	An integer specifying the specific posterior draw(s) to use in estimation (default <code>NULL</code>).
newdata	An optional data frame for estimation. If <code>NULL</code> (default), <code>newdata</code> 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 <code>NULL</code> (default), no custom grid is used. The grid can be constructed using marginaleffects::datagrid() . If <code>datagrid = list()</code> , essential arguments such as <code>model</code> and <code>newdata</code> 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 <code>NA</code> (default), individual-level effects are excluded, and population average growth parameters are computed. When <code>NULL</code> , individual-level effects are included in the computation, and the resulting growth parameters are individual-specific. In both cases (<code>NA</code> or <code>NULL</code>), 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 <code>newdata</code> . 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 <code>FALSE</code>). Only relevant if <code>newdata</code> is provided.

sample_new_levels

Indicates how to sample new levels for grouping factors specified in `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 were 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 <code>ipts</code> argument. Available options: • 'm1': Adapted from the iapvbs package (documented here). This method internally constructs the data frame based on the model configuration. <i>Note:</i> This method may fail if the model includes covariates (especially in <code>univariate_by</code> models). • 'm2': Based on the JMbayes package (documented here). This method uses the exact data frame from the model fit (<code>fit\$data</code>) and is generally more robust. If <code>idata_method = NULL</code>, '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 <code>length.out</code> argument for <code>seq()</code>, controlling the smoothness of distance and velocity curves without altering the predictor range. NULL (the default) Engages automatic behavior based on the <code>dpar</code> argument: it is internally set to 50 for the mean response (<code>dpar = 'mu'</code>), ensuring smooth curves, while for the distributional parameters (e.g., <code>dpar = 'sigma'</code>), 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: <code>fitted_draws()</code>, <code>predict_draws()</code>, <code>growthparameters()</code>, <code>plot_curves()</code>, <code>get_predictions()</code>, <code>get_comparisons()</code>, and <code>get_growthparameters()</code>.
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 <code>future.apply::future_sapply()</code> .
future_session	A character string or a named list specifying the parallel plan when <code>future = TRUE</code>. • Character string: Defaults to "multisession". Use "multicore" for forking (not supported on Windows). • Named list: Useful for advanced plans like <code>future.mirai::mirai_cluster()</code>. The list must contain: – <code>future_session</code>: The planner function (e.g., <code>mirai_cluster</code>). – Additional named arguments passed to the planner (e.g., <code>daemons</code> for <code>mirai::daemons()</code>).

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 <code>future::plan("multicore")</code>. Input options: • List: (e.g., <code>list(1:6, 7:10)</code>). Each element is a sequence of numbers passed to <code>draw_ids</code> to process in a separate chunk. • Numeric vector of length 2: (e.g., <code>c(100, 4)</code>). The first element is the total number of draws, and the second is the number of splits. Indices are generated via <code>parallel::splitIndices()</code>. • Numeric vector of length 1: Specifies the total number of draws. Splits are calculated automatically. • TRUE: Automatically creates splits based on <code>ndraws</code> and <code>cores</code>. If both <code>ndraws</code> and <code>draw_ids</code> are specified, <code>draw_ids</code> takes precedence. This is consistent with post-processing functions in the <code>bsitar</code> and <code>brms</code> packages. • FALSE: Splitting is disabled. Note: On Windows with <code>future::plan("multisession")</code>, background R processes may not free resources automatically. If needed, use <code>installr::kill_all_Rscript_s()</code> to terminate them.
future_method	A character string (default 'future') specifying the parallel computation back-end. Options include: • 'future': Uses <code>future.apply::future_lapply()</code> for execution. • 'dofuture': Uses <code>foreach::foreach()</code> via the <code>doFuture</code> package.
future_re_expose	A logical value (default NULL) indicating whether to re-expose internal Stan functions when <code>future = TRUE</code>. This is critical when <code>future::plan()</code> 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 <code>usedtplyr</code> argument is evaluated only when <code>method = 'custom'</code>.
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 <code>future::availableCores() - 1</code>. • On non-Windows systems, this can be controlled globally via the <code>mc.cores</code> 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 <code>marginaleffects::comparisons()</code> (if FALSE) or <code>marginaleffects::avg_comparisons()</code> (if TRUE). Default is FALSE.
plot	A logical indicating whether to plot the comparisons using <code>marginaleffects::plot_comparisons()</code> . 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., <code>geom_ribbon</code> , then include <code>rm_geom_ribbon = TRUE</code> . 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: <pre>mapping_facet = list(group = 'id', colour = 'id', facet_wrap = c('study'), rm_geom_ribbon = TRUE).</pre>
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., <code>variables = list('x' = 1e-6)</code> where 'x' is the level 1 predictor. By default, <code>variables = list('age' = 1e-6)</code> 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 <code>variables = list('age' = 0)</code> . Note that if the default behavior is used (<code>deriv = 0</code> and <code>variables = list('x' = 1e-6)</code>), additional arguments cannot be passed to variables. In contrast, when using an alternative approach (<code>deriv = 0</code> and <code>variables = list('x' = 0)</code>), additional options can be passed to the <code>marginaleffects::comparisons()</code> and <code>marginaleffects::avg_comparisons()</code> 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 <code>?stats::fivenum</code>.
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 <code>model_deriv = TRUE</code> for functions that require the velocity curve, such as <code>growthparameters()</code> and <code>plot_curves()</code> . Set it to <code>NULL</code> for functions that use the distance curve (i.e., fitted values), such as <code>loo_validation()</code> and <code>plot_ppc()</code> .
method	A character string specifying whether to compute comparisons using the marginal-effects machinery (<code>method = 'pkg'</code>) or via custom functions for efficiency (<code>method = 'custom'</code>). Default is <code>'pkg'</code> . The <code>'custom'</code> method is useful when testing hypotheses and should be used cautiously.
method_call	A character string specifying whether to compute comparisons using <code>marginalEffects::predictions()</code> (<code>method_call = 'predictions'</code>) or <code>marginalEffects::comparisons()</code> (<code>method_call = 'comparisons'</code>). The <code>'method_call'</code> is evaluated only when <code>method = 'custom'</code> ignored otherwise. Default <code>method_call = NULL</code> allows automatic selection of the <code>'method_call'</code> depending on whether comparisons are for slopes or predictions.
marginals	A list, data.frame, or tibble of velocity returned by the marginalEffects functions (default <code>NULL</code>). This is only evaluated when <code>method = 'custom'</code> . The <code>marginals</code> can be the output from marginalEffects functions or posterior draws from <code>marginalEffects::posterior_draws()</code> . The <code>marginals</code> argument is primarily used for internal purposes only.
pdrawso	A character string (default <code>FALSE</code>) to indicate whether to return the original posterior draws for parameters. Options include: • <code>'return'</code>: returns the original posterior draws, • <code>'add'</code>: adds the original posterior draws to the outcome. When <code>pdrawso = TRUE</code>, the default behavior is <code>pdrawso = 'return'</code>. Note that the posterior draws are returned before calling <code>marginalEffects::posterior_draws()</code>.
pdrawsp	A character string (default <code>FALSE</code>) to indicate whether to return the posterior draws for parameters. Options include: • <code>'return'</code>: returns the posterior draws for parameters, • <code>'add'</code>: adds the posterior draws to the outcome. When <code>pdrawsp = TRUE</code>, the default behavior is <code>pdrawsp = 'return'</code>. The <code>pdrawsp</code> represent the parameter estimates for each of the posterior samples, and the summary of these are the estimates returned.
pdrawsh	A character string (default <code>FALSE</code>) to indicate whether to return the posterior draws for parameter contrasts. Options include: • <code>'return'</code>: 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 <code>sitar::getPeak()</code>, <code>sitar::getTakeoff()</code>, and <code>sitar::getTrough()</code> 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 <code>marginaleffects::predictions()</code> function and should not be confused with the transform argument from the deprecated <code>brms::posterior_predict()</code> function. It's important to note that for both <code>marginaleffects::predictions()</code> and <code>marginaleffects::avg_predictions()</code>, 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").
- wt** 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 `wt`, 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 `wt=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 <code>marginaleffects::comparisons()</code> .
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 <code>marginaleffects::comparisons()</code> as a data.frame, with column names such as <code>conf.low</code> as <code>Q2.5</code> , <code>conf.high</code> as <code>Q97.5</code> , and dropping unnecessary columns like <code>term</code> , <code>contrast</code> , and <code>predicted</code> .
estimate_center	A character string (default NULL) specifying how to center estimates: either 'mean' or 'median'. This option sets the global options as follows: <code>options("marginaleffects_posterior_center" = "mean")</code> or <code>options("marginaleffects_posterior_center" = "median")</code> . These global options are restored upon function exit using <code>base::on.exit()</code> .
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: <code>options("marginaleffects_posterior_interval" = "eti")</code> or <code>options("marginaleffects_posterior_interval" = "hdi")</code> , and is restored on exit using <code>base::on.exit()</code> .
dummy_to_factor	A named list (default NULL) to convert dummy variables into a factor variable. The list must include the following elements: <code>factor.dummy</code>: A character vector of dummy variables to be converted to factors. <code>factor.name</code>: The name for the newly created factor variable (default is 'factor.var' if NULL). <code>factor.level</code>: A vector specifying the factor levels. If NULL, levels are taken from <code>factor.dummy</code>. If <code>factor.level</code> is provided, its length must match <code>factor.dummy</code>.
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 <code>fit_criteria</code> or <code>bayes_R2</code> 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 <code>expose_functions</code> argument from the <code>bsitar()</code> call. Manual specification of <code>usesavedfuns</code> 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, <code>clearenvfuns</code> is set based on the value of <code>usesavedfuns</code> : TRUE if <code>usesavedfuns</code> = TRUE, or FALSE if <code>usesavedfuns</code> = FALSE.
funlist	A list (default NULL) specifying function names. This is rarely needed, as required functions are typically retrieved automatically. A use case for <code>funlist</code> is when <code>sigma_formula</code> , <code>sigma_formula_gr</code> , or <code>sigma_formula_gr_str</code> use an external function (e.g., <code>poly(age)</code>). The <code>funlist</code> should include function names defined in the <code>globalenv()</code> . For functions needing both distance and velocity curves (e.g., <code>plot_curves(..., opt = 'dv')</code>), <code>funlist</code> 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 <code>xvar</code> is inferred internally. A use case is when conflicting variables exist (e.g., <code>sigma_formula</code>) 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 <code>xvar</code> is set as <code>difx</code> if specified. The argument <code>difx</code> is evaluated only when <code>dpar</code> = 'sigma', ignored otherwise. Note that argument <code>xvar</code> itself is rarely used because <code>xvar</code> is inferred internally. A use case is when conflicting variables exist (e.g., <code>sigma_formula</code>) and user wants to set a specific variable as 'x'.
idvar	A character string (default NULL) specifying the 'id' variable. Rarely used because <code>idvar</code> is inferred internally.
itransform	A character string (default NULL) indicating the variables names that are reverse transformed. Options are <code>c("x", "y", "sigma")</code> . The <code>itransform</code> is primarily used to get the <code>xvar</code> variable at original scale i.e., <code>itransform</code> = 'x'. To turn off all transformations, use <code>itransform</code> = "". When <code>itransform</code> = NULL, the appropriate transformation for <code>xvar</code> is selected automatically. Note that when no match for <code>xvar</code> is found in the data frame, the <code>itransform</code> will be ignored within the calling function, <code>'prepare_transformations()'</code> .
newdata_fixed	An indicator specifying how to handle the format and structure of the user-provided <code>newdata</code> before use. The available options are: • NULL (default): Checks the data format and structure of <code>newdata</code>, and applies <code>prepare_data2</code> and <code>prepare_transformations</code> as needed. • TRUE: Returns <code>newdata</code> as-is, without any format checking or transformation.

- 0: Same as TRUE — returns newdata without checking the data format or applying prepare_data2 and prepare_transformations.
- 1: Checks the data format and, if needed, prepares it using prepare_data2, but does not apply prepare_transformations.
- 2: Applies prepare_transformations only, assuming the data format is already correct.

It is strongly recommended to leave both newdata = NULL and newdata_fixed = NULL (the defaults). In this case, the data used during model fitting is automatically retrieved, checked for the required format, and transformed as needed via prepare_data2 and prepare_transformations. The flags newdata_fixed = 0, 1, and 2 are intended primarily for internal use.

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::comparisons\(\)](#) 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

<code>model</code>	An object of class <code>bgmfit</code> .
<code>resp</code>	A character string (default <code>NULL</code>) to specify the response variable when processing posterior draws for <code>univariate_by</code> and <code>multivariate</code> models. See bsitar() for details on <code>univariate_by</code> and <code>multivariate</code> models.
<code>dpar</code>	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
<code>ndraws</code>	A positive integer indicating the number of posterior draws to use in estimation. If <code>NULL</code> (default), all draws are used.
<code>draw_ids</code>	An integer specifying the specific posterior draw(s) to use in estimation (default <code>NULL</code>).

<code>newdata</code>	An optional data frame for estimation. If NULL (default), <code>newdata</code> is retrieved from the model.
<code>datagrid</code>	A grid of user-specified values to be used in the <code>newdata</code> 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 <code>datagrid</code> 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 <code>datagrid = list()</code> , in which case essential arguments like <code>model</code> and <code>newdata</code> are inferred from the respective arguments specified elsewhere. Additionally, the first-level predictor (such as <code>age</code>) and any covariates included in the model (e.g., <code>gender</code>) are automatically inferred from the <code>model</code> object.
<code>re_formula</code>	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 <code>numeric_cov_at</code> for details), while factor covariates remain unaltered, allowing for the estimation of covariate-specific population average and individual-specific growth parameters.
<code>newdata2</code>	A named list of objects containing new data, which cannot be passed via argument <code>newdata</code> . Required for some objects used in autocorrelation structures, or stanvars .
<code>allow_new_levels</code>	A flag indicating if new levels of group-level effects are allowed (defaults to FALSE). Only relevant if <code>newdata</code> is provided.
<code>sample_new_levels</code>	Indicates how to sample new levels for grouping factors specified in <code>re_formula</code> . This argument is only relevant if <code>newdata</code> is provided and <code>allow_new_levels</code> is set to TRUE. If "uncertainty" (default), 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 were observed in the <code>old_data</code> . 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.
<code>parameter</code>	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
- 'spgv': Size (length/height) at peak growth velocity age, 'apgv'
- 'stgv': Size at takeoff growth velocity age, 'atgv'
- 'scgv': Size at cessation growth velocity age, 'acgv'
- 'satXX': Size at specific age XX years of predictor xvar (e.g., 'sat15' = size at 15 years, 'sat12.5' = size at 12.5 years)
- 'all': All six primary velocity/age parameters ('apgv', 'pgv', 'atgv', 'tgv', 'acgv', 'cgv'). Cannot be used when 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 <code>ipts</code> . By default, NULL sets the individual-specific predictor range. Setting <code>xrange = 1</code> applies the same range for individuals within the same higher grouping variable (e.g., study). Setting <code>xrange = 2</code> applies an identical range across the entire sample. Alternatively, a numeric vector (e.g., <code>xrange = c(6, 20)</code>) 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 <code>cgv</code> and <code>acgv</code> growth parameters. The <code>acg_velocity</code> should be greater than 0 and less than 1. The default value of <code>acg_velocity = 0.10</code> 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 <code>digits</code> value is passed to the <code>base::round()</code> 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, <code>numeric_cov_at = list(xx = 2)</code> will set the continuous covariate variable 'xx' to 2. The argument <code>numeric_cov_at</code> 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 <code>ipts</code> 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 <code>aux_variables</code> .
<code>grid_add</code>	An optional argument to specify the variable(s) that can be passed to the <code>marginalEffects::datagrid()</code> . 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 <code>grid_add</code> . Note that unlike <code>aux_variables</code> which are passed to the internal data functions such as 'get.newdata', the <code>grid_add</code> are passed to the <code>marginalEffects::datagrid()</code> .
<code>levels_id</code>	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, <code>levels_id</code> is automatically inferred from the model fit. For models with three or more levels, <code>levels_id</code> 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 <code>levels_id</code> is sorted correctly, so it is better to set it manually when fitting a model with three or more levels of hierarchy.
<code>avg_reffects</code>	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, <code>avg_reffects = list(feby = 'study', reby = NULL, over = 'age')</code> .
<code>idata_method</code>	A character string specifying the interpolation method (default NULL). The number of interpolation points is controlled by the <code>ipts</code> argument. Available options: • 'm1': Adapted from the iapvbs package (documented here). This method internally constructs the data frame based on the model configuration. <i>Note:</i> This method may fail if the model includes covariates (especially in <code>univariate_by</code> models). • 'm2': Based on the JMbayes package (documented here). This method uses the exact data frame from the model fit (<code>fit\$data</code>) and is generally more robust. If <code>idata_method = NULL</code> , 'm2' is automatically selected. It is recommended to use 'm2' if 'm1' encounters errors with covariate-dependent models.
<code>ipts</code>	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 <code>length.out</code> argument for <code>seq()</code> , controlling the smoothness of distance and velocity curves without altering the predictor range. NULL (the default) Engages automatic behavior based on the <code>dpar</code> argument: it is internally set to 50 for the mean response (<code>dpar = 'mu'</code>), ensuring smooth curves, while for the distributional parameters (e.g., <code>dpar = 'sigma'</code>), 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: <code>fitted_draws()</code>, <code>predict_draws()</code>, <code>growthparameters()</code>, <code>plot_curves()</code>, <code>get_predictions()</code>, <code>get_comparisons()</code>, and <code>get_growthparameters()</code>.
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 <code>future.apply::future_apply()</code> .
future_session	A character string or a named list specifying the parallel plan when <code>future = TRUE</code>. • Character string: Defaults to "multisession". Use "multicore" for forking (not supported on Windows). • Named list: Useful for advanced plans like <code>future.mirai::mirai_cluster()</code>. The list must contain: – <code>future_session</code>: The planner function (e.g., <code>mirai_cluster</code>). – Additional named arguments passed to the planner (e.g., <code>daemons</code> for <code>mirai::daemons()</code>). Example: <code>list(future_session = mirai_cluster, daemons = list(...))</code>.
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 <code>future::plan("multicore")</code>. Input options: • List: (e.g., <code>list(1:6, 7:10)</code>). Each element is a sequence of numbers passed to <code>draw_ids</code> to process in a separate chunk. • Numeric vector of length 2: (e.g., <code>c(100, 4)</code>). The first element is the total number of draws, and the second is the number of splits. Indices are generated via <code>parallel::splitIndices()</code>. • Numeric vector of length 1: Specifies the total number of draws. Splits are calculated automatically. • TRUE: Automatically creates splits based on <code>ndraws</code> and <code>cores</code>. If both <code>ndraws</code> and <code>draw_ids</code> are specified, <code>draw_ids</code> takes precedence. This is consistent with post-processing functions in the <code>bsitar</code> and <code>brms</code> packages. • FALSE: Splitting is disabled. Note: On Windows with <code>future::plan("multisession")</code>, background R processes may not free resources automatically. If needed, use <code>installr::kill_all_Rscript_s()</code> to terminate them.
future_method	A character string (default 'future') specifying the parallel computation backend. Options include: • 'future': Uses <code>future.apply::future_lapply()</code> for execution. • 'doFuture': Uses <code>foreach::foreach()</code> via the <code>doFuture</code> package.
future_re_expose	A logical value (default NULL) indicating whether to re-expose internal Stan functions when <code>future = TRUE</code>. This is critical when <code>future::plan()</code> 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, <code>parallel::makeCluster()</code> 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 <code>marginaleffects::comparisons()</code> or the <code>marginaleffects::avg_comparisons()</code> function. If FALSE (default), <code>marginaleffects::comparisons()</code> is called, otherwise <code>marginaleffects::avg_comparisons()</code> is used when average = TRUE.
plot	A logical value specifying whether to plot comparisons by calling the <code>marginaleffects::plot_comparisons()</code> function (TRUE) or not (FALSE). If FALSE (default), then <code>marginaleffects::comparisons()</code> or <code>marginaleffects::avg_comparisons()</code> 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., <code>geom_ribbon</code> , then include <code>rm_geom_ribbon = TRUE</code> . 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: <pre>mapping_facet = list(group = 'id', colour = 'id', facet_wrap = c('study'), rm_geom_ribbon = TRUE).</pre>
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 <code>esp</code> 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., <code>variables = list('x' = 1e-6)</code> where 'x' is the level 1 predictor. By default, <code>variables = list('age' = 1e-6)</code> in the marginaleffects package, as velocity is usually computed by differentiating the distance curve using the <code>dydx</code> approach. When using this default, the argument <code>deriv</code> is automatically set to 0 and <code>model_deriv</code> to FALSE. If parameters are to be estimated based on the model's first derivative, <code>deriv</code> must be set to 1 and variables will be defined as <code>variables = list('age' = 0)</code>. Note that if the default behavior is used (<code>deriv = 0</code> and <code>variables = list('x' = 1e-6)</code>), additional arguments cannot be passed to variables. In contrast, when using an alternative approach (<code>deriv = 0</code> and <code>variables = list('x' = 0)</code>), additional options can be passed to the <code>marginaleffects::comparisons()</code> and <code>marginaleffects::avg_comparisons()</code> 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 <code>?stats::fivenum</code>.
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 <code>model_deriv = TRUE</code> for functions that require the velocity curve, such as <code>growthparameters()</code> and <code>plot_curves()</code>. Set it to NULL for functions that use the distance curve (i.e., fitted values), such as <code>loo_validation()</code> and <code>plot_ppc()</code>.
method	A character string indicating whether to compute estimates using the 'marginaleffects' package (<code>method = 'pkg'</code>) or custom functions for efficiency and speed (<code>method = 'custom'</code>, default). The <code>method = 'pkg'</code> option is only suitable for simple cases and should be used with caution. <code>method = 'custom'</code> 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 <code>hypothesis</code> argument, and allows users to add or return draws (see <code>pdraws</code> for details). If <code>method = 'pkg'</code>, the <code>by</code> 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 <code>sitar::getPeak()</code> , <code>sitar::getTakeoff()</code> , and <code>sitar::getTrough()</code> 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 <code>marginaleffects::predictions()</code> function and should not be confused with the transform argument from the deprecated <code>brms::posterior_predict()</code> function. It's important to note that for both <code>marginaleffects::predictions()</code> and <code>marginaleffects::avg_predictions()</code> , 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").
- wt** 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 `wt`, 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 `wt=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`, `tg`, `cg`).
- `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.

	• <code>inline</code>: Internal use only; executes custom equivalence function (not for users).
<code>eps</code>	NULL or numeric value which determines the step size to use when calculating numerical derivatives: $(f(x+eps)-f(x))/eps$. When <code>eps</code> 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 <code>eps</code> may be necessary to avoid numerical problems in certain models.
<code>constrats_by</code>	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 <code>constrats_by</code> should be a subset of those specified in the <code>by</code> argument. If <code>constrats_by</code> = NULL, it will copy all variables from <code>by</code> , except for the level-1 predictor (e.g., <code>age</code>). To disable this automatic behavior, use <code>constrats_by</code> = FALSE. This argument is evaluated only when <code>method</code> = 'custom' and <code>hypothesis</code> is not NULL.
<code>constrats_at</code>	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., <code>constrats_at</code> = <code>list(age = 'min')</code>) or as numeric vectors (e.g., <code>constrats_at</code> = <code>list(age = c(6, 7))</code>). If <code>constrats_at</code> = NULL, level-1 predictors (e.g., <code>age</code>) are automatically set to their unique values (i.e., <code>constrats_at</code> = <code>list(age = 'unique')</code>). To turn off this behavior, use <code>constrats_at</code> = FALSE. Note that <code>constrats_at</code> only affects data subsets prepared via <code>marginalEffects::datagrid()</code> or the <code>newdata</code> argument. The argument is evaluated only when <code>method</code> = 'custom' and <code>hypothesis</code> is not NULL.
<code>constrats_subset</code>	A named list (default FALSE) to subset draws for which contrasts should be computed via the hypothesis argument. This is similar to <code>constrats_at</code> , except that <code>constrats_subset</code> filters draws based on the character vector of variable names (e.g., <code>constrats_subset</code> = <code>list(id = c('id1', 'id2'))</code>) rather than numeric values. The argument is evaluated only when <code>method</code> = 'custom' and <code>hypothesis</code> is not NULL.
<code>reformat</code>	A logical (default TRUE) indicating whether to reformat the output returned by <code>marginalEffects</code> as a data frame. Column names are redefined as <code>conf.low</code> to Q2.5 and <code>conf.high</code> to Q97.5 (assuming <code>conf_int</code> = 0.95). Additionally, some columns (<code>term</code> , <code>contrast</code> , etc.) are dropped from the data frame.
<code>estimate_center</code>	A character string (default NULL) specifying how to center estimates: either 'mean' or 'median'. This option sets the global options as follows: <code>options("marginalEffects_posterior_center" = "mean")</code> or <code>options("marginalEffects_posterior_center" = "median")</code> . These global options are restored upon function exit using <code>base::on.exit()</code> .
<code>estimate_interval</code>	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: <code>options("marginalEffects_posterior_interval" = "eti")</code> or <code>options("marginalEffects_posterior_interval" = "hdi")</code> , and is restored on exit using <code>base::on.exit()</code> .

<code>dummy_to_factor</code>	A named list (default NULL) to convert dummy variables into a factor variable. The list must include the following elements: • <code>factor.dummy</code>: A character vector of dummy variables to be converted to factors. • <code>factor.name</code>: The name for the newly created factor variable (default is <code>'factor.var'</code> if NULL). • <code>factor.level</code>: A vector specifying the factor levels. If NULL, levels are taken from <code>factor.dummy</code>. If <code>factor.level</code> is provided, its length must match <code>factor.dummy</code>.
<code>verbose</code>	A logical argument (default FALSE) to specify whether to print information collected during the setup of the object(s).
<code>expose_function</code>	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 <code>bayes_R2</code> during model optimization. • FALSE (default): Stan functions are not exposed. • NULL: The setting is inherited from the original <code>bsitar()</code> model fit configuration. Note: In the <code>optimize_model()</code> function, the default is NULL (inheriting behavior), whereas other post-processing functions default to FALSE.
<code>usesavedfuns</code>	A logical value (default NULL) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the <code>expose_functions</code> argument from the <code>bsitar()</code> call. Manual specification of <code>usesavedfuns</code> is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.
<code>clearenvfuns</code>	A logical value indicating whether to clear the exposed Stan functions from the environment (TRUE) or not (FALSE). If NULL, <code>clearenvfuns</code> is set based on the value of <code>usesavedfuns</code> : TRUE if <code>usesavedfuns = TRUE</code> , or FALSE if <code>usesavedfuns = FALSE</code> .
<code>funlist</code>	A list (default NULL) specifying function names. This is rarely needed, as required functions are typically retrieved automatically. A use case for <code>funlist</code> is when <code>sigma_formula</code> , <code>sigma_formula_gr</code> , or <code>sigma_formula_gr_str</code> use an external function (e.g., <code>poly(age)</code>). The <code>funlist</code> should include function names defined in the <code>globalenv()</code> . For functions needing both distance and velocity curves (e.g., <code>plot_curves(..., opt = 'dv')</code>), <code>funlist</code> must include two functions: one for the distance curve and one for the velocity curve.
<code>xvar</code>	A character string (default NULL) specifying the 'x' variable. Rarely used because <code>xvar</code> is inferred internally. A use case is when conflicting variables exist (e.g., <code>sigma_formula</code>) and user wants to set a specific variable as 'x'.
<code>difx</code>	A character string (default NULL) specifying the 'x' variable that should be used for manual differentiation of the distance curve. Internally, the <code>xvar</code> is set as <code>difx</code> if specified. The argument <code>difx</code> is evaluated only when <code>dpar =</code>

	'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 off 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 specifying how to handle the format and structure of the user-provided newdata before use. The available options are: • NULL (default): Checks the data format and structure of newdata, and applies prepare_data2 and prepare_transformations as needed. • TRUE: Returns newdata as-is, without any format checking or transformation. • 0: Same as TRUE — returns newdata without checking the data format or applying prepare_data2 and prepare_transformations. • 1: Checks the data format and, if needed, prepares it using prepare_data2, but does not apply prepare_transformations. • 2: Applies prepare_transformations only, assuming the data format is already correct. It is strongly recommended to leave both newdata = NULL and newdata_fixed = NULL (the defaults). In this case, the data used during model fitting is automatically retrieved, checked for the required format, and transformed as needed via prepare_data2 and prepare_transformations. The flags newdata_fixed = 0, 1, and 2 are intended primarily for internal use.
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 <code>brms::fitted.brmsfit()</code> function. Please see <code>brms::fitted.brmsfit()</code> 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_model_criterion.bgmfit

Return Model-Fit Criteria for a Bayesian SITAR Model

Description

get_model_criterion() is a wrapper around [add_model_criterion\(\)](#) that computes and returns model-fit criteria. See [add_model_criterion\(\)](#) for details and available arguments.

In addition to the criteria supported by [add_model_criterion\(\)](#) ('loo', 'waic', 'kfold', 'loo_subsample', 'bayes_R2', 'loo_R2', 'marglik'), get_model_criterion() also computes conditional and marginal versions of:

- Bayesian R2 (bayes_R2_conditional and bayes_R2_marginal) via [performance::r2_bayes\(\)](#), and
- LOO-adjusted Bayesian R2 (loo_R2_conditional and loo_R2_marginal) via [performance::r2_loo\(\)](#).

The LOO-adjusted R2 is conceptually analogous to the adjusted R2 in classical regression. The marginal R2 reflects the variance explained by the fixed effects alone, whereas the conditional R2 reflects the variance explained by both the fixed and random effects. See [performance::r2_bayes\(\)](#) for further details.

Usage

```
## S3 method for class 'bgmfit'
get_model_criterion(
  model,
  ...,
  criterion = "loo",
  ndraws = NULL,
  draw_ids = NULL,
  pointwise = FALSE,
  model_name = NULL,
  summary = TRUE,
  robust = FALSE,
  probs = c(0.025, 0.975),
  newdata = NULL,
  resp = NULL,
  cores = 1,
  expose_function = FALSE,
  verbose = FALSE,
  reformat = NULL,
  tibble_table = FALSE,
  print_table = FALSE,
  digits = 3,
  add_attr = FALSE
)
```

```

get_model_criterion(model, ...)

## S3 method for class 'list'
get_model_criterion(model, ...)

## S3 method for class 'character'
get_model_criterion(model, ...)

## Default S3 method:
get_model_criterion(model, ...)

```

Arguments

<code>model</code>	An R object of class <code>bsitar</code> or a list of models.
<code>...</code>	Additional arguments passed to <code>add_model_criterion()</code> .
<code>criterion</code>	The name of the criterion to be extracted from <code>brmsfit</code> objects.
<code>ndraws</code>	A positive integer indicating the number of posterior draws to use in estimation. If <code>NULL</code> (default), all draws are used.
<code>draw_ids</code>	An integer specifying the specific posterior draw(s) to use in estimation (default <code>NULL</code>).
<code>pointwise</code>	A flag indicating whether to compute the full log-likelihood matrix at once or separately for each observation. The latter approach is usually considerably slower but requires much less working memory. Accordingly, if one runs into memory issues, <code>pointwise = TRUE</code> is the way to go.
<code>model_name</code>	Optional name of the model. If <code>NULL</code> (the default) the name is taken from the call to <code>x</code> .
<code>summary</code>	A logical value indicating whether only the estimate should be computed (<code>TRUE</code>), or whether the estimate along with SE and CI should be returned (<code>FALSE</code> , default). Setting <code>summary</code> to <code>FALSE</code> will increase computation time. Note that <code>summary = FALSE</code> is required to obtain correct estimates when <code>re_formula = NULL</code> .
<code>robust</code>	A logical value to specify the summary options. If <code>FALSE</code> (default), the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If <code>TRUE</code> , the median and median absolute deviation (MAD) are applied instead. Ignored if <code>summary</code> is <code>FALSE</code> .
<code>probs</code>	The percentiles to be computed by the <code>quantile</code> function. Only used if <code>summary</code> is <code>TRUE</code> .
<code>newdata</code>	An optional data frame for estimation. If <code>NULL</code> (default), <code>newdata</code> is retrieved from the model.
<code>resp</code>	A character string (default <code>NULL</code>) to specify the response variable when processing posterior draws for <code>univariate_by</code> and <code>multivariate</code> models. See <code>bsitar()</code> for details on <code>univariate_by</code> and <code>multivariate</code> models.
<code>cores</code>	An integer specifying the number of cores for parallel execution. • If <code>NULL</code> (default), the number of cores is set to <code>future::availableCores() - 1</code>.

	On non-Windows systems, this can be controlled globally via the <code>mc.cores</code> option.
<code>expose_function</code>	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 <code>bayes_R2</code> during model optimization. - FALSE (default): Stan functions are not exposed. - NULL: The setting is inherited from the original <code>bsitar()</code> model fit configuration. Note: In the <code>optimize_model()</code> function, the default is NULL (inheriting behavior), whereas other post-processing functions default to FALSE.
<code>verbose</code>	A logical (default FALSE) to print some useful information.
<code>reformat</code>	Logical indicating whether to round numeric variables in the output <code>data.frame</code> using <code>base::round()</code> . The default is NULL, which is treated as TRUE. When TRUE, numeric variables are rounded to the number of decimal places specified by <code>digits</code> .
<code>tibble_table</code>	Logical indicating whether to return the table as a <code>data.frame</code> (FALSE) or as a tibble (TRUE). Default is FALSE.
<code>print_table</code>	Logical indicating whether to print the table using <code>knitr::kable()</code> (TRUE) or return it as an object (FALSE). When <code>print_table = TRUE</code> , the table is printed and the function returns <code>invisible(NULL)</code> . Default is FALSE.
<code>digits</code>	An integer (default 2) to set the decimal places for rounding the results using the <code>base::round()</code> function.
<code>add_attr</code>	Logical indicating whether to attach complex list elements of the criteria as attributes to the returned <code>data.frame</code> . Default is FALSE.

Value

A `data.frame`. If `add_attr = TRUE`, an additional attribute named "attr_object" is attached to the returned data frame. This attribute contains the complex, nested list components from `add_model_criterion()` that cannot be represented as regular columns.

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)

# For illustration purposes, we make a copy of model with itself
# In the example below, get_model_criterion() should indicate no difference
# between model_1 and model_2 as both these models are exactly identical
model_1 <- model
model_2 <- model

# Add model fit criteria (e.g., WAIC).
out_1 <- get_model_criterion(model_1, criterion = c("waic"))
out_2 <- get_model_criterion(model_2, criterion = c("loo"))

# compare models model_1 and model_2
out_12 <- get_model_criterion(model_1, model_2, criterion = c("waic"))

# compare models model_1 and model_2 - model names: "mods[[1L]]" "mods[[2L]]"
mods <- list(model_1, model_2)
out_12 <- get_model_criterion(mods)

# compare models model_1 and model_2 - model names extracted as such
# Note that list() could be supplied as a string "list()" also.
out_12 <- get_model_criterion(list(model_1, model_2))
out_12 <- get_model_criterion("list(model_1, model_2)")

# compare different versions of R2, from brms and performance packages
set_criterion <- c("bayes_R2", "loo_R2",
  "bayes_R2_conditional", "bayes_R2_marginal",
  "loo_R2_conditional", "loo_R2_marginal")
out_R2 <- get_model_criterion(model_2, criterion = set_criterion)

```

```
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

<code>model</code>	An object of class <code>bgmfit</code> .
<code>resp</code>	A character string (default <code>NULL</code>) to specify the response variable when processing posterior draws for <code>univariate_by</code> and <code>multivariate</code> models. See bsitar() for details on <code>univariate_by</code> and <code>multivariate</code> models.
<code>dpar</code>	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
<code>ndraws</code>	A positive integer indicating the number of posterior draws to use in estimation. If <code>NULL</code> (default), all draws are used.
<code>draw_ids</code>	An integer specifying the specific posterior draw(s) to use in estimation (default <code>NULL</code>).
<code>newdata</code>	An optional data frame for estimation. If <code>NULL</code> (default), <code>newdata</code> 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 <code>datagrid = list()</code> , in which case essential arguments like <code>model</code> and <code>newdata</code> are inferred from the respective arguments specified elsewhere. Additionally, the first-level predictor (such as <code>age</code>) and any covariates included in the model (e.g., <code>gender</code>) are automatically inferred from the <code>model</code> 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 <code>numeric_cov_at</code> 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 <code>newdata</code> . 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 <code>newdata</code> is provided.
sample_new_levels	Indicates how to sample new levels for grouping factors specified in <code>re_formula</code> . This argument is only relevant if <code>newdata</code> is provided and <code>allow_new_levels</code> is set to 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 were observed in the <code>old_data</code> . 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
- 'spgv': Size (length/height) at peak growth velocity age, 'apgv'
- 'stgv': Size at takeoff growth velocity age, 'atgv'
- 'scgv': Size at cessation growth velocity age, 'acgv'
- 'satXX': Size at specific age XX years of predictor xvar (e.g., 'sat15' = size at 15 years, 'sat12.5' = size at 12.5 years)
- 'all': All six primary velocity/age parameters ('apgv', 'pgv', 'atgv', 'tgv', 'acgv', 'cgv'). Cannot be used when 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 <code>ipts</code> . By default, NULL sets the individual-specific predictor range. Setting <code>xrange = 1</code> applies the same range for individuals within the same higher grouping variable (e.g., study). Setting <code>xrange = 2</code> applies an identical range across the entire sample. Alternatively, a numeric vector (e.g., <code>xrange = c(6, 20)</code>) 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 <code>cgv</code> and <code>acgv</code> growth parameters. The <code>acg_velocity</code> should be greater than 0 and less than 1. The default value of <code>acg_velocity = 0.10</code> 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 <code>base::round()</code> 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, <code>numeric_cov_at = list(xx = 2)</code> will set the continuous covariate variable 'xx' to 2. The argument <code>numeric_cov_at</code> 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 <code>ipts</code> 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 <code>aux_variables</code> .

grid_add	An optional argument to specify the variable(s) that can be passed to the <code>marginalEffects::datagrid()</code> . 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 <code>marginalEffects::datagrid()</code> .
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. <i>Note:</i> 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 <code>seq()</code> , 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: <code>fitted_draws()</code> , <code>predict_draws()</code> , <code>growthparameters()</code> , <code>plot_curves()</code> , <code>get_predictions()</code> , <code>get_comparisons()</code> , and <code>get_growthparameters()</code> .

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 <code>future.apply::future_sapply()</code> .
future_session	A character string or a named list specifying the parallel plan when <code>future = TRUE</code>. • Character string: Defaults to "multisession". Use "multicore" for forking (not supported on Windows). • Named list: Useful for advanced plans like <code>future.mirai::mirai_cluster()</code>. The list must contain: – <code>future_session</code>: The planner function (e.g., <code>mirai_cluster</code>). – Additional named arguments passed to the planner (e.g., <code>daemons</code> for <code>mirai::daemons()</code>). Example: <code>list(future_session = mirai_cluster, daemons = list(...))</code>.
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 <code>future::plan("multicore")</code>. Input options: • List: (e.g., <code>list(1:6, 7:10)</code>). Each element is a sequence of numbers passed to <code>draw_ids</code> to process in a separate chunk. • Numeric vector of length 2: (e.g., <code>c(100, 4)</code>). The first element is the total number of draws, and the second is the number of splits. Indices are generated via <code>parallel::splitIndices()</code>. • Numeric vector of length 1: Specifies the total number of draws. Splits are calculated automatically. • TRUE: Automatically creates splits based on <code>ndraws</code> and <code>cores</code>. If both <code>ndraws</code> and <code>draw_ids</code> are specified, <code>draw_ids</code> takes precedence. This is consistent with post-processing functions in the <code>bsitar</code> and <code>brms</code> packages. • FALSE: Splitting is disabled. Note: On Windows with <code>future::plan("multisession")</code>, background R processes may not free resources automatically. If needed, use <code>installr::kill_all_Rscript_s()</code> to terminate them.
future_method	A character string (default 'future') specifying the parallel computation backend. Options include: • 'future': Uses <code>future.apply::future_lapply()</code> for execution. • 'dofuture': Uses <code>foreach::foreach()</code> via the <code>doFuture</code> package.
future_re_expose	A logical value (default NULL) indicating whether to re-expose internal Stan functions when <code>future = TRUE</code>. This is critical when <code>future::plan()</code> 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 <code>future::availableCores() - 1</code>. • On non-Windows systems, this can be controlled globally via the <code>mc.cores</code> 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 <code>marginaleffects::predictions()</code> or <code>marginaleffects::avg_predictions()</code> function. If FALSE (default), <code>marginaleffects::predictions()</code> is called; otherwise, <code>marginaleffects::avg_predictions()</code> is used when average = TRUE.
plot	A logical specifying whether to plot predictions by calling <code>marginaleffects::plot_predictions()</code> (TRUE) or not (FALSE). If FALSE (default), <code>marginaleffects::predictions()</code> or <code>marginaleffects::avg_predictions()</code> are called to compute predictions (see average for details). Note that <code>marginaleffects::plot_predictions()</code> 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., <code>geom_ribbon</code>, then include <code>rm_geom_ribbon = TRUE</code>. 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: <pre>mapping_facet = list(group = 'id', colour = 'id', facet_wrap = c('study'), rm_geom_ribbon = TRUE).</pre>
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 <code>?stats::fivenum</code>.
deriv	An integer to indicate whether to estimate the distance curve or its derivative (i.e., velocity curve). The <code>deriv = 0</code> (default) is for the distance curve, whereas <code>deriv = 1</code> 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 <code>model_deriv = TRUE</code> for functions that require the velocity curve, such as <code>growthparameters()</code> and <code>plot_curves()</code> . Set it to <code>NULL</code> for functions that use the distance curve (i.e., fitted values), such as <code>loo_validation()</code> and <code>plot_ppc()</code> .
method	A character string specifying the computation method: whether to use the marginaleffects machinery at the post-draw stage, i.e., <code>marginaleffects::comparisons()</code> (<code>method = 'pkg'</code>) or to use custom functions for efficiency and speed (<code>method = 'custom'</code> , default). Note that <code>method = 'custom'</code> is particularly useful when testing hypotheses. Also, when <code>method = 'custom'</code> , <code>marginaleffects::predictions()</code> is used internally instead of <code>marginaleffects::comparisons()</code> .
marginals	A list, data.frame, or tibble of velocity returned by the marginaleffects functions (default <code>NULL</code>). This is only evaluated when <code>method = 'custom'</code> . The marginals can be the output from marginaleffects functions or posterior draws from <code>marginaleffects::posterior_draws()</code> . The <code>marginals</code> 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 <code>marginalEffects::posterior_draws()</code>.
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 <code>marginaleffects::predictions()</code> function and should not be confused with the transform argument from the deprecated <code>brms::posterior_predict()</code> function. It's important to note that for both <code>marginaleffects::predictions()</code> and <code>marginaleffects::avg_predictions()</code>, 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.
wt	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 wt, 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 wt=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
- $b1 \geq 10$
- 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 <code>x</code>: object produced by a <code>marginalEffects</code> function or a data frame with column <code>rowid</code> and <code>estimate</code> – Returns a data frame with columns <code>term</code> and <code>estimate</code> (mandatory) and <code>rowid</code> (optional). – The function can also accept optional input arguments: <code>newdata</code>, <code>by</code>, <code>draws</code>. – This function approach will not work for Bayesian models or with bootstrapping. In those cases, it is easy to use <code>get_draws()</code> to extract and manipulate the draws directly. • See the Examples section below and the vignette: https://marginaleffects.com/chapters/hypothesis.html • Warning: When calling <code>predictions()</code> with <code>type="invlink(link)"</code> (the default in some models), hypothesis is tested and p values are computed on the link scale.
<code>equivalence</code>	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.
<code>eps</code>	NULL or numeric value which determines the step size to use when calculating numerical derivatives: $(f(x+eps)-f(x))/eps$. When <code>eps</code> 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 <code>eps</code> may be necessary to avoid numerical problems in certain models.
<code>constrats_by</code>	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 <code>constrats_by</code> should be a subset of the variables included in the <code>'by'</code> argument.
<code>constrats_at</code>	A character vector (default NULL) specifying the variable(s) at which hypotheses (at the post-draw stage) should be tested. <code>constrats_at</code> 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.
<code>reformat</code>	A logical (default TRUE) indicating whether to reformat the output returned by <code>marginaleffects</code> as a data frame. Column names are redefined as <code>conf.low</code> to Q2.5 and <code>conf.high</code> to Q97.5 (assuming <code>conf_int = 0.95</code>). Additionally, some columns (<code>term</code> , <code>contrast</code> , etc.) are dropped from the data frame.
<code>estimate_center</code>	A character string (default NULL) specifying how to center estimates: either <code>'mean'</code> or <code>'median'</code> . This option sets the global options as follows: <code>options("marginaleffects_posterior_center" = "mean")</code> or <code>options("marginaleffects_posterior_center" = "median")</code> . These global options are restored upon function exit using <code>base::on.exit()</code> .
<code>estimate_interval</code>	A character string (default NULL) to specify the type of credible intervals: <code>'eti'</code> for equal-tailed intervals or <code>'hdi'</code> for highest density intervals. This option sets the global options as follows: <code>options("marginaleffects_posterior_interval" = "eti")</code> or <code>options("marginaleffects_posterior_interval" = "hdi")</code> , and is restored on exit using <code>base::on.exit()</code> .

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 <code>xvar</code> itself is rarely used because <code>xvar</code> is inferred internally. A use case is when conflicting variables exist (e.g., <code>sigma_formula</code>) and user wants to set a specific variable as 'x'.
<code>idvar</code>	A character string (default <code>NULL</code>) specifying the 'id' variable. Rarely used because <code>idvar</code> is inferred internally.
<code>itransform</code>	A character string (default <code>NULL</code>) indicating the variables names that are reverse transformed. Options are <code>c("x", "y", "sigma")</code> . The <code>itransform</code> is primarily used to get the <code>xvar</code> variable at original scale i.e., <code>itransform = 'x'</code> . To turn off all transformations, use <code>itransform = ""</code> . when <code>itransform = NULL</code> , the appropriate transformation for <code>xvar</code> is selected automatically. Note that when no match for <code>xvar</code> is found in the data, frame, the <code>itransform</code> will be ignored within the calling function, <code>'prepare_transformations()'</code> .
<code>newdata_fixed</code>	An indicator specifying how to handle the format and structure of the user-provided <code>newdata</code> before use. The available options are: • <code>NULL</code> (default): Checks the data format and structure of <code>newdata</code>, and applies <code>prepare_data2</code> and <code>prepare_transformations</code> as needed. • <code>TRUE</code>: Returns <code>newdata</code> as-is, without any format checking or transformation. • <code>0</code>: Same as <code>TRUE</code> — returns <code>newdata</code> without checking the data format or applying <code>prepare_data2</code> and <code>prepare_transformations</code>. • <code>1</code>: Checks the data format and, if needed, prepares it using <code>prepare_data2</code>, but does not apply <code>prepare_transformations</code>. • <code>2</code>: Applies <code>prepare_transformations</code> only, assuming the data format is already correct. It is strongly recommended to leave both <code>newdata = NULL</code> and <code>newdata_fixed = NULL</code> (the defaults). In this case, the data used during model fitting is automatically retrieved, checked for the required format, and transformed as needed via <code>prepare_data2</code> and <code>prepare_transformations</code>. The flags <code>newdata_fixed = 0, 1, and 2</code> are intended primarily for internal use.
<code>envir</code>	The environment used for function evaluation. The default is <code>NULL</code> , which sets the environment to <code>parent.frame()</code> . Since most post-processing functions rely on brms , it is recommended to set <code>envir = globalenv()</code> or <code>envir = .GlobalEnv</code> , especially for derivatives like velocity curves.
<code>...</code>	Additional arguments passed to the <code>brms::fitted.brmsfit()</code> function. Please see <code>brms::fitted.brmsfit()</code> 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 <code>bgmfit</code> .
newdata	An optional data frame for estimation. If <code>NULL</code> (default), <code>newdata</code> is retrieved from the model.
resp	A character string (default <code>NULL</code>) to specify the response variable when processing posterior draws for <code>univariate_by</code> and <code>multivariate</code> models. See bsitar() for details on <code>univariate_by</code> and <code>multivariate</code> 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 <code>NULL</code> (default), all draws are used.
draw_ids	An integer specifying the specific posterior draw(s) to use in estimation (default <code>NULL</code>).
summary	A logical value indicating whether only the estimate should be computed (<code>TRUE</code>), or whether the estimate along with SE and CI should be returned (<code>FALSE</code> , default). Setting <code>summary</code> to <code>FALSE</code> will increase computation time. Note that <code>summary = FALSE</code> is required to obtain correct estimates when <code>re_formula = NULL</code> .
robust	A logical value to specify the summary options. If <code>FALSE</code> (default), the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If <code>TRUE</code> , the median and median absolute deviation (MAD) are applied instead. Ignored if <code>summary</code> is <code>FALSE</code> .
transform_draws	A function applied to individual draws from the posterior distribution before computing summaries (default <code>NULL</code>). The argument <code>transform_draws</code> is derived from the marginaleffects::predictions() function and should not be confused with the <code>transform</code> argument from the deprecated brms::posterior_predict() function. It's important to note that for both marginaleffects::predictions() and marginaleffects::avg_predictions() , the <code>transform_draws</code> argument takes precedence over the <code>transform</code> argument. Note that when <code>transform_draws = NULL</code> , an attempt is made to automatically set <code>transform_draws = 'exp'</code> for <code>dpar = 'sigma'</code> . User can set <code>transform_draws = FALSE</code> to turn off this automatic assignment of 'exp' to the <code>transform_draws</code> . It is also important to set <code>transform_draws = FALSE</code> when computing the first derivative (velocity) for <code>dpar = 'sigma'</code> .

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 <code>numeric_cov_at</code> 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 <code>acgv_velocity</code> argument (see below). The <code>acgv_velocity</code> 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 <code>acgv</code> . 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 <code>fitted_draws()</code> , while the 'predict' method calls <code>predict_draws()</code> . See <code>brms::fitted.brmsfit()</code> and <code>brms::predict.brmsfit()</code> for details.
allow_new_levels	A flag indicating if new levels of group-level effects are allowed (defaults to FALSE). Only relevant if <code>newdata</code> is provided.
sample_new_levels	Indicates how to sample new levels for grouping factors specified in <code>re_formula</code> . This argument is only relevant if <code>newdata</code> is provided and <code>allow_new_levels</code> is set to 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 were 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 <code>ipts</code> 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 <code>aux_variables</code> .
grid_add	An optional argument to specify the variable(s) that can be passed to the <code>marginaleffects::datagrid()</code> . 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 <code>grid_add</code> . Note that unlike <code>aux_variables</code> which are passed to the internal data functions such as <code>'get.newdata'</code> , the <code>grid_add</code> are passed to the <code>marginaleffects::datagrid()</code> .
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 <code>length.out</code> argument for <code>seq()</code> , controlling the smoothness of distance and velocity curves without altering the predictor range.
	NULL (the default) Engages automatic behavior based on the <code>dpar</code> argument: it is internally set to 50 for the mean response (<code>dpar = 'mu'</code>), ensuring smooth curves, while for the distributional parameters (e.g., <code>dpar = 'sigma'</code>), 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: <code>fitted_draws()</code>, <code>predict_draws()</code>, <code>growthparameters()</code>, <code>plot_curves()</code>, <code>get_predictions()</code>, <code>get_comparisons()</code>, and <code>get_growthparameters()</code>.
model_deriv	A logical value specifying whether to estimate the velocity curve from the derivative function or by differentiating the distance curve. Set <code>model_deriv = TRUE</code> for functions that require the velocity curve, such as <code>growthparameters()</code> and <code>plot_curves()</code> . Set it to <code>NULL</code> for functions that use the distance curve (i.e., fitted values), such as <code>loo_validation()</code> and <code>plot_ppc()</code> .
conf	A numeric value (default 0.95) to compute the confidence interval (CI). Internally, <code>conf</code> is translated into paired probability values as <code>c((1 - conf)/2, 1 - (1 - conf)/2)</code> . For <code>conf = 0.95</code> , this computes a 95% CI where the lower and upper limits are named <code>Q.2.5</code> and <code>Q.97.5</code> , respectively.
xrange	An integer to set the predictor range (e.g., age) when executing the interpolation via <code>ipts</code> . By default, <code>NULL</code> sets the individual-specific predictor range. Setting <code>xrange = 1</code> applies the same range for individuals within the same higher grouping variable (e.g., study). Setting <code>xrange = 2</code> applies an identical range across the entire sample. Alternatively, a numeric vector (e.g., <code>xrange = c(6, 20)</code>) 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 <code>xrange_search = NULL</code> .
digits	An integer (default 2) to set the decimal places for rounding the results using the <code>base::round()</code> 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 <code>future.apply::future_sapply()</code> .
future_session	A character string or a named list specifying the parallel plan when <code>future = TRUE</code>. • Character string: Defaults to "multisession". Use "multicore" for forking (not supported on Windows). • Named list: Useful for advanced plans like <code>future.mirai::mirai_cluster()</code>. The list must contain: – <code>future_session</code>: The planner function (e.g., <code>mirai_cluster</code>). – Additional named arguments passed to the planner (e.g., <code>daemons</code> for <code>mirai::daemons()</code>). Example: <code>list(future_session = mirai_cluster, daemons = list(...))</code>.
cores	An integer specifying the number of cores for parallel execution. • If <code>NULL</code> (default), the number of cores is set to <code>future::availableCores() - 1</code>.

	On non-Windows systems, this can be controlled globally via the <code>mc.cores</code> option.
<code>parms_eval</code>	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.
<code>idata_method</code>	A character string specifying the interpolation method (default NULL). The number of interpolation points is controlled by the <code>ipts</code> argument. Available options: 'm1': Adapted from the iapvbs package (documented here). This method internally constructs the data frame based on the model configuration. <i>Note:</i> This method may fail if the model includes covariates (especially in <code>univariate_by</code> models). 'm2': Based on the JMbayes package (documented here). This method uses the exact data frame from the model fit (<code>fit\$data</code>) and is generally more robust. If <code>idata_method = NULL</code>, 'm2' is automatically selected. It is recommended to use 'm2' if 'm1' encounters errors with covariate-dependent models.
<code>parms_method</code>	A character string specifying the method used when evaluating <code>parms_eval</code> . The default method is <code>getPeak</code> , which uses the <code>sitar::getPeak()</code> function from the <code>sitar</code> package. Alternatively, <code>findpeaks</code> uses the <code>findpeaks</code> function from the <code>pracma</code> package. This parameter is for internal use and ensures compatibility across internal functions.
<code>verbose</code>	A logical argument (default FALSE) to specify whether to print information collected during the setup of the object(s).
<code>fullframe</code>	A logical value indicating whether to return a <code>fullframe</code> object in which <code>newdata</code> is bound to the summary estimates. Note that <code>fullframe</code> cannot be used with <code>summary = FALSE</code> , and it is only applicable when <code>idata_method = 'm2'</code> . A typical use case is when fitting a <code>univariate_by</code> model. This option is mainly for internal use.
<code>dummy_to_factor</code>	A named list (default NULL) to convert dummy variables into a factor variable. The list must include the following elements: <code>factor.dummy</code>: A character vector of dummy variables to be converted to factors. <code>factor.name</code>: The name for the newly created factor variable (default is 'factor.var' if NULL). <code>factor.level</code>: A vector specifying the factor levels. If NULL, levels are taken from <code>factor.dummy</code>. If <code>factor.level</code> is provided, its length must match <code>factor.dummy</code>.
<code>expose_function</code>	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 <code>fit.criteria</code> or <code>bayes_R2</code> 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.

<code>usesavedfuns</code>	A logical value (default NULL) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the <code>expose_functions</code> argument from the <code>bsitar()</code> call. Manual specification of <code>usesavedfuns</code> is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.
<code>clearenvfuns</code>	A logical value indicating whether to clear the exposed Stan functions from the environment (TRUE) or not (FALSE). If NULL, <code>clearenvfuns</code> is set based on the value of <code>usesavedfuns</code> : TRUE if <code>usesavedfuns</code> = TRUE, or FALSE if <code>usesavedfuns</code> = FALSE.
<code>funlist</code>	A list (default NULL) specifying function names. This is rarely needed, as required functions are typically retrieved automatically. A use case for <code>funlist</code> is when <code>sigma_formula</code> , <code>sigma_formula_gr</code> , or <code>sigma_formula_gr_str</code> use an external function (e.g., <code>poly(age)</code>). The <code>funlist</code> should include function names defined in the <code>globalenv()</code> . For functions needing both distance and velocity curves (e.g., <code>plot_curves(..., opt = 'dv')</code>), <code>funlist</code> must include two functions: one for the distance curve and one for the velocity curve.
<code>xvar</code>	A character string (default NULL) specifying the 'x' variable. Rarely used because <code>xvar</code> is inferred internally. A use case is when conflicting variables exist (e.g., <code>sigma_formula</code>) and user wants to set a specific variable as 'x'.
<code>difx</code>	A character string (default NULL) specifying the 'x' variable that should be used for manual differentiation of the distance curve. Internally, the <code>xvar</code> is set as <code>difx</code> if specified. The argument <code>difx</code> is evaluated only when <code>dpar</code> = 'sigma', ignored otherwise. Note that argument <code>xvar</code> itself is rarely used because <code>xvar</code> is inferred internally. A use case is when conflicting variables exist (e.g., <code>sigma_formula</code>) and user wants to set a specific variable as 'x'.
<code>idvar</code>	A character string (default NULL) specifying the 'id' variable. Rarely used because <code>idvar</code> is inferred internally.
<code>itransform</code>	A character string (default NULL) indicating the variables names that are reverse transformed. Options are <code>c("x", "y", "sigma")</code> . The <code>itransform</code> is primarily used to get the <code>xvar</code> variable at original scale i.e., <code>itransform</code> = 'x'. To turn off all transformations, use <code>itransform</code> = "". when <code>itransform</code> = NULL, the appropriate transformation for <code>xvar</code> is selected automatically. Note that when no match for <code>xvar</code> is found in the data, frame, the <code>itransform</code> will be ignored within the calling function, <code>'prepare_transformations()'</code> .
<code>newdata_fixed</code>	An indicator specifying how to handle the format and structure of the user-provided <code>newdata</code> before use. The available options are: • NULL (default): Checks the data format and structure of <code>newdata</code>, and applies <code>prepare_data2</code> and <code>prepare_transformations</code> as needed. • TRUE: Returns <code>newdata</code> as-is, without any format checking or transformation.

- 0: Same as TRUE — returns newdata without checking the data format or applying prepare_data2 and prepare_transformations.
- 1: Checks the data format and, if needed, prepares it using prepare_data2, but does not apply prepare_transformations.
- 2: Applies prepare_transformations only, assuming the data format is already correct.

It is strongly recommended to leave both newdata = NULL and newdata_fixed = NULL (the defaults). In this case, the data used during model fitting is automatically retrieved, checked for the required format, and transformed as needed via prepare_data2 and prepare_transformations. The flags newdata_fixed = 0, 1, and 2 are intended primarily for internal use.

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,
    drop_probtitles = NULL,
    verbose = FALSE,
    envir = NULL,
    ...
)

hypothesis_test(model, ...)

```

Arguments

model	A fitted model object of class <code>bgmfit</code> , 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 <code>coef(object)</code> uniquely identify estimates, they can be used in the formula. Otherwise, use <code>b1</code>, <code>b2</code>, etc. to identify the position of each parameter. The <code>b*</code> 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: – <code>hp = drat</code> – <code>hp + drat = 12</code> – <code>b1 + b2 + b3 = 0</code> – <code>b* / b1 = 1</code> – <code><= 0</code> – <code>>= -3.5</code>

- `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 <code>brms::hypothesis()</code> . This renaming is done to avoid conflicts with the hypothesis argument for the <code>marginaleffects::comparison</code>
parameters	Regular expression pattern that describes the parameters that should be returned. Meta-parameters (like <code>lp__</code> or <code>prior_</code>) are filtered by default, so only parameters that typically appear in the <code>summary()</code> 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 • 'spgv': Size (length/height) at peak growth velocity age, 'apgv' • 'stgv': Size at takeoff growth velocity age, 'atgv' • 'scgv': Size at cessation growth velocity age, 'acgv' • 'satXX': Size at specific age XX years of predictor xvar (e.g., 'sat15' = size at 15 years, 'sat12.5' = size at 12.5 years) • 'all': All six primary velocity/age parameters ('apgv', 'pgv', 'atgv', 'tgv', 'acgv', 'cgv'). Cannot be used when <code>by = TRUE</code>. Multiple parameters can be requested simultaneously as a vector, e.g., <code>c('apgv', 'pgv', 'spgv')</code>. 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 "mbccombo". • 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 "mbcambio".
- `na.rm`: If TRUE (default), remove NA values.
- `inline`: Internal use only; executes custom equivalence function (not for users).

<code>rope_test</code>	Logical; if TRUE, run ROPE-based practical equivalence testing via <code>bayestestR::equivalence_test()</code> . This is rarely used because appropriate setting TRUE/FALSE is assigned automatically depending on the <code>equivalence_test</code> . Note that when <code>equivalence_test</code> is used to set up the <code>rope_test</code> , then all missing arguments to the <code>equivalence_test</code> list are assigned from the default or user defined arguments included in the <code>hypothesis_test()</code> call. See <code>equivalence_test</code> for details.
<code>pd_test</code>	Logical; if TRUE, compute probability of direction via <code>bayestestR::p_direction()</code> . This is rarely used because appropriate setting TRUE/FALSE is assigned automatically depending on the <code>p_direction</code> . Note that when <code>p_direction</code> is used to set up the <code>pd_test</code> , then all missing arguments to the <code>p_direction</code> list are assigned from the default or user defined arguments included in the <code>hypothesis_test()</code> call. See <code>p_direction</code> for details.
<code>range</code>	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., <code>c(-0.1, 0.1)</code>), • a list of numeric vector of the same length as numbers of parameters (see 'Examples'). • a list of <i>named</i> 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 <code>c(-0.1, 0.1)</code>. If "default" and input is a Bayesian model, <code>rope_range()</code> is used. See 'Examples'.
<code>method</code>	Can be "direct" or one of methods of <code>estimate_density()</code> , such as "kernel", "logspline" or "KernSmooth". See details.
<code>method_call</code>	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 <code>pd_to_p()</code> .
remove_na	Should missing values be removed before computation? Note that Inf (infinity) are <i>not</i> removed.
rvar_col	A single character - the name of an rvar column in the data frame to be processed. See example in <code>p_direction()</code> .
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 <i>Model Components</i> . May be abbreviated. Note that the <i>conditional</i> component also refers to the <i>count</i> or <i>mean</i> component - names may differ, depending on the modeling package. There are three convenient shortcuts (not applicable to <i>all</i> 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 <code>marginalEffects::comparisons()</code> via the <code>get_growthparameters()</code> . 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 <code>bayestestR::equivalence_test()</code>), and null (for <code>bayestestR::p_direction()</code>). 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 <code>marginalEffects::comparisons()</code> via the <code>getGrowthParameters()</code> . 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 <code>bayestestR::equivalence_test()</code>), and null (for <code>bayestestR::p_direction()</code>). 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 <code>bayestestR::equivalence_test()</code> 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 <code>bayestestR::equivalence_test()</code> .
reformat	Logical; if TRUE, post-process the data frame returned by <code>marginalEffects::comparisons()</code> to make it consistent with posterior-summary conventions. This includes renaming interval columns such as <code>conf.low</code> to <code>Q2.5</code> and <code>conf.high</code> to <code>Q97.5</code> , 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: <code>options("marginalEffects_posterior_center" = "mean")</code> or <code>options("marginalEffects_posterior_center" = "median")</code> . These global options are restored upon function exit using <code>base::on.exit()</code> .
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: <code>options("marginaleffects_posterior_interval" = "eti")</code> or <code>options("marginaleffects_posterior_interval" = "hdi")</code> , and is restored on exit using <code>base::on.exit()</code> .
<code>na.rm</code>	Logical; if TRUE (default), then remove NA values.
<code>alpha</code>	Alpha level for hypothesis tests used in <code>brms::hypothesis()</code> . 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 <code>brms::hypothesis()</code> .
<code>ci</code>	Confidence level used in <code>bayestestR::equivalence_test()</code> and <code>bayestestR::p_direction()</code> . 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.
<code>conf_level</code>	Confidence level used in <code>marginaleffects::comparisons()</code> . The default is NULL, in which case <code>conf_level</code> is inferred from probs. With the default probs, this corresponds to $conf_level = 0.95$, which is also the default in <code>marginaleffects::comparisons()</code> . Note that if <code>probs = NULL</code> , then <code>conf_level</code> is used to determine the probs as well as the alpha and ci (see below).
<code>probs</code>	Probability levels for credible intervals. Defaults to <code>c(0.025, 0.975)</code> (95% values for alpha, ci, and <code>conf_level</code> when those are NULL (see below).
<code>get_range_null_form</code>	Logical; if TRUE, return the expected structure of range (<code>bayestestR::equivalence_test()</code>) and null (<code>bayestestR::p_direction()</code>) values. User can use this format to set the range and null values that are then passed to the <code>bayestestR::equivalence_test()</code> and <code>bayestestR::equivalence_test()</code> via <code>comparison_range_null</code> and <code>hypothesis_range_null</code> .
<code>get_range_null_value</code>	Logical; if TRUE, return the default range (<code>bayestestR::equivalence_test()</code>) and null (<code>bayestestR::p_direction()</code>) values that are used to set up the range and null values for <code>bayestestR::equivalence_test()</code> and <code>bayestestR::equivalence_test()</code> .
<code>plot</code>	Logical or a character string "return"; if TRUE, plot the hypothesis test results and return the data frame. Set <code>plot = "return"</code> to return the plot object itself instead of the data frame. Note that only <code>brms::hypothesis()</code> and <code>bayestestR::equivalence_test()</code> support plotting.
<code>digits</code>	Number of digits to use when printing numeric results. Default 2.
<code>engine</code>	Optional engine selector specifying which backend(s) to use: • "brms" for <code>brms::hypothesis()</code> non-linear hypothesis testing • "bayestestR" for <code>bayestestR::equivalence_test()</code> (ROPE) and <code>bayestestR::p_direction()</code> (PD) testing • "marginaleffects" for <code>marginaleffects::comparisons()</code> pairwise/contrast comparisons • "mbcombo" for combined marginaleffects and bayestestR workflows • NULL (default) automatically selects appropriate engine(s) based on model class and requested analyses
<code>usesavedfuns</code>	A logical value (default NULL) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the <code>expose_functions</code> argument from the <code>bsitar()</code> call. Manual specification of <code>usesavedfuns</code> is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.

drop_probtitles	A logical indicating whether to drop the CI columns (such as 'Q2.5' and 'Q97.5') from the output. The default is NULL which sets drop_probtitles = TRUE if output includes HDI_range variable from the 'equivalence_test'. when drop_probtitles = TRUE, the column 'HDI_range' is relocated next to the 'Estimate' column. Also, the column indicating the confidence level such as 0.95 is also removed.
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

[illegible]

```

# 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 object is of class bgmfit

Description

Checks if object is of class bgmfit

Usage

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

Arguments

x An R object

Author(s)

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

`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

<code>model</code>	An object of class <code>bgmfit</code> .
<code>compare</code>	A logical flag indicating if the information criteria of the models should be compared using <code>loo::loo_compare()</code> .
<code>resp</code>	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 <code>loo::loo_moment_match()</code> should be applied to problematic observations. Defaults to FALSE. For most models, moment matching will only work if <code>save_pars = save_pars(all = TRUE)</code> was set when fitting the model with <code>brms::brm()</code> . See <code>brms::loo_moment_match()</code> for more details.
reloo	A logical flag indicating whether <code>brms::reloo()</code> should be applied to problematic observations. Defaults to FALSE.
k_threshold	The Pareto k threshold for which observations <code>loo_moment_match</code> or <code>reloo</code> is applied if argument <code>moment_match</code> or <code>reloo</code> is TRUE. Defaults to 0.7. See <code>pareto_k_ids</code> for more details.
save_psis	Should the "psis" object created internally be saved in the returned object? For more details see <code>loo</code> .
moment_match_args	An optional list of additional arguments passed to <code>loo::loo_moment_match()</code> .
reloo_args	An optional list of additional arguments passed to <code>brms::reloo()</code> .
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 <code>future::availableCores() - 1</code>. • On non-Windows systems, this can be controlled globally via the <code>mc.cores</code> option.
model_deriv	A logical value specifying whether to estimate the velocity curve from the derivative function or by differentiating the distance curve. Set <code>model_deriv = TRUE</code> for functions that require the velocity curve, such as <code>growthparameters()</code> and <code>plot_curves()</code> . Set it to NULL for functions that use the distance curve (i.e., fitted values), such as <code>loo_validation()</code> and <code>plot_ppc()</code> .
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: • <code>factor.dummy</code>: A character vector of dummy variables to be converted to factors.

	• <code>factor.name</code>: The name for the newly created factor variable (default is <code>'factor.var'</code> if NULL). • <code>factor.level</code>: A vector specifying the factor levels. If NULL, levels are taken from <code>factor.dummy</code>. If <code>factor.level</code> is provided, its length must match <code>factor.dummy</code>.
<code>expose_function</code>	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 <code>bayes_R2</code> during model optimization. • FALSE (default): Stan functions are not exposed. • NULL: The setting is inherited from the original <code>bsitar()</code> model fit configuration. Note: In the <code>optimize_model()</code> function, the default is NULL (inheriting behavior), whereas other post-processing functions default to FALSE.
<code>usesavedfuns</code>	A logical value (default NULL) indicating whether to use already exposed and saved Stan functions. This is typically set automatically based on the <code>expose_functions</code> argument from the <code>bsitar()</code> call. Manual specification of <code>usesavedfuns</code> is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.
<code>clearenvfuns</code>	A logical value indicating whether to clear the exposed Stan functions from the environment (TRUE) or not (FALSE). If NULL, <code>clearenvfuns</code> is set based on the value of <code>usesavedfuns</code> : TRUE if <code>usesavedfuns = TRUE</code> , or FALSE if <code>usesavedfuns = FALSE</code> .
<code>envir</code>	The environment used for function evaluation. The default is NULL, which sets the environment to <code>parent.frame()</code> . Since most post-processing functions rely on brms , it is recommended to set <code>envir = globalenv()</code> or <code>envir = .GlobalEnv</code> , especially for derivatives like velocity curves.
<code>...</code>	Additional arguments passed to the <code>brms::loo()</code> function. Please see <code>brms::loo</code> 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, `L00` 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 <code>bgmfit</code> .
resp	A character string (default <code>NULL</code>) 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 <code>NULL</code> (default), all draws are used.
draw_ids	An integer specifying the specific posterior draw(s) to use in estimation (default <code>NULL</code>).
newdata	An optional data frame for estimation. If <code>NULL</code> (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 <code>NULL</code> , 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 <code>datagrid = list()</code>, in which case essential arguments like <code>model</code> and <code>newdata</code> 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 <code>model</code> object.
<code>re_formula</code>	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.
<code>newdata2</code>	A named list of objects containing new data, which cannot be passed via argument <code>newdata</code> . Required for some objects used in autocorrelation structures, or stanvars .
<code>allow_new_levels</code>	A flag indicating if new levels of group-level effects are allowed (defaults to FALSE). Only relevant if <code>newdata</code> is provided.
<code>sample_new_levels</code>	Indicates how to sample new levels for grouping factors specified in <code>re_formula</code> . This argument is only relevant if <code>newdata</code> is provided and <code>allow_new_levels</code> is set to TRUE. If "uncertainty" (default), 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 were observed in the <code>old_data</code> . 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.
<code>parameter</code>	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.
<code>xrange</code>	An integer to set the predictor range (e.g., age) when executing the interpolation via <code>ipts</code> . By default, NULL sets the individual-specific predictor range. Setting <code>xrange = 1</code> applies the same range for individuals within the same higher grouping variable (e.g., study). Setting <code>xrange = 2</code> applies an identical range across the entire sample. Alternatively, a numeric vector (e.g., <code>xrange = c(6, 20)</code>) can be provided to set the range within the specified values.
<code>acg_velocity</code>	A real number specifying the percentage of peak growth velocity to be used as the cessation velocity when estimating the <code>cgv</code> and <code>acgv</code> growth parameters. The <code>acg_velocity</code> should be greater than 0 and less than 1. The default value of <code>acg_velocity = 0.10</code> 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 <code>base::round()</code> 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, <code>numeric_cov_at = list(xx = 2)</code> will set the continuous covariate variable 'xx' to 2. The argument <code>numeric_cov_at</code> 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 <code>ipts</code> 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 <code>aux_variables</code> .
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, <code>levels_id</code> is automatically inferred from the model fit. For models with three or more levels, <code>levels_id</code> 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 <code>levels_id</code> 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, <code>avg_reffects = list(feby = 'study', reby = NULL, over = 'age')</code> .
idata_method	A character string specifying the interpolation method (default NULL). The number of interpolation points is controlled by the <code>ipts</code> argument. Available options: • 'm1': Adapted from the iapvbs package (documented here). This method internally constructs the data frame based on the model configuration. <i>Note:</i> This method may fail if the model includes covariates (especially in <code>univariate_by</code> models). • 'm2': Based on the JMbayes package (documented here). This method uses the exact data frame from the model fit (<code>fit\$data</code>) and is generally more robust. If <code>idata_method = NULL</code>, '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 <code>length.out</code> argument for <code>seq()</code>, controlling the smoothness of distance and velocity curves without altering the predictor range. NULL (the default) Engages automatic behavior based on the <code>dpar</code> argument: it is internally set to 50 for the mean response (<code>dpar = 'mu'</code>), 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 <code>future.apply::future_sapply()</code> .
future_session	A character string or a named list specifying the parallel plan when <code>future = TRUE</code>. • Character string: Defaults to "multisession". Use "multicore" for forking (not supported on Windows). • Named list: Useful for advanced plans like <code>future.mirai::mirai_cluster()</code>. The list must contain: – <code>future_session</code>: The planner function (e.g., <code>mirai_cluster</code>). – Additional named arguments passed to the planner (e.g., <code>daemons</code> for <code>mirai::daemons()</code>). Example: <code>list(future_session = mirai_cluster, daemons = list(...))</code>.
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 <code>future::plan("multicore")</code>. Input options: • List: (e.g., <code>list(1:6, 7:10)</code>). Each element is a sequence of numbers passed to <code>draw_ids</code> to process in a separate chunk. • Numeric vector of length 2: (e.g., <code>c(100, 4)</code>). The first element is the total number of draws, and the second is the number of splits. Indices are generated via <code>parallel::splitIndices()</code>. • Numeric vector of length 1: Specifies the total number of draws. Splits are calculated automatically. • TRUE: Automatically creates splits based on <code>ndraws</code> and <code>cores</code>. If both <code>ndraws</code> and <code>draw_ids</code> are specified, <code>draw_ids</code> takes precedence. This is consistent with post-processing functions in the <code>bsitar</code> and <code>brms</code> packages. • FALSE: Splitting is disabled. Note: On Windows with <code>future::plan("multisession")</code>, background R processes may not free resources automatically. If needed, use <code>installr::kill_all_Rscript_s()</code> to terminate them.
future_method	A character string (default 'future') specifying the parallel computation backend. Options include: • 'future': Uses <code>future.apply::future_lapply()</code> for execution.

	• 'dofuture': Uses <code>foreach::foreach()</code> via the doFuture package.
future_re_expose	A logical value (default NULL) indicating whether to re-expose internal Stan functions when <code>future = TRUE</code>. This is critical when <code>future::plan()</code> 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 <code>method = 'custom'</code>.
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 <code>usecollapse = TRUE</code>. When <code>parallel = TRUE</code>, <code>parallel::makeCluster()</code> 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 <code>parallel = "FORK"</code>, 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 <code>future::availableCores() - 1</code>. • On non-Windows systems, this can be controlled globally via the <code>mc.cores</code> 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 <code>summary = FALSE</code>, and it is only applicable when <code>idata_method = 'm2'</code>. 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 <code>marginaleffects::comparisons()</code> or the <code>marginaleffects::avg_comparisons()</code> function. If FALSE (default), <code>marginaleffects::comparisons()</code> is called, otherwise <code>marginaleffects::avg_comparisons()</code> is used when <code>average = TRUE</code>.
plot	A logical value specifying whether to plot comparisons by calling the <code>marginaleffects::plot_comparisons()</code> function (TRUE) or not (FALSE). If FALSE (default), then <code>marginaleffects::comparisons()</code> or <code>marginaleffects::avg_comparisons()</code> 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 <code>re_formula = NA</code>, and FALSE if <code>re_formula = NULL</code>.
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.

<code>deriv</code>	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 <code>variables</code> argument for more details.
<code>model_deriv</code>	A logical value specifying whether to estimate the velocity curve from the derivative function or by differentiating the distance curve. Set <code>model_deriv = TRUE</code> for functions that require the velocity curve, such as <code>growthparameters()</code> and <code>plot_curves()</code> . Set it to <code>NULL</code> for functions that use the distance curve (i.e., fitted values), such as <code>loo_validation()</code> and <code>plot_ppc()</code> .
<code>method</code>	A character string indicating whether to compute estimates using the 'marginaleffects' package (<code>method = 'pkg'</code>) or custom functions for efficiency and speed (<code>method = 'custom'</code> , default). The <code>method = 'pkg'</code> option is only suitable for simple cases and should be used with caution. <code>method = 'custom'</code> 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 <code>hypothesis</code> argument, and allows users to add or return draws (see <code>pdraws</code> for details). If <code>method = 'pkg'</code> , the <code>by</code> argument must not contain the predictor (e.g., <code>age</code>), and <code>variables</code> must either be <code>NULL</code> (which defaults to <code>list(age = 1e-6)</code>) or a list with factor variables like <code>variables = list(class = 'pairwise')</code> or <code>variables = list(age = 1e-6, class = 'pairwise')</code> . With <code>method = 'custom'</code> , the <code>by</code> argument can include predictors, which will be ignored, and <code>variables</code> should not contain predictors, but can accept factor variables as a vector (e.g., <code>variables = c('class')</code>). Using <code>method = 'custom'</code> is strongly recommended for better performance and flexibility.
<code>marginals</code>	A list, data.frame, or tibble of velocity returned by the marginaleffects functions (default <code>NULL</code>). This is only evaluated when <code>method = 'custom'</code> . The <code>marginals</code> can be the output from marginaleffects functions or posterior draws from <code>marginaleffects::posterior_draws()</code> . The <code>marginals</code> argument is primarily used for internal purposes only.
<code>preparms</code>	A list, data.frame, or tibble of pre computed parameters (default <code>NULL</code>). This is only evaluated when <code>method = 'custom'</code> . The <code>preparms</code> 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 <code>marginaleffects::predictions()</code> function and should not be confused with the transform argument from the deprecated <code>brms::posterior_predict()</code> function. It's important to note that for both <code>marginaleffects::predictions()</code> and <code>marginaleffects::avg_predictions()</code> , 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 <code>avg_*()</code> or with the by argument, and not unit-level estimates. See <code>?weighted.mean</code> • 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 <code>insight::find_weights()</code> and use them when taking weighted averages of estimates. Warning: <code>newdata=datagrid()</code> 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.

<code>constrats_at</code>	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., <code>constrats_at = list(age = 'min')</code>) or as numeric vectors (e.g., <code>constrats_at = list(age = c(6, 7))</code>). If <code>constrats_at = NULL</code> , level-1 predictors (e.g., <code>age</code>) are automatically set to their unique values (i.e., <code>constrats_at = list(age = 'unique')</code>). To turn off this behavior, use <code>constrats_at = FALSE</code> . Note that <code>constrats_at</code> only affects data subsets prepared via <code>marginaleffects::datagrid()</code> or the <code>newdata</code> argument. The argument is evaluated only when <code>method = 'custom'</code> and hypothesis is not NULL.
<code>constrats_subset</code>	A named list (default FALSE) to subset draws for which contrasts should be computed via the hypothesis argument. This is similar to <code>constrats_at</code> , except that <code>constrats_subset</code> filters draws based on the character vector of variable names (e.g., <code>constrats_subset = list(id = c('id1', 'id2'))</code>) rather than numeric values. The argument is evaluated only when <code>method = 'custom'</code> and hypothesis is not NULL.
<code>reformat</code>	A logical (default TRUE) indicating whether to reformat the output returned by <code>marginaleffects</code> as a data frame. Column names are redefined as <code>conf.low</code> to Q2.5 and <code>conf.high</code> to Q97.5 (assuming <code>conf_int = 0.95</code>). Additionally, some columns (<code>term</code> , <code>contrast</code> , etc.) are dropped from the data frame.
<code>estimate_center</code>	A character string (default NULL) specifying how to center estimates: either 'mean' or 'median'. This option sets the global options as follows: <code>options("marginaleffects_posterior_center" = "mean")</code> or <code>options("marginaleffects_posterior_center" = "median")</code> . These global options are restored upon function exit using <code>base::on.exit()</code> .
<code>estimate_interval</code>	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: <code>options("marginaleffects_posterior_interval" = "eti")</code> or <code>options("marginaleffects_posterior_interval" = "hdi")</code> , and is restored on exit using <code>base::on.exit()</code> .
<code>dummy_to_factor</code>	A named list (default NULL) to convert dummy variables into a factor variable. The list must include the following elements: <code>factor.dummy</code>: A character vector of dummy variables to be converted to factors. <code>factor.name</code>: The name for the newly created factor variable (default is 'factor.var' if NULL). <code>factor.level</code>: A vector specifying the factor levels. If NULL, levels are taken from <code>factor.dummy</code>. If <code>factor.level</code> is provided, its length must match <code>factor.dummy</code>.
<code>verbose</code>	A logical argument (default FALSE) to specify whether to print information collected during the setup of the object(s).
<code>expose_function</code>	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 <code>expose_functions</code> argument from the <code>bsitar()</code> call. Manual specification of <code>usesavedfuns</code> 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, <code>clearenvfuns</code> is set based on the value of <code>usesavedfuns</code> : TRUE if <code>usesavedfuns</code> = TRUE, or FALSE if <code>usesavedfuns</code> = FALSE.
funlist	A list (default NULL) specifying function names. This is rarely needed, as required functions are typically retrieved automatically. A use case for <code>funlist</code> is when <code>sigma_formula</code> , <code>sigma_formula_gr</code> , or <code>sigma_formula_gr_str</code> use an external function (e.g., <code>poly(age)</code>). The <code>funlist</code> should include function names defined in the <code>globalenv()</code> . For functions needing both distance and velocity curves (e.g., <code>plot_curves(..., opt = 'dv')</code>), <code>funlist</code> 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 <code>xvar</code> is inferred internally. A use case is when conflicting variables exist (e.g., <code>sigma_formula</code>) and user wants to set a specific variable as 'x'.
idvar	A character string (default NULL) specifying the 'id' variable. Rarely used because <code>idvar</code> 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 <code>xvar</code> is set as <code>difx</code> if specified. The argument <code>difx</code> is evaluated only when <code>dpar</code> = 'sigma', ignored otherwise. Note that argument <code>xvar</code> itself is rarely used because <code>xvar</code> is inferred internally. A use case is when conflicting variables exist (e.g., <code>sigma_formula</code>) 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 <code>c("x", "y", "sigma")</code> . The <code>itransform</code> is primarily used to get the <code>xvar</code> variable at original scale i.e., <code>itransform</code> = 'x'. To turn off all transformations, use <code>itransform</code> = "". When <code>itransform</code> = NULL, the appropriate transformation for <code>xvar</code> is selected automatically. Note that when no match for <code>xvar</code> is found in the data frame, the <code>itransform</code> will be ignored within the calling function, <code>'prepare_transformations()'</code> .
incl_autocor	A flag indicating if correlation structures originally specified via <code>autocor</code> 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 specifying how to handle the format and structure of the user-provided newdata before use. The available options are:

- NULL (default): Checks the data format and structure of newdata, and applies `prepare_data2` and `prepare_transformations` as needed.
- TRUE: Returns newdata as-is, without any format checking or transformation.

- 0: Same as TRUE — returns newdata without checking the data format or applying prepare_data2 and prepare_transformations.
- 1: Checks the data format and, if needed, prepares it using prepare_data2, but does not apply prepare_transformations.
- 2: Applies prepare_transformations only, assuming the data format is already correct.

It is strongly recommended to leave both newdata = NULL and newdata_fixed = NULL (the defaults). In this case, the data used during model fitting is automatically retrieved, checked for the required format, and transformed as needed via prepare_data2 and prepare_transformations. The flags newdata_fixed = 0, 1, and 2 are intended primarily for internal use.

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

<code>model</code>	An object of class <code>bgmfit</code> .
<code>newdata</code>	An optional data frame for estimation. If <code>NULL</code> (default), <code>newdata</code> is retrieved from the model.
<code>optimize_df</code>	A list of integers specifying the degrees of freedom (df) values to be optimized. If <code>NULL</code> (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 <code>optimize_df = list(4, 5)</code> . For univariate_by and multivariate models, <code>optimize_df</code> can be a single integer (e.g., <code>optimize_df = 4</code>), a list (e.g., <code>optimize_df = list(4, 5)</code>), 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 <code>optimize_df = list(list(4, 5), list(5, 6))</code> .
<code>optimize_x</code>	A list specifying the transformations for the predictor variable (i.e., <code>x</code>). The available options are <code>NULL</code> , <code>'log'</code> , <code>'sqrt'</code> , 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., <code>optimize_x = list(NULL, 'log', 'sqrt')</code> . Similar to <code>optimize_df</code> , the user can specify different <code>optimize_x</code> values for <code>univariate_by</code> and <code>multivariate</code> submodels. Additionally, it is possible to pass any primitive function instead of fixed functions like <code>log</code> and <code>sqrt</code> . This greatly enhances the flexibility of model optimization by allowing the search for a wide range of <code>x</code> transformations, such as <code>optimize_x = list(function(x) log(x + 3/4))</code> .
<code>optimize_y</code>	A list specifying the transformations of the response variable (i.e., <code>y</code>). The approach and available options for <code>optimize_y</code> are the same as described above for <code>optimize_x</code> .
<code>transform_prior_class</code>	A character vector (default <code>NULL</code>) specifying the parameter class for which transformations of user-specified priors should be performed. These options are <code>'beta'</code>, <code>'sd'</code>, <code>'rsd'</code>, <code>'sigma'</code>, and <code>'dpar'</code>, and they can be specified as follows: <pre>transform_prior_class = c('beta', 'sd', 'rsd', 'sigma', 'dpar')</pre> Note that transformations can only be applied to location-scale based priors such as <code>normal()</code>. Currently, transformation are supported only for the <code>'log'</code> transformed <code>'x'</code> variable. The <code>'log'</code> transformation of prior is performed as follows: <pre>log_location = log(location / sqrt(scale^2 / location^2 + 1)), log_scale = sqrt(log(scale^2 / location^2 + 1)),</pre> where <code>location</code> and <code>scale</code> are the original parameters supplied by the user, and <code>log_location</code> and <code>log_scale</code> are the equivalent parameters on the log scale. Note that <code>transform_prior_class</code> is used on an experimental basis, and therefore the results may not be as intended. We recommend explicitly setting the desired prior for the <code>y</code> scale.
<code>transform_beta_coef</code>	A character vector (default <code>NULL</code>) specifying the regression coefficients for which transformations are applied. The coefficients that can be transformed are <code>'a'</code> , <code>'b'</code> , <code>'c'</code> , <code>'d'</code> , and <code>'s'</code> . The default is <code>transform_beta_coef = c('b', 'c',</code>

'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 character vector specifying the fit criteria to be added. See [add_model_criterion\(\)](#) for details.

`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 <code>base::round()</code> function.
cores	The number of cores to use in parallel processing (default 1). The argument <code>cores</code> is passed to <code>[brms::add_criterion()]</code> .
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 <code>bsitar()</code> model fit configuration. Note: In the <code>optimize_model()</code> 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 <code>expose_functions</code> argument from the <code>bsitar()</code> call. Manual specification of <code>usesavedfuns</code> 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, <code>clearenvfuns</code> is set based on the value of <code>usesavedfuns</code> : TRUE if <code>usesavedfuns</code> = TRUE, or FALSE if <code>usesavedfuns</code> = FALSE.
envir	The environment used for function evaluation. The default is NULL, which sets the environment to <code>parent.frame()</code> . Since most post-processing functions rely on brms , it is recommended to set <code>envir = globalenv()</code> or <code>envir = .GlobalEnv</code> , especially for derivatives like velocity curves.
...	Other arguments passed to <code>update_model</code> .

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_caterpillar.bgmfit

Caterpillar plot for bsitar random effects

Description

Create a caterpillar plot of group-level effects from a `bsitar` model using posterior summaries returned by `brms::ranef()`. When multiple plots are requested, plots are arranged and combined into a single display with **patchwork**.

Usage

```
## S3 method for class 'bgmfit'
plot_caterpillar(
  model = NULL,
  group = NULL,
  groupby = NULL,
  effect = NULL,
  transform_parameter = NULL,
  transform_fun = NULL,
  pdata = NULL,
  probs = c(0.025, 0.975),
  conf = NULL,
  robust = FALSE,
  sort = TRUE,
  resp = NULL,
  dpar = NULL,
  levels_id = NULL,
  idvar = NULL,
  layout = NULL,
  linecolor = NULL,
  linecolor1 = NULL,
  linecolor2 = NULL,
  label.x = NULL,
  label.y = NULL,
  ticks.x = NULL,
  ticks.y = NULL,
  label.title = NULL,
  label.subtitle = NULL,
  legendpos = "bottom",
  linewidth.main = NULL,
  linetype.groupby = NA,
  color.groupby = NULL,
  fill.groupby = NULL,
  point_size = NULL,
  point_color = NULL,
  interval_color = NULL,
  combine = TRUE,
  ncol = NULL,
  wrap_plots_ncol = NULL,
  wrap_plots_nrow = NULL,
  facet_ncol = NULL,
  facet_nrow = NULL,
  scales = "fixed",
  space = "fixed",
  patch_plot.margin = ggplot2::margin(10, 0, 10, 0),
  each_object = FALSE,
  refline = NULL,
  dodge = 0.4,
```

```

    envir = NULL,
    verbose = FALSE,
    ...
)

plot_caterpillar(model, ...)

```

Arguments

model	A <code>bsitar</code> object.
group	A character string specifying the grouping factor to plot. This is typically the individual identifier, such as <code>id</code> . If <code>group = NULL</code> (the default), it is determined automatically from the model object. In most cases, <code>idvar</code> and <code>levels_id</code> are used to define the grouping structure, so <code>group</code> will rarely need to be specified manually.
groupby	A character string specifying the factor within which <code>group</code> is nested. This corresponds to the <code>by</code> argument in <code>gr(..., by = xxx)</code> , where <code>xxx</code> is the grouping variable. If <code>groupby = NULL</code> (the default), it is determined automatically from the model object.
effect	A character string specifying the random-effect coefficient to plot. If <code>NULL</code> , the first available effect is used.
transform_parameter	Optional character vector of parameter names to transform. Each supplied value must match an available random-effect parameter.
transform_fun	Optional function or list of functions used to transform the selected parameters. Accepted forms are: • A single function, applied to all targeted combinations. • A named list of functions with one function per <code>transform_parameter</code>. • A named list of functions whose names match the internal transformation keys used by the method. Each function must accept a numeric vector and return a numeric vector of the same length.
pdata	An optional parameter data frame in wide format where parameter name and key variable such as <code>Estimate</code> are named as <code>effect.Estimate</code> . Currently ignored. Included here for future expansion.
probs	Numeric vector of length 2 giving the lower and upper credible interval probabilities passed to <code>brms::ranef()</code> . Default is <code>c(0.025, 0.975)</code> .
conf	A numeric value (default 0.95) specifying the confidence interval (CI) level for the bands. See <code>growthparameters()</code> for more details.
robust	A logical value to specify the summary options. If <code>FALSE</code> (default), the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If <code>TRUE</code> , the median and median absolute deviation (MAD) are applied instead. Ignored if <code>summary</code> is <code>FALSE</code> .
sort	Logical; if <code>TRUE</code> , levels are ordered by the posterior mean estimate. If <code>FALSE</code> , the original order is retained.

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.
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.
idvar	A character string (default NULL) specifying the 'id' variable. Rarely used because idvar is inferred internally.
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.
ticks.x	Optional logical value. If FALSE, x-axis tick labels are suppressed. Default is NULL, which is treated internally as FALSE.
ticks.y	Optional logical value. If FALSE, y-axis tick labels are suppressed. Default is NULL, which is treated internally as FALSE.
label.title	Optional character string used as the plot title. Default is 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, 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.

linewidth.main	A numeric value to specify the line width for distance and velocity curves. The default NULL sets the width to 0.35.
linetype.groupby	A character string specifying the line type of distance and velocity curves for covariates or individual-specific curves. If NA, then default color is 'black' for all lines.
color.groupby	A character string specifying the line color of distance and velocity curves for covariates or individual-specific curves. If NA, then default color is 'black' for all lines. If NULL (default), then line color is set based on the covariate extracted from the model_info. Note that for individual-specific curves for covariate adjusted models, user can set it as color.groupby = interaction(covariate, id)
fill.groupby	A character string specifying the fill color of distance and velocity curves for covariates or individual-specific curves. If NA, then default color is 'black' for all lines. If NULL (default), then fill color is set same as the color.groupby.
point_size	Size used for the posterior mean points. When NULL (default), the size is set internally to 1.5.
point_color	Color used for the posterior mean points. When NULL (default), the color is set internally to "#1f4e79" if groupby is NULL; otherwise, distinct colors are used for each level of the groupby variable.
interval_color	Color used for the posterior interval lines. When NULL (default), the color is set internally to "#1f4e79" if groupby is NULL; otherwise, distinct colors are used for each level of the groupby variable.
combine	Logical; if TRUE and more than one plot is generated, plots are combined with patchwork::wrap_plots() . If FALSE, the function returns the individual plot objects without combining them.
ncol	Optional integer (default NULL) specifying the number of columns for patchwork::wrap_plots() . If NULL, then number of columns is set as 2.
wrap_plots_ncol	Optional integer set the number of columns for the patchwork::wrap_plots() . Default is NULL, which allows internally setting the wrap_plots_ncol based on the number of plots.
wrap_plots_nrow	Optional integer set the number of rows for the patchwork::wrap_plots() . Default is NULL, which allows internally setting the wrap_plots_nrow based on the number of plots.
facet_ncol	Optional integer set the number of columns for the ggplot2::facet_wrap() . Default is NULL, which allows internally setting the facet_ncol based on the number of plots.
facet_nrow	Optional integer set the number of rows for the ggplot2::facet_wrap() . Default is NULL, which allows internally setting the facet_ncol based on the number of plots.
scales	Should scales be fixed ("fixed", the default), free ("free"), or free in one dimension ("free_x", "free_y")?

space	If "fixed" (default), all panels have the same size and the number of rows and columns in the layout can be arbitrary. If "free_x", panels have widths proportional to the length of the x-scale, but the layout is constrained to one row. If "free_y", panels have heights proportional to the length of the y-scale, but the layout is constrained to one column.
patch_plot.margin	Optional margin specification passed to patchwork .
each_object	Optional logical (default FALSE) indicating whether to return each plot object (combined and individual plots) or only the combined when multiple plots are requested.
refline	Numeric value for the vertical reference line. The default is NULL, which is internally treated as 0. Note that the same transformation is applied to refline as specified by transform_parameter. For example, if the parameter 'c' is transformed using 'exp', then refline = 0 is transformed to refline = exp(0), that is, refline = 1.
dodge	Width used to control the horizontal thickness of the interval bars.
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.
verbose	A logical argument (default FALSE) to specify whether to print information collected during the setup of the object(s).
...	Additional arguments passed to the brms::fitted.brmsfit() and brms::predict() functions.

Details

The function extracts group-level summaries using `brms::ranef()` with `summary = TRUE`, then plots posterior means and credible intervals for the selected grouping factor and effect. If transformations are requested, they are applied before summarizing the draws so that the plotted summaries are on the transformed scale.

Value

A ggplot object showing a caterpillar plot of the selected random effect.

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 '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

# Individual-specific size (parameter a)
plot_caterpillar(model, group = "id", effect = "a_Intercept")

# Individual-specific timing (parameter b)
plot_caterpillar(model, group = "id", effect = "b_Intercept")

# Individual-specific intensity (parameter c)
plot_caterpillar(model, group = "id", effect = "c_Intercept")

```

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 <code>bgmfit</code> .
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 <code>NULL</code> (the default), plots are generated for all main effects and two-way interactions estimated in the model. When specifying effects manually, <i>all</i> two-way interactions (including grouping variables) may be plotted even if not originally modeled.
conditions	An optional <code>data.frame</code> containing variable values to condition on. Each effect defined in <code>effects</code> will be plotted separately for each row of <code>conditions</code> . Val-

	ues in the <code>cond__</code> column will be used as titles of the subplots. If <code>cond__</code> is not given, the row names will be used for this purpose instead. It is recommended to only define a few rows in order to keep the plots clear. See make_conditions for an easy way to define conditions. If <code>NULL</code> (the default), numeric variables will be 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.
<code>int_conditions</code>	An optional named list whose elements are vectors of values of the variables specified in <code>effects</code> . At these values, predictions are evaluated. The names of <code>int_conditions</code> have to match the variable names exactly. Additionally, the elements of the vectors may be named themselves, in which case their names appear as labels for the conditions in the plots. Instead of vectors, functions returning vectors may be passed and are applied on the original values of the corresponding variable. If <code>NULL</code> (the default), predictions are evaluated at the <i>mean</i> and at <i>mean + / - sd</i> for numeric predictors and at all categories for factor-like predictors.
<code>re_formula</code>	A formula containing group-level effects to be considered in the conditional predictions. If <code>NULL</code> , include all group-level effects; if <code>NA</code> (default), include no group-level effects.
<code>spaghetti</code>	Logical. Indicates if predictions should be visualized via spaghetti plots. Only applied for numeric predictors. If <code>TRUE</code> , it is recommended to set argument <code>ndraws</code> to a relatively small value (e.g., 100) in order to reduce computation time.
<code>surface</code>	Logical. Indicates if interactions or two-dimensional smooths should be visualized as a surface. Defaults to <code>FALSE</code> . The surface type can be controlled via argument <code>stype</code> of the related plotting method.
<code>categorical</code>	Logical. Indicates if effects of categorical or ordinal models should be shown in terms of probabilities of response categories. Defaults to <code>FALSE</code> .
<code>ordinal</code>	(Deprecated) Please use argument <code>categorical</code> . Logical. Indicates if effects in ordinal models should be visualized as a raster with the response categories on the y-axis. Defaults to <code>FALSE</code> .
<code>method</code>	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.
<code>allow_new_levels</code>	A flag indicating if new levels of group-level effects are allowed (defaults to <code>FALSE</code>). Only relevant if <code>newdata</code> is provided.
<code>estimation_method</code>	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.
<code>transform</code>	A function or a character string naming a function to be applied on the predicted responses before summary statistics are computed. Only allowed if <code>method = "posterior_predict"</code> .

<code>transform_draws</code>	A function applied to individual draws from the posterior distribution before computing summaries (default NULL). The argument <code>transform_draws</code> is derived from the <code>marginaleffects::predictions()</code> function and should not be confused with the <code>transform</code> argument from the deprecated <code>brms::posterior_predict()</code> function. It's important to note that for both <code>marginaleffects::predictions()</code> and <code>marginaleffects::avg_predictions()</code> , the <code>transform_draws</code> argument takes precedence over the <code>transform</code> argument. Note that when <code>transform_draws = NULL</code> , an attempt is made to automatically set <code>transform_draws = 'exp'</code> for <code>dpar = 'sigma'</code> . User can set <code>transform_draws = FALSE</code> to turn off this automatic assignment of 'exp' to the <code>transform_draws</code> . It is also important to set <code>transform_draws = FALSE</code> when computing the first derivative (velocity) for <code>dpar = 'sigma'</code> .
<code>resolution</code>	Number of support points used to generate the plots. Higher resolution leads to smoother plots. Defaults to 100. If <code>surface</code> is TRUE, this implies 10000 support points for interaction terms, so it might be necessary to reduce resolution when only few RAM is available.
<code>select_points</code>	Positive number. Only relevant if <code>points</code> or <code>rug</code> 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 <code>select_points</code> from the values in conditions are excluded. By default, all points are used.
<code>too_far</code>	Positive number. For surface plots only: Grid points that are too far away from the actual data points can be excluded from the plot. <code>too_far</code> determines what is too far. The grid is scaled into the unit square and then grid points more than <code>too_far</code> from the predictor variables are excluded. By default, all grid points are used. Ignored for non-surface plots.
<code>probs</code>	(Deprecated) The quantiles to be used in the computation of uncertainty intervals. Please use argument <code>prob</code> instead.
<code>robust</code>	If TRUE (the default) the median is used as the measure of central tendency. If FALSE the mean is used instead.
<code>newdata</code>	An optional data frame for estimation. If NULL (default), <code>newdata</code> is retrieved from the model.
<code>ndraws</code>	A positive integer indicating the number of posterior draws to use in estimation. If NULL (default), all draws are used.
<code>dpar</code>	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
<code>draw_ids</code>	An integer specifying the specific posterior draw(s) to use in estimation (default NULL).
<code>levels_id</code>	An optional argument to specify the <code>ids</code> 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, <code>levels_id</code> is automatically inferred from the model fit. For models with three or more levels, <code>levels_id</code> is inferred from the model fit under the assumption that hierarchy is specified from the lowest to the uppermost level, i.e., <code>id</code> followed by <code>study</code> , where <code>id</code> is nested within <code>study</code> . However, it is not guaranteed that <code>levels_id</code> 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. <i>Note:</i> 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, 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: • <code>factor.dummy</code>: A character vector of dummy variables to be converted to factors. • <code>factor.name</code>: The name for the newly created factor variable (default is 'factor.var' if NULL). • <code>factor.level</code>: A vector specifying the factor levels. If NULL, levels are taken from <code>factor.dummy</code>. If <code>factor.level</code> is provided, its length must match <code>factor.dummy</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 <code>fit.criteria</code> or <code>bayes.R2</code> during model optimization. • FALSE (default): Stan functions are not exposed. • NULL: The setting is inherited from the original <code>bsitar()</code> model fit configuration. Note: In the <code>optimize_model()</code> 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 <code>expose_functions</code> argument from the <code>bsitar()</code> call. Manual specification of <code>usesavedfuns</code> 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, <code>clearenvfuns</code> is set based on the value of <code>usesavedfuns</code> : TRUE if <code>usesavedfuns</code> = TRUE, or FALSE if <code>usesavedfuns</code> = 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 off 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 specifying how to handle the format and structure of the user-provided newdata before use. The available options are: • NULL (default): Checks the data format and structure of newdata, and applies prepare_data2 and prepare_transformations as needed. • TRUE: Returns newdata as-is, without any format checking or transformation. • 0: Same as TRUE — returns newdata without checking the data format or applying prepare_data2 and prepare_transformations. • 1: Checks the data format and, if needed, prepares it using prepare_data2, but does not apply prepare_transformations. • 2: Applies prepare_transformations only, assuming the data format is already correct. It is strongly recommended to leave both newdata = NULL and newdata_fixed = NULL (the defaults). In this case, the data used during model fitting is automatically retrieved, checked for the required format, and transformed as needed via prepare_data2 and prepare_transformations. The flags newdata_fixed = 0, 1, and 2 are intended primarily for internal use.
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	<i>Plot growth curves for the Bayesian SITAR model</i>
--------------------	--

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,
  cov = NULL,
  grid_call = NULL,
  grid_by = NULL,
  grid_add = NULL,
  levels_id = NULL,
  avg_reffects = NULL,
  ipts = NULL,
  model_deriv = NULL,
  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,
transform_xaxis = list(x_min = NULL, x_max = NULL, n = 5, by = NULL, xaxis_fun = NULL,
  spacing = "data", accuracy = 0.1, expand_mult = 0, use_coord_limits = TRUE,
  coord_pad_mult = 0.05, coord_pad_add = 0),
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 = "bottom",
linetype.apv = NULL,
linewidth.main = NULL,
linewidth.apv = NULL,
linetype.groupby = NA,
color.groupby = NULL,
fill.groupby = NULL,
band.alpha = NULL,
band.legends = FALSE,
show_age_takeoff = TRUE,
show_age_peak = TRUE,
show_age_cessation = TRUE,
show_vel_takeoff = FALSE,
show_vel_peak = FALSE,
show_vel_cessation = FALSE,
each_object = FALSE,
ncol = NULL,
print = TRUE,
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 <code>bgmfit</code> .
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)
apv	A logical value (default FALSE) indicating whether to calculate and plot the age at peak velocity (APGV) when <code>opt</code> 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 <code>univariate_by</code> and <code>multivariate</code> models. See bsitar() for details on <code>univariate_by</code> and <code>multivariate</code> 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.
cov	An optional character string to specify the covariate passed to the plot argument (default NULL). Rarely used.
grid_call	An optional logical (default NULL) to indicate whether to call marginaleffects::datagrid() when dpar = 'sigma'. Rarely used. The two indications are when sigma_model = 'ls' or opt = 'v'.
grid_by	An optional character string to specify the by argument passed to the marginaleffects::datagrid() (default NULL). Rarely used. Ignored when grid_call = FALSE. If grid_call = TRUE and grid_by = NULL, then internally set as grid_by = cov.
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 <code>seq()</code>, 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: <code>fitted_draws()</code>, <code>predict_draws()</code>, <code>growthparameters()</code>, <code>plot_curves()</code>, <code>get_predictions()</code>, <code>get_comparisons()</code>, and <code>get_growthparameters()</code>.
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 <code>growthparameters()</code> and <code>plot_curves()</code> . Set it to NULL for functions that use the distance curve (i.e., fitted values), such as <code>loo_validation()</code> and <code>plot_ppc()</code> .
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 <code>acgv_velocity</code> argument (see below). The <code>acgv_velocity</code> 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 <code>acgv</code> . 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 <code>fitted_draws()</code> , while the 'predict' method calls <code>predict_draws()</code> . See <code>brms::fitted.brmsfit()</code> and <code>brms::predict.brmsfit()</code> for details.
allow_new_levels	A flag indicating if new levels of group-level effects are allowed (defaults to FALSE). Only relevant if <code>newdata</code> is provided.
sample_new_levels	Indicates how to sample new levels for grouping factors specified in <code>re_formula</code> . This argument is only relevant if <code>newdata</code> is provided and <code>allow_new_levels</code> is set to 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 were observed in the <code>old_data</code> . 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 <code>autocor</code> 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 <code>summary</code> is FALSE.
transform_draws	A function applied to individual draws from the posterior distribution before computing summaries (default NULL). The argument <code>transform_draws</code> is derived from the <code>marginaleffects::predictions()</code> 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'`.

transform_xaxis

A named `list()` of arguments for `build_scale_x()`. Any unspecified arguments are automatically filled with the default values from `build_scale_x()`. Common elements include:

- `x_min`: lower x-axis limit. If `NULL` (default), `x_min` is inferred internally from the `xvar`
- `x_max`: upper x-axis limit. If `NULL` (default), `x_max` is inferred internally from the `xvar`
- `n`: number of breaks.
- `by`: optional step size for breaks.
- `xaxis_fun`: transformation used for break placement and inverse labeling, for example "identity", "log", "log10", "sqrt", or "square", or a function
- `spacing`: either "data" or "trans".
- `accuracy`: label accuracy.
- `expand_mult`: scale expansion multiplier.
- `use_coord_limits`: logical; if `TRUE`, zoom with `ggplot2::coord_cartesian()`.
- `coord_pad_mult`: multiplicative padding added to both x-axis ends.
- `coord_pad_add`: additive padding added to both x-axis ends.

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 <code>future::availableCores() - 1</code>. • On non-Windows systems, this can be controlled globally via the <code>mc.cores</code> 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 <code>opt</code> includes combinations like 'dv', 'Dv', 'dV', or 'DV'. The alternative layout option 'facet' uses <code>facet_wrap</code> from ggplot2 to map and draw plots when <code>opt</code> 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 <code>opt = 'dv'</code> , the distance line is controlled by <code>linecolor1</code> . The default NULL sets <code>linecolor1</code> to 'orange2'.
linecolor2	The color of the second line when the layout is 'single'. For example, in <code>opt = 'dv'</code> , the velocity line is controlled by <code>linecolor2</code> . The default NULL sets <code>linecolor2</code> 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 <code>layout = 'facet'</code> , 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, 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 of distance and velocity curves for covariates or individual-specific curves. If NA, then default color is 'black' for all lines.

color.groupby	A character string specifying the line color of distance and velocity curves for covariates or individual-specific curves. If NA, then default color is 'black' for all lines. If NULL (default), then line color is set based on the covariate extracted from the model_info. Note that for individual-specific curves for covariate adjusted models, user can set it as color.groupby = interaction(covariate, id)
fill.groupby	A character string specifying the fill color of distance and velocity curves for covariates or individual-specific curves. If NA, then default color is 'black' for all lines. If NULL (default), then fill color is set same as the color.groupby.
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.
band.legends	A logical value (default FALSE) to indicate whether to show legends for CI bands. It sets ggplot2::guides(fill = 'none')
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.
each_object	Optional logical (default FALSE) indicating whether to return each plot object (combined and individual plots) or only the combined when multiple plots are requested.
ncol	Optional integer (default NULL) specifying the number of columns for patchwork::wrap_plots(). If NULL, then number of columns is set as 2.
print	A logical value (default TRUE) to indicate whether to print the plot object before returning it.
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 <code>ipts</code> argument. Available options: 'm1': Adapted from the iapvbs package (documented here). This method internally constructs the data frame based on the model configuration. <i>Note:</i> This method may fail if the model includes covariates (especially in <code>univariate_by</code> models). 'm2': Based on the JMbayes package (documented here). This method uses the exact data frame from the model fit (<code>fit\$data</code>) and is generally more robust. If <code>idata_method = NULL</code>, '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 <code>parms_eval</code>. The default method is <code>getPeak</code>, which uses the <code>sitar::getPeak()</code> function from the <code>sitar</code> package. Alternatively, <code>findpeaks</code> uses the <code>findpeaks</code> function from the <code>pracma</code> 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 <code>fullframe</code> object in which <code>newdata</code> is bound to the summary estimates. Note that <code>fullframe</code> cannot be used with <code>summary = FALSE</code>, and it is only applicable when <code>idata_method = 'm2'</code>. A typical use case is when fitting a <code>univariate_by</code> 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: <code>factor.dummy</code>: A character vector of dummy variables to be converted to factors. <code>factor.name</code>: The name for the newly created factor variable (default is 'factor.var' if NULL). <code>factor.level</code>: A vector specifying the factor levels. If NULL, levels are taken from <code>factor.dummy</code>. If <code>factor.level</code> is provided, its length must match <code>factor.dummy</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 <code>fit.criteria</code> or <code>bayes_R2</code> during model optimization. - FALSE (default): Stan functions are not exposed. - NULL: The setting is inherited from the original <code>bsitar()</code> model fit configuration. Note: In the <code>optimize_model()</code> 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 <code>expose_functions</code> argument from the <code>bsitar()</code> call. Manual specification of <code>usesavedfuns</code> 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, <code>clearenvfuns</code> is set based on the value of <code>usesavedfuns</code> : TRUE if <code>usesavedfuns</code> = TRUE, or FALSE if <code>usesavedfuns</code> = FALSE.
funlist	A list (default NULL) specifying function names. This is rarely needed, as required functions are typically retrieved automatically. A use case for <code>funlist</code> is when <code>sigma_formula</code> , <code>sigma_formula_gr</code> , or <code>sigma_formula_gr_str</code> use an external function (e.g., <code>poly(age)</code>). The <code>funlist</code> should include function names defined in the <code>globalenv()</code> . For functions needing both distance and velocity curves (e.g., <code>plot_curves(..., opt = 'dv')</code>), <code>funlist</code> 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 <code>xvar</code> is inferred internally. A use case is when conflicting variables exist (e.g., <code>sigma_formula</code>) 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 <code>xvar</code> is set as <code>difx</code> if specified. The argument <code>difx</code> is evaluated only when <code>dpar</code> = 'sigma', ignored otherwise. Note that argument <code>xvar</code> itself is rarely used because <code>xvar</code> is inferred internally. A use case is when conflicting variables exist (e.g., <code>sigma_formula</code>) and user wants to set a specific variable as 'x'.
idvar	A character string (default NULL) specifying the 'id' variable. Rarely used because <code>idvar</code> is inferred internally.
itransform	A character string (default NULL) indicating the variables names that are reverse transformed. Options are <code>c("x", "y", "sigma")</code> . The <code>itransform</code> is primarily used to get the <code>xvar</code> variable at original scale i.e., <code>itransform</code> = 'x'. To turn off all transformations, use <code>itransform</code> = "". When <code>itransform</code> = NULL, the appropriate transformation for <code>xvar</code> is selected automatically. Note that when no match for <code>xvar</code> is found in the data frame, the <code>itransform</code> will be ignored within the calling function, <code>'prepare_transformations()'</code> .
newdata_fixed	An indicator specifying how to handle the format and structure of the user-provided <code>newdata</code> before use. The available options are: • NULL (default): Checks the data format and structure of <code>newdata</code>, and applies <code>prepare_data2</code> and <code>prepare_transformations</code> as needed. • TRUE: Returns <code>newdata</code> as-is, without any format checking or transformation. • 0: Same as TRUE — returns <code>newdata</code> without checking the data format or applying <code>prepare_data2</code> and <code>prepare_transformations</code>. • 1: Checks the data format and, if needed, prepares it using <code>prepare_data2</code>, but does not apply <code>prepare_transformations</code>. • 2: Applies <code>prepare_transformations</code> only, assuming the data format is already correct.

	It is strongly recommended to leave both <code>newdata = NULL</code> and <code>newdata_fixed = NULL</code> (the defaults). In this case, the data used during model fitting is automatically retrieved, checked for the required format, and transformed as needed via <code>prepare_data2</code> and <code>prepare_transformations</code> . The flags <code>newdata_fixed = 0</code> , <code>1</code> , and <code>2</code> are intended primarily for internal use.
<code>envir</code>	The environment used for function evaluation. The default is <code>NULL</code> , which sets the environment to <code>parent.frame()</code> . Since most post-processing functions rely on brms , it is recommended to set <code>envir = globalenv()</code> or <code>envir = .GlobalEnv</code> , especially for derivatives like velocity curves.
<code>...</code>	Additional arguments passed to the <code>brms::fitted.brmsfit()</code> and <code>brms::predict()</code> functions.

Details

The `plot_curves()` function is a generic tool for visualizing up to six types of growth curves:

- Population-average distance curve (`opt = "d"`)
- Population-average velocity curve (`opt = "v"`)
- Individual-specific distance curves (`opt = "D"`)
- Individual-specific velocity curves (`opt = "V"`)
- Unadjusted individual growth curves (i.e., observed growth curves; `opt = "u"`)
- Adjusted individual growth curves, adjusted for the model-estimated random effects (`opt = "a"`)

Users can request any combination of these plots by specifying the desired curve types in the `opt` argument. For example, to request all six plots, set `opt = "dvDVua"`. The plots are generated and assembled in the same order as the characters appear in `opt`.

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 make sense to interpolate 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_diagnostics.bgmfit

Diagnostic plots for Bayesian SITAR models

Description

This function generate various model diagnostic plots including residual-based diagnostics, MCMC convergence diagnostics, and posterior predictive checks. When multiple plots are requested, patchable plots are combined into a single display using **patchwork**.

Usage

```
## S3 method for class 'bgmfit'
plot_diagnostics(
  model,
  newdata = NULL,
  plots = c("rvf", "rvp", "qq", "qqn", "qqp", "pairs", "acf", "acfp", "acfr", "trace",
    "dens_overlay", "rhat", "rhat_hist", "neff", "ppc_overlay", "ppc_hist",
    "ppc_scatter", "ppc_stat"),
  set_draws = "epred",
  resp = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  seed = 123,
  re_formula = NULL,
  dpar = NULL,
  variable = NULL,
  combine = TRUE,
  ncol = NULL,
  point_alpha = 0.15,
  bins = 30,
  regex = FALSE,
  smooth_se = FALSE,
  smooth_method = "lm",
  ppc_stat = "mean",
  rank_overlay = FALSE,
  add_plot_df = FALSE,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  category = ".category",
  wrap_title = NULL,
  title_size = 12,
  each_object = FALSE,
  qq_plot_args = list(draw_ids = NULL, draw_ids_select = 1, summary = "mean", qq_type =
```

```

    "qq", seed = 123),
  patch_plot.margin = ggplot2::margin(10, 0, 10, 0),
  funlist = NULL,
  future = FALSE,
  future_session = "multisession",
  future_splits = TRUE,
  future_method = "future",
  future_re_expose = NULL,
  transform = NULL,
  transform_draws = NULL,
  itransform = NULL,
  model_deriv = NULL,
  dummy_to_factor = NULL,
  xvar = NULL,
  difx = NULL,
  idvar = NULL,
  levels_id = NULL,
  conf_level = 0.95,
  deriv = 0,
  plot = FALSE,
  method = "pkg",
  ipts = FALSE,
  idata_method = NULL,
  newdata_fixed = NULL,
  verbose = FALSE,
  envir = NULL,
  ...
)

plot_diagnostics(model, ...)

```

Arguments

<code>model</code>	An object of class <code>bgmfit</code>
<code>newdata</code>	Optional <code>data.frame</code> used to compute residual-based diagnostics. If <code>NULL</code> , the function attempts to use the data stored in the fitted model object. Supplying <code>newdata</code> explicitly is safer and is recommended for reproducibility.
<code>plots</code>	Character vector specifying which diagnostics to produce. Supported values are: "rvf" residual vs fitted plot showing residuals plotted against fitted values derived from posterior draws. "rvp" residual vs predictor (such as age) plot showing residuals plotted against predictor values derived from posterior draws. "qqn" Normal Q-Q plot based on summarized posterior residuals. "qqp" Pearson Q-Q plot based on standardized posterior residuals. "qq" Same as "qqp". "pairs" Pair plots for MCMC draws via <code>bayesplot::mcmc_pairs()</code>.

	"acfp" Autocorrelation plots of parameters for MCMC draws via <code>brms::mcmc_plot()</code> with type = "acf". "acfr" Autocorrelation plots of residuals via custom <code>acf_residuals_custom</code> function. "acf" Same as "acfr". "trace" Trace plots for selected parameters via <code>brms::mcmc_plot()</code> with type = "trace". "dens_overlay" Density overlays for selected parameters via <code>brms::mcmc_plot()</code> with type = "dens_overlay". "rhat" Potential scale reduction diagnostics via <code>brms::mcmc_plot()</code> with type = "rhat". "rhat_hist" Potential scale reduction diagnostics via <code>brms::mcmc_plot()</code> with type = "rhat_hist". "neff" Effective sample size diagnostics via <code>brms::mcmc_plot()</code> with type = "neff". "ppc_overlay" Posterior predictive density overlay via <code>brms::pp_check()</code> with type = "dens_overlay". "ppc_hist" Posterior predictive histogram via <code>brms::pp_check()</code> with type = "hist". "ppc_scatter" Posterior predictive observed-versus-replicated average scatter plot via <code>brms::pp_check()</code> with type = "scatter_avg". "ppc_stat" Posterior predictive check for a summary statistic via <code>brms::pp_check()</code> with type = "stat".
set_draws	Character scalar indicating which fitted-value draw type to use for the residual-versus-fitted plot. Must be one of: "epred" Expected posterior predictive values from <code>tidybayes::add_epred_draws()</code>. This is usually the preferred option when available. "linpred" Linear predictor values from <code>tidybayes::add_linpred_draws()</code>. "prediction" Posterior predictive draws from <code>tidybayes::add_predicted_draws()</code>. The tidybayes documentation recommends using expectation-scale predictions from <code>tidybayes::add_epred_draws()</code> when available, because these correspond to the expectation of the posterior predictive distribution more directly than the linear predictor in many models.
resp	Optional name of the response variable to plot. This is primarily useful for multivariate models, where a specific response must often be selected.
ndraws	Optional integer specifying the number of posterior draws to use in residual-based diagnostics and posterior predictive checks.
draw_ids	An integer specifying the specific posterior draw(s) to use in estimation (default NULL).
seed	Optional random seed used when subsampling posterior draws.
re_formula	Optional formula passed to prediction methods to control whether group-level effects are included.
dpar	Optional distributional parameter name passed to the underlying draw function when relevant.

variable	Optional character vector of parameter names for MCMC diagnostic plots such as trace, autocorrelation, and density overlays.
combine	Logical; if TRUE and more than one plot is generated, compatible plots are combined with <code>patchwork::wrap_plots()</code> . If FALSE, the function returns the individual plot objects without combining them.
ncol	Optional integer giving the number of columns to use when combining multiple plots with patchwork . If NULL, a simple default is chosen automatically.
point_alpha	Numeric transparency level used for points in the residual-versus-fitted plot.
bins	Numeric value passed to <code>'stat_bin()'</code> for <code>'ppc_hist'</code> and <code>'ppc_stat'</code> plots.
regex	Logical; Indicates whether variable should be treated as regular expressions. Defaults to FALSE.
smooth_se	Logical; should the residual smoothing line display a standard-error ribbon.
smooth_method	Smoother passed to <code>ggplot2::geom_smooth()</code> for the residual-versus-fitted plot. The default is <code>"lm"</code> .
ppc_stat	Summary statistic to use when plots includes <code>"ppc_stat"</code> . This value is passed to <code>brms::pp_check()</code> as <code>type = "stat"</code> , <code>stat = 'ppc_stat'</code> .
rank_overlay	Logical; if TRUE, sets the bayesplot color scheme to <code>"blue"</code> . This does not itself create a rank-overlay plot, but allows a simple style customization for compatible bayesplot -based outputs.
add_plot_df	Optional logical (default FALSE) indicating whether to add <code>plot_df</code> to the returned value.
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 <code>bayes_R2</code> during model optimization. • FALSE (default): Stan functions are not exposed. • NULL: The setting is inherited from the original <code>bsitar()</code> model fit configuration. Note: In the <code>optimize_model()</code> 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 <code>expose_functions</code> argument from the <code>bsitar()</code> call. Manual specification of <code>usesavedfuns</code> 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, <code>clearenvfuns</code> is set based on the value of <code>usesavedfuns</code> : TRUE if <code>usesavedfuns = TRUE</code> , or FALSE if <code>usesavedfuns = FALSE</code> .
category	For <i>some</i> ordinal, multinomial, and multivariate models (notably, <code>brms::brm()</code> models but <i>not</i> <code>rstanarm::stan_polr()</code> models), multiple sets of rows will be

returned per input row for `epred_draws()` or `predicted_draws()`, depending on the model type. For ordinal/multinomial models, these rows correspond to different categories of the response variable. For multivariate models, these correspond to different response variables. The `category` argument specifies the name of the column to put the category names (or variable names) into in the resulting data frame. The default name of this column (`".category"`) reflects the fact that this functionality was originally used only for ordinal models and has been re-used for multivariate models. The fact that multiple rows per response are returned only for some model types reflects the fact that tidybayes takes the approach of tidying whatever output is given to us, and the output from different modeling functions differs on this point. See `vignette("tidy-brms")` and `vignette("tidy-rstanarm")` for examples of dealing with output from ordinal models using both approaches.

<code>wrap_title</code>	Optional logical indicating whether to wrap plot title. If <code>NULL</code> (default), then <code>wrap_title</code> is internally set as <code>TRUE</code> when three or more plots are requested.
<code>title_size</code>	Text size of plot title.
<code>each_object</code>	Optional logical (default <code>FALSE</code>) indicating whether to return each plot object (combined and individual plots) or only the combined when multiple plots are requested.
<code>qq_plot_args</code>	A named list of arguments passed on to the <code>"qq_plot_pearson"</code> . Ignored except when <code>plots = "qqp"</code> or <code>plots = "qq"</code> . Note that <code>plots = "qq"</code> is internally evaluated as <code>plots = "qqp"</code> .
<code>patch_plot.margin</code>	Optional setting the margins for the patchwork .
<code>funlist</code>	Currently ignored and serves as a placeholder. It will be implemented in future updates.
<code>future</code>	Currently ignored and serves as a placeholder.
<code>future_session</code>	Currently ignored and serves as a placeholder.
<code>future_splits</code>	Currently ignored and serves as a placeholder.
<code>future_method</code>	Currently ignored and serves as a placeholder.
<code>future_re_expose</code>	Currently ignored and serves as a placeholder.
<code>transform</code>	Currently ignored and serves as a placeholder.
<code>transform_draws</code>	Currently ignored and serves as a placeholder.
<code>itransform</code>	Currently ignored and serves as a placeholder.
<code>model_deriv</code>	Currently ignored and serves as a placeholder.
<code>dummy_to_factor</code>	Currently ignored and serves as a placeholder.
<code>xvar</code>	Currently ignored and serves as a placeholder.
<code>difx</code>	Currently ignored and serves as a placeholder.
<code>idvar</code>	Currently ignored and serves as a placeholder.

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.
conf_level	Currently ignored and serves as a placeholder.
deriv	Currently ignored and serves as a placeholder.
plot	Currently ignored and serves as a placeholder.
method	Currently ignored and serves as a placeholder.
ipts	Currently ignored and serves as a placeholder.
idata_method	Currently ignored and serves as a placeholder.
newdata_fixed	Currently ignored and serves as a placeholder.
verbose	A logical argument (default FALSE) to specify whether to print information collected during the setup of the object(s).
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

This function provides a single interface to several common Bayesian model diagnostics. Residual-based plots are constructed with **tidybayes**, using posterior residual draws together with fitted-value draws from [tidybayes::add_epred_draws\(\)](#), [tidybayes::add_linpred_draws\(\)](#), or [tidybayes::add_predicted_draws\(\)](#). MCMC diagnostics are produced through 'mcmc_plot' which is a wrapper around plotting functionality from **bayesplot**. Posterior predictive checks are produced through [brms::pp_check\(\)](#)

Residual-based diagnostics are created by joining posterior residual draws from [tidybayes::add_residual_draws\(\)](#) with fitted-value draws from one of [tidybayes::add_epred_draws\(\)](#), [tidybayes::add_linpred_draws\(\)](#), or [tidybayes::add_predicted_draws\(\)](#). The resulting joined draws are used for the residual-versus-fitted plot, and the residuals are summarized with [tidybayes::median_qi\(\)](#) before creating the Q-Q plot. The **tidybayes** documentation notes that the default output column names depend on the draw function: ".epred", ".linpred", ".prediction", and ".residual".

MCMC diagnostic plots are delegated to 'mcmc_plot' which provides a convenient wrapper around **bayesplot** diagnostics such as trace plots, autocorrelation plots, density overlays, R-hat, and effective sample size summaries.

Posterior predictive checks are delegated to [brms::pp_check\(\)](#), which exposes **bayesplot** PPC types such as density overlays, histograms, average scatter checks, and statistic-based checks.

Value

A list. The returned structure depends on combine and the number of plots requested:

If combine = FALSE or only one plot is requested A named list of plot objects.

If combine = TRUE and multiple compatible plots are requested A list with components:

combined A patchwork object combining all patchable plots.

plots A named list containing all individual plot objects.

plot_df The joined residual/fitted draws data frame used for residual-based plots, or NULL if no residual-based plots were requested.

Interpretation

- For Gaussian models, residual-versus-fitted plots and Q-Q plots are often useful summaries of residual behavior.
- set_draws = "epred" is usually the best default because expected predictions are often easier to interpret than linear predictors or raw posterior predictive draws.

Author(s)

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

See Also

`brms::pp_check()`, `brms::mcmc_plot()`, `tidybayes::add_residual_draws()`, `tidybayes::add_epred_draws()`, `tidybayes::add_linpred_draws()`, `tidybayes::add_predicted_draws()`, `patchwork::wrap_plots()`

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)

# Residual vs Fitted plot
res1 <- plot_diagnostics(
  model = model,
  plots = "rvf"
)

# MCMC diagnostics for selected parameters
res2 <- plot_diagnostics(
  model = model,
  plots = c("trace", "dens_overlay"),
  variable = c("b_a_Intercept", "b_b_Intercept")
)
res2$combined
```

```

# Posterior predictive checks
res3 <- plot_diagnostics(
  model = model,
  plots = c("ppc_overlay", "ppc_scatter"),
  ndraws = 10,
  ppc_stat = "sd"
)
res3$combined

# Use linear predictor instead of expected predictions
res4 <- plot_diagnostics(
  model = model,
  plots = "rvf",
  set_draws = "linpred"
)

```

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",
  bounds = NULL,
  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,
    expose_function = FALSE,
    usesavedfuns = NULL,
    clearenvfuns = NULL,
    newdata_fixed = NULL,
    envir = NULL,
    ...
)

plot_ppc(model, ...)

```

Arguments

model	An object of class <code>bgmfit</code> .
type	Type of the ppc plot as given by a character string. See PPC for an overview of currently supported types. You may also use an invalid type (e.g. <code>type = "xyz"</code>) to get a list of supported types in the resulting error message.
ndraws	A positive integer indicating the number of posterior draws to use in estimation. If <code>NULL</code> (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 <code>NULL</code>).
prefix	The prefix of the bayesplot function to be applied. Either <code>"ppc"</code> (posterior predictive check; the default) or <code>"ppd"</code> (posterior predictive distribution), the latter being the same as the former except that the observed data is not shown for <code>"ppd"</code> .
group	Optional name of a factor variable in the model by which to stratify the ppc plot. This argument is required for <code>ppc *_grouped</code> types and ignored otherwise.
x	Optional name of a variable in the model. Only used for ppc types having an <code>x</code> argument and ignored otherwise.
newdata	An optional data frame for estimation. If <code>NULL</code> (default), <code>newdata</code> is retrieved from the model.
resp	A character string (default <code>NULL</code>) to specify the response variable when processing posterior draws for <code>univariate_by</code> and <code>multivariate</code> models. See bsitar() for details on <code>univariate_by</code> and <code>multivariate</code> models.

size, alpha	Passed to the appropriate geom to control the appearance of the predictive distributions.
trim	A logical scalar passed to <code>ggplot2::geom_density()</code> .
bw	The smoothing bandwidth to be used. The kernels are scaled such that this is the standard deviation of the smoothing kernel (Note this differs from the reference books cited below). bw can also be a character string giving a rule to choose the bandwidth. The default, "nrd0" has remained the default for historical and compatibility reasons, rather than as a general recommendation, where "SJ" would rather fit.
adjust	the bandwidth used is actually <code>adjust*bw</code> . This makes it easy to specify values like 'half the default' bandwidth.
kernel	Kernel. See list of available kernels in <code>density()</code> .
bounds	Known lower and upper bounds for estimated data. Default <code>c(-Inf, Inf)</code> means that there are no (finite) bounds. If any bound is finite, boundary effect of default density estimation will be corrected by reflecting tails outside bounds around their closest edge. Data points outside of bounds are removed with a warning.
n_dens	number of equally spaced points at which the density is to be estimated, should be a power of two, see <code>stats::density()</code> for details.
pad	A logical scalar passed to <code>ggplot2::stat_ecdf()</code> .
discrete	[Deprecated] The discrete argument is deprecated. The ECDF is a step function by definition, so <code>geom_step()</code> is now always used.
binwidth	Passed to <code>ggplot2::geom_histogram()</code> , <code>ggplot2::geom_area()</code> , and <code>ggdist::stat_dots()</code> to override the default binwidth.
bins	Passed to <code>ggplot2::geom_histogram()</code> and <code>ggplot2::geom_area()</code> to override the default binning.
breaks	Passed to <code>ggplot2::geom_histogram()</code> as an alternative to binwidth.
freq	For histograms and frequency polygons, <code>freq=TRUE</code> (the default) puts count on the y-axis. Setting <code>freq=FALSE</code> 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 <code>yaxis_text()</code> convenience function.)
y_draw	For <code>ppc_violin_grouped()</code> , a string specifying how to draw y: "violin" (default), "points" (jittered points), or "both".
y_jitter, y_size, y_alpha	For <code>ppc_violin_grouped()</code> , if <code>y_draw</code> is "points" or "both" then <code>y_size</code> , <code>y_alpha</code> , and <code>y_jitter</code> are passed to to the size, alpha, and width arguments of <code>ggplot2::geom_jitter()</code> to control the appearance of y points. The default of <code>y_jitter=NULL</code> 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).
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 <code>expose_functions</code> argument from the <code>bsitar()</code> call. Manual specification of <code>usesavedfuns</code> is rarely needed and is intended for internal testing, as improper use can lead to unreliable estimates.
clearenvfun	A logical value indicating whether to clear the exposed Stan functions from the environment (TRUE) or not (FALSE). If NULL, <code>clearenvfun</code> is set based on the value of <code>usesavedfuns</code> : TRUE if <code>usesavedfuns</code> = TRUE, or FALSE if <code>usesavedfuns</code> = FALSE.
newdata_fixed	An indicator specifying how to handle the format and structure of the user-provided newdata before use. The available options are: • NULL (default): Checks the data format and structure of newdata, and applies <code>prepare_data2</code> and <code>prepare_transformations</code> as needed. • TRUE: Returns newdata as-is, without any format checking or transformation. • 0: Same as TRUE — returns newdata without checking the data format or applying <code>prepare_data2</code> and <code>prepare_transformations</code>. • 1: Checks the data format and, if needed, prepares it using <code>prepare_data2</code>, but does not apply <code>prepare_transformations</code>. • 2: Applies <code>prepare_transformations</code> only, assuming the data format is already correct. It is strongly recommended to leave both <code>newdata</code> = NULL and <code>newdata_fixed</code> = NULL (the defaults). In this case, the data used during model fitting is automatically retrieved, checked for the required format, and transformed as needed via <code>prepare_data2</code> and <code>prepare_transformations</code>. The flags <code>newdata_fixed</code> = 0, 1, and 2 are intended primarily for internal use.
envir	The environment used for function evaluation. The default is NULL, which sets the environment to <code>parent.frame()</code> . Since most post-processing functions rely on brms , it is recommended to set <code>envir</code> = <code>globalenv()</code> or <code>envir</code> = <code>.GlobalEnv</code> , especially for derivatives like velocity curves.
...	Additional arguments passed to the <code>brms::pp_check.brmsfit()</code> function. Please refer to <code>brms::pp_check.brmsfit()</code> 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 '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

<code>model</code>	An object of class <code>bgmfit</code> .
<code>newdata</code>	An optional data frame for estimation. If <code>NULL</code> (default), <code>newdata</code> is retrieved from the model.
<code>resp</code>	A character string (default <code>NULL</code>) to specify the response variable when processing posterior draws for <code>univariate_by</code> and <code>multivariate</code> models. See bsitar() for details on <code>univariate_by</code> and <code>multivariate</code> models.
<code>dpar</code>	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
<code>ndraws</code>	A positive integer indicating the number of posterior draws to use in estimation. If <code>NULL</code> (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 were 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, <code>avg_reeffects = list(feby = 'study', reby = NULL, over = 'age')</code> .
aux_variables	An optional argument to specify the variable(s) that can be passed to the <code>ipts</code> 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 <code>aux_variables</code> .
grid_add	An optional argument to specify the variable(s) that can be passed to the <code>marginaleffects::datagrid()</code> . 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 <code>grid_add</code> . Note that unlike <code>aux_variables</code> which are passed to the internal data functions such as <code>'get.newdata'</code> , the <code>grid_add</code> are passed to the <code>marginaleffects::datagrid()</code> .
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 <code>length.out</code> argument for <code>seq()</code> , controlling the smoothness of distance and velocity curves without altering the predictor range. NULL (the default) Engages automatic behavior based on the <code>dpar</code> argument: it is internally set to 50 for the mean response (<code>dpar = 'mu'</code>), ensuring smooth curves, while for the distributional parameters (e.g., <code>dpar = 'sigma'</code>), 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: <code>fitted_draws()</code> , <code>predict_draws()</code> , <code>growthparameters()</code> , <code>plot_curves()</code> , <code>get_predictions()</code> , <code>get_comparisons()</code> , and <code>get_growthparameters()</code> .
deriv	An integer indicating whether to estimate the distance curve or its derivative (velocity curve). The default <code>deriv = 0</code> is for the distance curve, while <code>deriv = 1</code> 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 <code>model_deriv = TRUE</code> for functions that require the velocity curve, such as <code>growthparameters()</code> and <code>plot_curves()</code> . Set it to <code>NULL</code> for functions that use the distance curve (i.e., fitted values), such as <code>loo_validation()</code> and <code>plot_ppc()</code> .
summary	A logical value indicating whether only the estimate should be computed (<code>TRUE</code>), or whether the estimate along with SE and CI should be returned (<code>FALSE</code> , default). Setting <code>summary</code> to <code>FALSE</code> will increase computation time. Note that <code>summary = FALSE</code> is required to obtain correct estimates when <code>re_formula = NULL</code> .
robust	A logical value to specify the summary options. If <code>FALSE</code> (default), the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If <code>TRUE</code> , the median and median absolute deviation (MAD) are applied instead. Ignored if <code>summary</code> is <code>FALSE</code> .

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 <code>optimize_model()</code> 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 <code>bsitar()</code> 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 off 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 specifying how to handle the format and structure of the user-provided newdata before use. The available options are: • NULL (default): Checks the data format and structure of newdata, and applies prepare_data2 and prepare_transformations as needed. • TRUE: Returns newdata as-is, without any format checking or transformation. • 0: Same as TRUE — returns newdata without checking the data format or applying prepare_data2 and prepare_transformations. • 1: Checks the data format and, if needed, prepares it using prepare_data2, but does not apply prepare_transformations. • 2: Applies prepare_transformations only, assuming the data format is already correct. It is strongly recommended to leave both newdata = NULL and newdata_fixed = NULL (the defaults). In this case, the data used during model fitting is automatically retrieved, checked for the required format, and transformed as needed via prepare_data2 and prepare_transformations. The flags newdata_fixed = 0, 1, and 2 are intended primarily for internal use.
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 <code>brms::predict.brmsfit()</code> function. Please see <code>brms::predict.brmsfit()</code> 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)
```

prior_conflict.bgmfit *Flag parameters with possible prior-data conflict*

Description

Screen [prior_sensitivity\(\)](#) results for parameters showing notable prior-data conflict, likelihood conflict, or both. Parameters flagged for both forms of sensitivity may indicate possible prior-data conflict in the **priorsense** framework, while parameters flagged primarily for prior sensitivity may reflect weak likelihood informativeness. See [prior_sensitivity\(\)](#) for details.

Usage

```
## S3 method for class 'bgmfit'
prior_conflict(
  x,
  threshold_prior = 0.05,
  threshold_lik = 0.05,
  empty = "-",
  print = FALSE,
  return_table = TRUE,
  tibble_table = FALSE,
  print_table = FALSE,
  return_file = NULL,
  flex_table = FALSE,
  path = NULL,
  title = NULL,
  align = "center",
  sheet_name = "table",
  ...
)

prior_conflict(x, ...)
```

Arguments

x	An object returned by prior_sensitivity() .
threshold_prior	Numeric threshold used to flag prior sensitivity.
threshold_lik	Numeric threshold used to flag likelihood sensitivity.
empty	A string that is used to replace NULL values.
print	A logical to print the object.
return_table	A logical indicating whether to return the table <code>return_table = TRUE</code> or the list <code>return_table = FALSE</code> .
tibble_table	A logical indicating whether to return the table as a data frame (<code>tibble_table = FALSE</code>) or as a tibble (<code>tibble_table = TRUE</code>). Default <code>FALSE</code> . Ignored if <code>return_table = FALSE</code> .

print_table	A logical indicating whether to print the table using <code>knitr::kable()</code> by setting <code>print_table = TRUE</code> , or return it as an object by setting <code>print_table = FALSE</code> . If <code>print_table = TRUE</code> , the table is printed but not returned; the function returns NULL invisibly. The default is FALSE. This argument is ignored when <code>return_table = FALSE</code> .
return_file	An optional character string specifying the output format. Supported values are "word", "docx", "html", "png", "pdf", "svg", and "xlsx". If NULL, the function returns either a data frame or a flextable object, depending on the value of <code>flex_table</code> . This argument is ignored when <code>path</code> is specified.
flex_table	A logical indicating whether to return the data frame (<code>flex_table = FALSE</code>) or a flextable object (<code>flex_table = TRUE</code>).
path	An optional character string specifying the output file path. If NULL, a default file name is generated based on the value of <code>return_file</code> ; for example, "table_output.docx" or "table_output.xlsx". If no file extension is specified and <code>return_file = NULL</code> , the default extension is ".xlsx". Otherwise, the file extension is set to match <code>return_file</code> .
title	Optional title used in exported output where supported. For Word and HTML output, named flextable objects can be used as document titles or section titles.
align	Alignment used for Word export. Must be one of "left", "center", or "right".
sheet_name	Character string giving the worksheet name for Excel output. Default is "table".
...	Ignored.

Details

This helper is designed as a lightweight screening step after `prior_sensitivity()`. It does not replace graphical inspection. The recommended workflow is:

1. Run `prior_sensitivity()` to compute sensitivity.
2. Use `prior_conflict()` to identify parameters of potential concern.
3. Re-run `prior_sensitivity()` with `plot = "dens"`, `plot = "ecdf"`, or `plot = "quantities"` for the flagged variables.

Else, user can call `prior_conflict()` directly from within the `prior_sensitivity()` by setting the argument `return_conflicts` as TRUE. See `prior_sensitivity()` for details and examples.

Value

A list when `return_table = FALSE` or a flextable if `return_table = TRUE`

prior_flagged Character vector of variables flagged for prior sensitivity.

likelihood_flagged Character vector of all variables flagged for likelihood sensitivity.

prior_likelihood_flagged Character vector of variables flagged for both prior and likelihood sensitivity

See Also

[prior_sensitivity\(\)](#)

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)

# Note return_table = FALSE which ensures correct output is returned from
# the prior_sensitivity()
ps <- prior_sensitivity(
  model = model,
  variable = c("b_a_Intercept", "sigma"),
  return_table = FALSE
)

conflict <- prior_conflict(ps, return_table = TRUE)
print(conflict)

# Note that you can call prior_conflict() from within the prior_sensitivity()
# with return_conflict = TRUE that returns the same results as above,
conflict2 <- prior_sensitivity(
  model = model,
  variable = c("b_a_Intercept", "sigma"),
  return_conflict = TRUE, return_table = TRUE
)
stopifnot(identical(conflict, conflict2))
```

prior_sensitivity.bgmfit

*Prior sensitivity analysis workflow for **bsitar** models*

Description

Perform power-scaling sensitivity analysis for a fitted **bsitar** model. Supports appending derived quantities to a posterior draws object as the recommended **priorsense** workflow. Furthermore, **prior_sensitivity** also allows visualization of sensitivity analysis plot(s).

Usage

```
## S3 method for class 'bgmfit'
prior_sensitivity(
  model,
  variable = NULL,
  lower_alpha = 0.99,
  upper_alpha = 1.01,
```

```
div_measure = "cjs_dist",
measure_args = list(),
component = c("prior", "likelihood"),
sensitivity_threshold = 0.05,
moment_match = FALSE,
k_threshold = 0.5,
resample = FALSE,
transform = NULL,
prediction = NULL,
prior_selection = NULL,
likelihood_selection = NULL,
num_args = NULL,
include_criterion = NULL,
include_jointlik = NULL,
include_predictor = NULL,
name_predictor = NULL,
plot = NULL,
plot_variable = NULL,
plot_print = FALSE,
plot_return = FALSE,
facet = NULL,
empty = "-",
print = FALSE,
return_table = NULL,
tibble_table = FALSE,
print_table = FALSE,
return_file = NULL,
flex_table = FALSE,
return_conflict = FALSE,
digits = NULL,
path = NULL,
title = NULL,
align = "center",
sheet_name = "table",
criterion = "bayes_R2",
pointwise = FALSE,
model_name = NULL,
overwrite = FALSE,
force_save = FALSE,
resp = NULL,
dpar = NULL,
re_formula = NULL,
allow_new_levels = FALSE,
sample_new_levels = "uncertainty",
incl_autocor = TRUE,
summary = FALSE,
robust = FALSE,
scale = c("response", "linear"),
```

```

    probs = c(0.025, 0.975),
    verbose = FALSE,
    expose_function = FALSE,
    usesavedfuns = NULL,
    clearenvfuns = NULL,
    xvar = NULL,
    envir = NULL,
    ...
)

prior_sensitivity(model, ...)

```

Arguments

model	A fitted model object from bsitar.
variable	Optional character vector of parameter or quantity names to assess. Passed to <code>priorsense::powerscale_sensitivity()</code> and, when plotting, to the corresponding priorsense plotting function.
lower_alpha	Lower alpha value for gradient calculation.
upper_alpha	Upper alpha value for gradient calculation.
div_measure	Character (case sensitive) specifying the divergence measure to use. The following methods are implemented: • "cjs_dist": Cumulative Jensen-Shannon distance. Default method. See function <code>cjs_dist</code> for more details. • "js_dist": Jensen-Shannon distance. • "js_div": Jensen-Shannon divergence. • "hellinger_dist": Hellinger distance. • "kl_dist": Kullback-Leibler distance. • "kl_div": Kullback-Leibler divergence. • "ks_dist": Kolmogorov-Smirnov distance. • "hellinger_dist": Hellinger distance. • "ws_dist": Wassterstein distance (pass <code>measure_args = list(p = N)</code>) for a different order, where N is the order.
measure_args	Named list of further arguments passed to divergence measure functions.
component	Character vector specifying component(s) to scale (default is both "prior" and "likelihood").
sensitivity_threshold	Threshold for flagging variable as sensitive to power-scaling.
moment_match	Logical; Indicate whether or not moment matching should be performed. Can only be TRUE if <code>is_method</code> is "psis".
k_threshold	Threshold value for Pareto k values above which the moment matching algorithm is used. Default is 0.5.
resample	Logical; Indicate whether or not draws should be resampled based on calculated importance weights.

transform	Indicate a transformation of posterior draws to perform before sensitivity analysis. Either "scale" or "whiten".
prediction	Function taking the model fit and returning a draws_df of predictions to be appended to the posterior draws
prior_selection	Vector specifying partitions of component to be included in power-scaling. Default is NULL, which takes all partitions. If this is a character, then it is appended to the variable name (specified by log_prior_name) with an _ between them. If numeric, then it is appended inside [].
likelihood_selection	Vector specifying partitions of component to be included in power-scaling. Default is NULL, which takes all partitions. If this is a character, then it is appended to the variable name (specified by log_lik_name) with an _ between them. If numeric, then it is appended inside [].
num_args	(named list) Optional arguments passed to num() for pretty printing of summaries. Can be controlled globally via the posterior.num_args option.
include_criterion	Logical; if TRUE, appends one or more of the following fit criteria using add_model_criterion(): "bayes_R2", "loo_R2", "jointlik", or "margliklik".
include_jointlik	Logical; if TRUE, appends a joint log-likelihood quantity using priorsense::predictions_as_draws with the custom jointlik_fun function. If include_criterion contains "jointlik", include_jointlik is automatically set to TRUE.
include_predictor	A character string or named list specifying predictor-based quantities to append as expected values at predictor values, using priorsense::predictions_as_draws() with a custom call to brms::posterior_epred().
name_predictor	A character string or NULL (default). If NULL, name_predictor is set internally to xvar. Note that prediction_names are generated internally as paste0(name_predictor, include_predictor). These prediction_names can then be included as plot_variable or to get potential conflicts by running the prior_conflict().
plot	One of NULL, FALSE, "dens", "ecdf", "quantities", or "both". If NULL or FALSE, no plot is produced. If a string is supplied, the corresponding priorsense plot is created and returned in the output object.
plot_variable	Optional character string naming the variable to plot. If NULL, the first element of variable is used when available.
plot_print	Optional logical indicating whether to print plot object.
plot_return	Optional logical indicating whether to print plot object.
facet	Optional value passed to facet_rows in priorsense::powerscale_plot_dens() and priorsense::powerscale_plot_ecdf().
empty	A string that is used to replace NULL values.
print	A logical to print the object.
return_table	Optional logical indicating whether to return the raw sensitivity object from priorsense::powerscale_sensitivity() or a structured list with class attributes. Set return_table = FALSE if you plan to call prior_conflict()

	later. Note that when <code>return_table = NULL</code> (the default), then it is internally set as <code>TRUE</code> when <code>return_conflict = FALSE</code> .
<code>tibble_table</code>	A logical indicating whether to return the table as a data frame (<code>tibble_table = FALSE</code>) or as a tibble (<code>tibble_table = TRUE</code>). Default <code>FALSE</code> . Ignored if <code>return_table = FALSE</code> .
<code>print_table</code>	A logical indicating whether to print the table using <code>knitr::kable()</code> by setting <code>print_table = TRUE</code> , or return it as an object by setting <code>print_table = FALSE</code> . If <code>print_table = TRUE</code> , the table is printed but not returned; the function returns <code>NULL</code> invisibly. The default is <code>FALSE</code> . This argument is ignored when <code>return_table = FALSE</code> .
<code>return_file</code>	An optional character string specifying the output format. Supported values are "word", "docx", "html", "png", "pdf", "svg", and "xlsx". If <code>NULL</code> , the function returns either a data frame or a flextable object, depending on the value of <code>flex_table</code> . This argument is ignored when <code>path</code> is specified.
<code>flex_table</code>	A logical indicating whether to return the data frame (<code>flex_table = FALSE</code>) or a flextable object (<code>flex_table = TRUE</code>).
<code>return_conflict</code>	Optional logical indicating whether to return the raw sensitivity results <code>return_conflict = FALSE</code> from <code>priorsense::powerscale_sensitivity()</code> or the conflict <code>return_conflict = TRUE</code> as evaluated using the <code>prior_conflict()</code> . The <code>return_table</code> behavior is same whether <code>return_conflict = TRUE</code> or <code>return_conflict = FALSE</code> .
<code>digits</code>	Optional integer to pass on to <code>base::round()</code> applied to the numeric variables returned from the <code>priorsense::powerscale_sensitivity()</code> . When <code>digits = NULL</code> (default), <code>digits</code> is ignored.
<code>path</code>	An optional character string specifying the output file path. If <code>NULL</code> , a default file name is generated based on the value of <code>return_file</code> ; for example, "table_output.docx" or "table_output.xlsx". If no file extension is specified and <code>return_file = NULL</code> , the default extension is ".xlsx". Otherwise, the file extension is set to match <code>return_file</code> .
<code>title</code>	Optional title used in exported output where supported. For Word and HTML output, named flextable objects can be used as document titles or section titles.
<code>align</code>	Alignment used for Word export. Must be one of "left", "center", or "right".
<code>sheet_name</code>	Character string giving the worksheet name for Excel output. Default is "table".
<code>criterion</code>	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).
<code>pointwise</code>	A flag indicating whether to compute the full log-likelihood matrix at once or separately for each observation. The latter approach is usually considerably slower but requires much less working memory. Accordingly, if one runs into memory issues, <code>pointwise = TRUE</code> is the way to go.
<code>model_name</code>	Optional name of the model. If <code>NULL</code> (the default) the name is taken from the call to <code>x</code> .

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 <code>add_criterion</code> .
resp	A character string (default NULL) to specify the response variable when processing posterior draws for <code>univariate_by</code> and <code>multivariate</code> models. See <code>bsitar()</code> for details on <code>univariate_by</code> and <code>multivariate</code> models.
dpar	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
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 <code>numeric_cov_at</code> 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 <code>newdata</code> is provided.
sample_new_levels	Indicates how to sample new levels for grouping factors specified in <code>re_formula</code> . This argument is only relevant if <code>newdata</code> is provided and <code>allow_new_levels</code> is set to 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 were observed in the <code>old_data</code> . 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 <code>autocor</code> should be included in the predictions. Defaults 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 <code>summary</code> to FALSE will increase computation time. Note that <code>summary = FALSE</code> is required to obtain correct estimates when <code>re_formula = NULL</code> .
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.
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.
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 <code>optimize_model()</code> 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.
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'.
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 priorsense::powerscale_sensitivity().

Details

The **prior_sensitivity** function is intended as a package-level convenience wrapper around `priorsense::powerscale_sensitivity` and the associated plotting tools.

If requested, the function also constructs an augmented draws object using fit criterion (see argument `include_criterion`), joint log-likelihood (see argument `include_jointlik`), or posterior

expected predictions for representative covariate values (see argument `include_predictor`). This follows the general workflow recommended in the **priorsense** package.

The **prior_sensitivity** supports two closely related workflows for prior sensitivity analysis.

Standard workflow: The function calls `priorsense::powerscale_sensitivity()` directly on the fitted `bsitar` model and returns the sensitivity results. This is the simpler approach when you only need sensitivity assessments for the model parameters themselves.

Extended workflow: A posterior draws object is created using `posterior::as_draws_df()` which allows computing the additional derived quantities. This extended draws object is useful when user wants to explore prior sensitivity not only for model parameters but also for model-based fit criteria and predictive quantities. The posterior draws object can be extended with the following quantities:

- **Fit criteria** via `add_model_criterion()`. Use the `include_criterion` argument to specify which criteria to append. Available options include "bayes_R2", "loo_R2", "jointlik", and "margliklik". Note that for "margliklik", the model should have fitted using `save_pars = save_pars(all = TRUE)`. See `bsitar()` for details on the `save_pars` argument.
- **Joint log-likelihood** (`jointlik`) quantity computed using the custom `jointlik_fun` function. This captures the joint likelihood of all observations under each posterior draw. See `include_jointlik` argument for details.
- **Posterior expected predictions** via `brms::posterior_epred()`. Use the `include_predictor` argument to specify predictor values for which to compute expected predictions. See `include_predictor` argument for details.
- **Point-wise log-likelihood draws** via `priorsense::log_lik_draws()` (currently ignored)
- **Log-prior draws** via `priorsense::log_prior_draws()` (currently ignored)

Interpretation of sensitivity analysis results: In the **priorsense** framework, the pattern of sensitivity to prior and likelihood perturbations provides diagnostic information about the Prior-data conflict and the strength of likelihood:

- **Prior-data conflict:** When sensitivity to **both** prior perturbations **and** likelihood perturbations is strong, it indicates a prior-data conflict. The prior information is inconsistent with what the data suggest.
- **Weakly informative likelihood:** When sensitivity to prior perturbations is **strong** but sensitivity to likelihood perturbations is **weak**, this suggests that the likelihood is only weakly informative for that quantity. The data provide limited information relative to the prior.

These diagnostics help user to assess whether prior choices are appropriate and whether certain quantities are well-identified by the data.

Value

When `return_table = TRUE`, a table of sensitivity values for each specified variable is returned. If `return_table = FALSE`, a "prior_sensitivity" object is returned as a list with the following components:

model The extracted `bsitar` object.

post_draws The augmented draws object if any of `include_criterion`, `include_jointlik`, or `include_predictor` are specified, otherwise `NULL`.

sensitivity The object returned by `priorsense::powerscale_sensitivity()`.
plot A plot object, or list of plot objects when `plot = "both"`, otherwise NULL.
plot_type The requested plot type, if any.
plot_variable The plotted variable, if any.
call The matched function call.

See Also

`priorsense::powerscale_sensitivity()`, `priorsense::powerscale_plot_dens()`, `priorsense::powerscale_plot_quantities()`, `priorsense::predictions_as_draws()`, `prior_conflict()`

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)

# Basic workflow: use brmsfit support directly
ps <- prior_sensitivity(
  model = model,
  variable = c("b_a_Intercept", "sigma")
)

# Directly request a density plot
ps2 <- prior_sensitivity(
  model = model,
  variable = c("sigma"),
  plot = "dens"
)
ps2$plot

# Extended workflow with derived predictive quantities
# In `variable` and `plot_variable`, the custom variables `bayes_R2`,
# `jointlik`, `agemin`, and `agemax` are requested via the `criterion`
# and `include_predictor` arguments. Here, the predictor of interest is
# `age`, so `min` and `max` are prefixed with `age`, giving `agemin`
# and `agemax`. The predictor `age` is automatically inferred from the
# `model` object, but it can also be specified explicitly using `xvar`.

ps3 <- prior_sensitivity(
  model = model,
  variable = c("sigma", "bayes_R2", "jointlik", "agemin", "agemax"),
  criterion = c("bayes_R2", "jointlik"),
  include_predictor = c("min", "max"),
  plot = "ecdf",
  plot_variable = c("sigma", "bayes_R2", "jointlik", "agemin", "agemax")
)
```

```
# Note that user can also get conflict by call `prior_conflict()` from within
# the `prior_sensitivity()` by setting return_conflict = TRUE
conflict <- prior_sensitivity(
  model = model,
  variable = c("b_a_Intercept", "sigma"),
  return_conflict = TRUE, return_table = TRUE
)
```

prior_table.bgmfit *Create a prior summary table for the bsitar model*

Description

Builds a formatted summary table of user-specified prior distributions from a bsitar model. The table includes the prior distribution text, one or more simulated credible interval columns derived from draws from the parsed prior distributions, and a simulated empirical range column based on `base::range()`.

Usage

```
## S3 method for class 'bgmfit'
prior_table(
  model,
  set_width = 0.95,
  set_digits = 1,
  empty = "-",
  print = FALSE,
  return_table = TRUE,
  tibble_table = FALSE,
  print_table = FALSE,
  return_file = NULL,
  flex_table = FALSE,
  path = NULL,
  title = NULL,
  align = "center",
  sheet_name = "table",
  draw_samples = 1e+05,
  add_range = FALSE,
  transform_class = NULL,
  transform_parameter = NULL,
  transform_fun = NULL,
  range_method_arg = NULL,
  seed = 123,
  verbose = FALSE,
```

```

    ...
  )

prior_table(model, ...)

```

Arguments

<code>model</code>	A fitted bsitar model object.
<code>set_width</code>	A numeric vector of interval masses to report, such as 0.95 or c(0.95, 0.99). Each value creates one CI column.
<code>set_digits</code>	An integer giving the number of digits used when extracting priors from <code>priors_to_textdata()</code> .
<code>empty</code>	A string that is used to replace NULL values.
<code>print</code>	A logical to print the object.
<code>return_table</code>	A logical indicating whether to return the table <code>return_table = TRUE</code> or the list <code>return_table = FALSE</code> .
<code>tibble_table</code>	A logical indicating whether to return the table as a data frame (<code>tibble_table = FALSE</code>) or as a tibble (<code>tibble_table = TRUE</code>). Default <code>FALSE</code> . Ignored if <code>return_table = FALSE</code> .
<code>print_table</code>	A logical indicating whether to print the table using <code>knitr::kable()</code> by setting <code>print_table = TRUE</code> , or return it as an object by setting <code>print_table = FALSE</code> . If <code>print_table = TRUE</code> , the table is printed but not returned; the function returns NULL invisibly. The default is <code>FALSE</code> . This argument is ignored when <code>return_table = FALSE</code> .
<code>return_file</code>	An optional character string specifying the output format. Supported values are "word", "docx", "html", "png", "pdf", "svg", and "xlsx". If NULL, the function returns either a data frame or a flextable object, depending on the value of <code>flex_table</code> . This argument is ignored when <code>path</code> is specified.
<code>flex_table</code>	A logical indicating whether to return the data frame (<code>flex_table = FALSE</code>) or a flextable object (<code>flex_table = TRUE</code>).
<code>path</code>	An optional character string specifying the output file path. If NULL, a default file name is generated based on the value of <code>return_file</code> ; for example, "table_output.docx" or "table_output.xlsx". If no file extension is specified and <code>return_file = NULL</code> , the default extension is ".xlsx". Otherwise, the file extension is set to match <code>return_file</code> .
<code>title</code>	Optional title used in exported output where supported. For Word and HTML output, named flextable objects can be used as document titles or section titles.
<code>align</code>	Alignment used for Word export. Must be one of "left", "center", or "right".
<code>sheet_name</code>	Character string giving the worksheet name for Excel output. Default is "table".
<code>draw_samples</code>	Integer number of simulated draws used per prior distribution to compute both the reported CI columns and the empirical range column. Note that the default value is 100000 to approximate the stabilized distribution. However, it make take some time (approximate one minute).
<code>add_range</code>	Logical indicating whether to include range column in the returned object.

transform_class	Optional character vector of class values to transform. These are expanded with transform_parameter using a Cartesian product. Any supplied value must exist among the available classes in the parsed priors.
transform_parameter	Optional character vector of parameter values to transform. These are expanded with transform_class using a Cartesian product. Any supplied value must exist among the available parameters in the parsed priors.
transform_fun	Optional function or list of functions controlling the transformation to apply. Accepted forms are: • A single function, recycled to all targeted combinations. • A list of functions of length length(transform_parameter), matched to parameters and recycled across classes. • A list of functions of length length(transform_class) * length(transform_parameter), matched directly to all expanded combinations. Each function must accept a numeric vector and return a numeric vector of the same length.
range_method_arg	An optional named list to pass arguments to the 'range_method()'.
seed	Integer random seed used before drawing simulated values from the parsed prior distributions.
verbose	Logical. If TRUE, prints information.
...	Ignored.

Details

The function parses priors returned by `priors_to_textdata()`, simulates draws from each parsed prior distribution, computes interval summaries with `ggdist::point_interval()`, and formats the result as a flextable. If a tag column is present but entirely empty, it is removed before the table is built.

Optional transformations can be applied to selected combinations of class and parameter. Transformation targets are defined by the Cartesian product of all values supplied in `transform_class` and all values supplied in `transform_parameter`. This allows any number of available classes and any number of available parameters to be selected.

More formally, if `transform_class` contains the selected classes and `transform_parameter` contains the selected parameters, then the transformation targets all combinations in `transform_class` x `transform_parameter`.

The `transform_fun` argument controls which function is applied:

- If a single function is supplied, it is applied to every targeted class/parameter combination.
- If a list of functions with length equal to `length(transform_parameter)` is supplied, the functions are matched to parameters and recycled across all selected classes.
- If a list of functions with length equal to `length(transform_class) * length(transform_parameter)` is supplied, the functions are matched directly to the fully expanded class/parameter combinations in `expand.grid()` order.

Transformations are applied to simulated draws before the credible intervals and empirical ranges are computed. This means the reported summaries reflect the transformed scale directly.

The function validates requested transformation classes and parameters against the values actually present in the parsed prior summary. If a user supplies a class or parameter that does not exist, the function stops with an informative error.

Footnote markers are assigned dynamically in sequence. The first four markers are always used for Class, Parameter, Coefficient, and Prior distribution. Credible interval columns are assigned the next available letters, the Range column receives the next letter after the final CI column, and the transformation mapping note receives the next letter after the Range note.

The Range column is not an analytic support bound. It is a simulated empirical range computed from `draw_samples` draws from each parsed prior distribution. Consequently, the reported range depends slightly on the random seed, the number of simulated draws, and any transformation applied.

Credible interval columns are also simulation-based. They are computed from the same simulated draws used for the range column, which ensures consistent row alignment between interval summaries and range summaries.

Requested transformations are validated in two stages. First, every supplied class must exist among available classes and every supplied parameter must exist among available parameters. Second, every expanded class/parameter combination must exist in the parsed prior table; otherwise the function stops with an error.

Transformations are applied to simulated draws before summary computation. This is generally preferable to transforming already-computed interval endpoints because the summaries then directly reflect the transformed distribution.

If the resulting summary data contains a tag column and every entry is either NA or an empty string after trimming whitespace, that column is removed automatically before constructing the flextable.

Value

A flextable object when `return_table = TRUE`. Otherwise returns `invisible(NULL)` after optionally saving the table as `.docx` or `.xlsx`.

Author(s)

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

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)

prior_table(
  model = model,
```

```

    print_table = TRUE
  )

  # Apply 'exp' transformation to the prior for 'c' parameter
  prior_table(
    model = model,
    transform_class = c("b"),
    transform_parameter = c("c"),
    transform_fun = function(x) exp(x),
    print_table = TRUE
  )

```

update_model.bgmfit	<i>Update the Bayesian SITAR model</i>
---------------------	--

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,
  new_threads = FALSE,
  envir = NULL,
  ...
)

update_model(model, ...)

```

Arguments

<code>model</code>	An object of class <code>bgmfit</code> .
<code>newdata</code>	An optional <code>data.frame</code> to be used when updating the model. If <code>NULL</code> (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 <code>bsitar()</code> model fit configuration. Note: In the <code>optimize_model()</code> 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.
new_threads	A logical (default FALSE) indicating whether to drop the threads i.e., setting threads = NULL (by using new_threads = NULL or new_threads = NA). The threads argument sets the number of threads used for within-chain parallelization. See <code>bsitar()</code> for details. The new_threads is rarely used as threads is automatically inferred from the model. The only indication of setting new_threads = NULL is when <code>brms::brm()</code> does support within-chain parallelization. such as in case of modelling autocorrelation of residuals (See argument autocor_formula).
envir	The environment used for function evaluation. The default is NULL, which sets the environment to <code>parent.frame()</code> . Since most post-processing functions rely on brms , it is recommended to set <code>envir = globalenv()</code> or <code>envir = .GlobalEnv</code> , especially for derivatives like velocity curves.
...	Other arguments passed to <code>[brms::brm()]</code> .

Details

This function is an adapted version of the **update()** function from the **brms** package.

Value

An updated object of class `bgmfit`.

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, [7](#)
 berkeley_exdata, [8](#)
 berkeley_exfit, [9](#)

add_model_criterion
 (add_model_criterion.bgmfit), [3](#)
add_model_criterion(), [51](#), [95](#), [96](#), [157](#),
 [210](#), [214](#)
add_model_criterion.bgmfit, [3](#)

base::jitter(), [38](#)
base::on.exit(), [74](#), [91](#), [110](#), [130](#), [131](#), [151](#)
base::round(), [66](#), [81](#), [97](#), [102](#), [118](#), [143](#),
 [158](#), [176](#), [211](#)
base::Vectorize(), [53](#)
bayesplot::mcmc_pairs(), [187](#)
bayestestR::equivalence_test(), [90](#), [122](#),
 [123](#), [127–133](#)
bayestestR::p_direction(), [90](#), [122](#), [123](#),
 [127–133](#)
berkeley, [7](#), [8](#)
berkeley_exdata, [8](#), [9](#)
berkeley_exfit, [9](#)
brms::add_criterion(), [3](#), [138](#), [159](#)
brms::add_ic(), [6](#), [97](#)
brms::add_loo, [3](#), [6](#), [97](#)
brms::add_waic(), [6](#), [97](#)
brms::bayes_R2(), [3](#), [6](#), [97](#)
brms::bridge_sampler(), [41](#)
brms::brm(), [27](#), [37](#), [39–44](#), [48](#), [137](#), [189](#), [221](#)
brms::brmsformula(), [19](#), [28](#), [39](#), [42](#), [48](#)
brms::conditional_effects, [172](#)
brms::conditional_effects(), [172](#)
brms::custom_family(), [28](#)
brms::fitted.brmsfit, [61](#), [112](#)
brms::fitted.brmsfit(), [54](#), [61](#), [93](#), [112](#),
 [116](#), [167](#), [178](#)
brms::get_prior(), [39](#)
brms::hypothesis(), [42](#), [122](#), [126](#), [131–133](#)
brms::lf(), [22](#)
brms::loo(), [136](#), [138](#), [139](#)
brms::loo_compare(), [50](#), [51](#)
brms::loo_moment_match(), [137](#)
brms::mcmc_plot(), [188](#), [192](#)
brms::nlf(), [22](#)
brms::opencl(), [41](#)
brms::posterior_epred(), [210](#), [214](#)
brms::posterior_linpred(), [139](#)
brms::posterior_predict(), [58](#), [71](#), [87](#),
 [108](#), [115](#), [148](#), [168](#), [179](#), [201](#)
brms::pp_check(), [188](#), [189](#), [191](#), [192](#), [196](#)
brms::pp_check.brmsfit(), [196](#)
brms::predict.brmsfit(), [112](#), [116](#), [167](#),
 [178](#), [197](#), [203](#), [204](#)
brms::prior(), [28](#), [46](#), [48](#)
brms::ranef(), [159](#), [161](#), [164](#)
brms::reloo(), [137](#)
brms::save_pars(), [42](#)
brms::set_prior(), [42](#)
brms::stancode(), [39](#)
brms::standata(), [39](#)
brms::stanvar(), [39](#)
brms::validate_prior(), [39](#)
bsitar, [9](#)
bsitar(), [5](#), [6](#), [15](#), [16](#), [25](#), [56](#), [60](#), [65](#), [75](#), [79](#),
 [92](#), [96](#), [100](#), [111](#), [115](#), [120](#), [131](#), [138](#),
 [141](#), [152](#), [158](#), [162](#), [169](#), [170](#), [175](#),
 [183](#), [189](#), [194](#), [196](#), [198](#), [202](#),
 [212–214](#), [221](#)
coef.brmsfit, [127](#)
compare_model(compare_models.bgmfit),
 [50](#)
compare_models(compare_models.bgmfit),
 [50](#)
compare_models.bgmfit, [50](#)
density(), [195](#)

- `estimate_density()`, [128](#)
- `expose_model_functions`
 - `(expose_model_functions.bgmfit)`, [52](#)
- `expose_model_functions()`, [51](#)
- `expose_model_functions.bgmfit`, [52](#)
- `fitted_draws(fitted_draws.bgmfit)`, [54](#)
- `fitted_draws()`, [57](#), [61](#), [67](#), [83](#), [98](#), [103](#), [116](#), [118](#), [121](#), [144](#), [167](#), [169](#), [177](#), [178](#), [184](#), [200](#)
- `fitted_draws.bgmfit`, [54](#)
- `foreach::foreach()`, [68](#), [83](#), [104](#), [145](#)
- `future`, [43](#)
- `future.apply::future_lapply()`, [68](#), [83](#), [104](#), [144](#)
- `future.apply::future_sapply()`, [67](#), [83](#), [104](#), [118](#), [144](#), [179](#)
- `future.mirai::mirai_cluster()`, [67](#), [83](#), [104](#), [118](#), [144](#), [179](#)
- `future::plan()`, [68](#), [83](#), [104](#), [145](#)
- `get_comparisons`
 - `(get_comparisons.bgmfit)`, [63](#)
- `get_comparisons()`, [57](#), [67](#), [83](#), [103](#), [118](#), [144](#), [169](#), [177](#), [200](#)
- `get_comparisons.bgmfit`, [63](#)
- `get_gp(get_growthparameters.bgmfit)`, [77](#)
- `get_growthparameters`
 - `(get_growthparameters.bgmfit)`, [77](#)
- `get_growthparameters()`, [57](#), [67](#), [83](#), [103](#), [113](#), [118](#), [129](#), [130](#), [132](#), [144](#), [169](#), [177](#), [200](#)
- `get_growthparameters.bgmfit`, [77](#)
- `get_model_criterion`
 - `(get_model_criterion.bgmfit)`, [95](#)
- `get_model_criterion.bgmfit`, [95](#)
- `get_predictions`
 - `(get_predictions.bgmfit)`, [98](#)
- `get_predictions()`, [54](#), [57](#), [67](#), [83](#), [98](#), [103](#), [118](#), [144](#), [169](#), [172](#), [173](#), [177](#), [197](#), [200](#)
- `get_predictions.bgmfit`, [98](#)
- `getNsObject`, [62](#)
- `ggdist::stat_dots()`, [195](#)
- `ggplot2::facet_wrap()`, [163](#)
- `ggplot2::geom_area()`, [195](#)
- `ggplot2::geom_density()`, [195](#)
- `ggplot2::geom_histogram()`, [195](#)
- `ggplot2::geom_jitter()`, [195](#)
- `ggplot2::geom_smooth()`, [189](#)
- `ggplot2::stat_ecdf()`, [195](#)
- `growthparameters`
 - `(growthparameters.bgmfit)`, [113](#)
- `growthparameters()`, [57](#), [67](#), [83](#), [103](#), [118](#), [144](#), [161](#), [169](#), [175](#), [177](#), [184](#), [200](#)
- `growthparameters.bgmfit`, [113](#)
- `growthparameters_comparison`
 - `(get_growthparameters.bgmfit)`, [77](#)
- `Hmisc::rcspline.eval()`, [15](#), [18](#)
- `hypothesis_test`
 - `(hypothesis_test.bgmfit)`, [122](#)
- `hypothesis_test()`, [128](#)
- `hypothesis_test.bgmfit`, [122](#)
- `installr::kill_all_Rscript_s()`, [68](#), [83](#), [104](#), [144](#)
- `is.bgmfit`, [135](#)
- `knitr::kable()`, [97](#), [206](#), [211](#), [217](#)
- `loo`, [137](#)
- `loo::loo_compare()`, [136](#), [138](#)
- `loo::loo_moment_match()`, [137](#)
- `loo_compare`, [4](#)
- `loo_moment_match`, [137](#)
- `loo_validation(loo_validation.bgmfit)`, [136](#)
- `loo_validation.bgmfit`, [136](#)
- `make_conditions`, [167](#)
- `marginal_comparison`
 - `(get_comparisons.bgmfit)`, [63](#)
- `marginal_comparisons`
 - `(get_comparisons.bgmfit)`, [63](#)
- `marginal_draws`
 - `(get_predictions.bgmfit)`, [98](#)
- `marginal_growthparameters`
 - `(get_growthparameters.bgmfit)`, [77](#)
- `marginaleffects::avg_comparisons()`, [63](#), [69](#), [76](#), [77](#), [84](#), [85](#), [94](#), [106](#), [145](#), [146](#)
- `marginaleffects::avg_predictions()`, [58](#), [71](#), [87](#), [98](#), [105](#), [108](#), [113](#), [115](#), [148](#), [168](#), [179](#), [201](#)

- `marginalEffects::comparisons()`, 63, 69, 70, 74, 76, 77, 84, 85, 90, 94, 106, 122, 126, 127, 129–133, 145, 146
- `marginalEffects::datagrid()`, 57, 65, 66, 80, 82, 91, 101, 103, 117, 141, 151, 176, 200
- `marginalEffects::plot_comparisons()`, 69, 76, 84, 94, 145
- `marginalEffects::plot_predictions()`, 98, 105, 113
- `marginalEffects::posterior_draws()`, 70, 86, 107, 147
- `marginalEffects::predictions()`, 58, 70, 71, 87, 98, 105, 106, 108, 113, 115, 148, 154, 168, 178, 179, 201
- `mirai::daemons()`, 67, 83, 104, 118, 144, 179
- `modelbased_growthparameters`
(`modelbased_growthparameters.bgmfit`), 139
- `modelbased_growthparameters.bgmfit`, 139

- `nlme::varConstPower()`, 23
- `nlme::varExp()`, 23
- `nlme::varPower()`, 23
- `num()`, 210

- `optimize_model` (`optimize_model.bgmfit`), 155
- `optimize_model()`, 6, 51, 59, 75, 92, 97, 111, 120, 138, 152, 158, 170, 182, 189, 196, 202, 213, 221
- `optimize_model.bgmfit`, 155
- `option`, 210
- `options`, 41, 43

- `p_direction()`, 129
- `parallel::makeCluster()`, 84, 145
- `parallel::splitIndices()`, 68, 83, 104, 144
- `pareto_k_ids`, 137
- `patchwork::wrap_plots()`, 163, 181, 189, 192
- `pd_to_p()`, 129
- `performance::r2_bayes()`, 95
- `performance::r2_loo()`, 95
- `plan`, 43
- `plot_caterpillar`
(`plot_caterpillar.bgmfit`), 159
- `plot_caterpillar.bgmfit`, 159
- `plot_conditional_effects`
(`plot_conditional_effects.bgmfit`), 165
- `plot_conditional_effects.bgmfit`, 165
- `plot_curves` (`plot_curves.bgmfit`), 173
- `plot_curves()`, 57, 67, 83, 103, 118, 144, 169, 177, 200
- `plot_curves.bgmfit`, 173
- `plot_diagnostics`
(`plot_diagnostics.bgmfit`), 186
- `plot_diagnostics.bgmfit`, 186
- `plot_ppc` (`plot_ppc.bgmfit`), 193
- `plot_ppc.bgmfit`, 193
- `posterior::as_draws_df()`, 214
- PPC, 194
- `predict_draws` (`predict_draws.bgmfit`), 197
- `predict_draws()`, 57, 67, 83, 98, 103, 116, 118, 121, 144, 167, 169, 177, 178, 184, 200
- `predict_draws.bgmfit`, 197
- `prior_conflict` (`prior_conflict.bgmfit`), 205
- `prior_conflict()`, 206, 210, 211, 215
- `prior_conflict.bgmfit`, 205
- `prior_sensitivity`
(`prior_sensitivity.bgmfit`), 207
- `prior_sensitivity()`, 205, 206
- `prior_sensitivity.bgmfit`, 207
- `prior_table` (`prior_table.bgmfit`), 216
- `prior_table.bgmfit`, 216
- `priorsense::log_lik_draws()`, 214
- `priorsense::log_prior_draws()`, 214
- `priorsense::powerscale_plot_dens()`, 210, 215
- `priorsense::powerscale_plot_ecdf()`, 210, 215
- `priorsense::powerscale_plot_quantities()`, 215
- `priorsense::powerscale_sensitivity()`, 209–211, 213–215
- `priorsense::predictions_as_draws()`, 210, 215

- `ranef.brmsfit`, 127
- `reloo`, 137
- `rope_range()`, 128
- `rstan::expose_stan_functions()`, 52–54

`rstan::rstan()`, 44
`rstan::stan_model`, 42
`rstan::stan_model()`, 53, 54
`rstan::stanc()`, 53, 54
`rstanarm::stan_polr()`, 189

`saveRDS`, 5, 43, 157
`seq()`, 57, 67, 82, 103, 117, 143, 169, 177, 200
`set.seed`, 127
`sitar::getPeak()`, 59, 71, 87, 119, 121, 147, 182, 201
`sitar::getTakeoff()`, 71, 87, 121, 147
`sitar::getTrough()`, 71, 87, 121, 147
`sitar::plot.sitar`, 180
`sitar::sitar()`, 44, 45
`sitar::velout()`, 40
`sitar::zapvelout()`, 40
`splines2::nsk()`, 18, 22, 44
`splines2::nsp()`, 18, 44
`splines::ns()`, 44
`stanvars`, 65, 80, 101, 142
`stats::density()`, 195
`stats::model.matrix()`, 19

`tidybayes::add_epred_draws()`, 188, 191, 192
`tidybayes::add_linpred_draws()`, 188, 191, 192
`tidybayes::add_predicted_draws()`, 188, 191, 192
`tidybayes::add_residual_draws()`, 191, 192
`tidybayes::median_qi()`, 191

`update`, 43
`update_model`, 158
`update_model(update_model.bgmfit)`, 220
`update_model.bgmfit`, 220

`variables`, 127

`yaxis_text()`, 195