Package: bsitar (via r-universe)

September 13, 2024

Type Package

Title Bayesian Super Imposition by Translation and Rotation Growth Curve Analysis

Version 0.2.2.01

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

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

License GPL-2

Depends R (>= 3.6)

Imports brms (>= 2.21.0), rstan (>= 2.32.6), loo (>= 2.7.0), dplyr (>=

1.1.3), rlang (>= 1.1.2), Rdpack (>= 2.6.1), insight (>= 0.20.3), data.table (>= 1.15.4), collapse (>= 2.0.15), marginaleffects (>= 0.21.0), sitar, magrittr, methods, utils

Suggests ggplot2 (>= 3.4.0), bayesplot (>= 1.11.0), posterior (>= 1.3.1), testthat (>= 3.0.0), dtplyr (>= 1.3.1), checkmate (>= 2.3.1), doParallel (>= 1.0.17), parallel (>= 4.3.1), foreach (>= 1.5.2), ggridges (>= 0.5.6), jtools (>= 2.2.2), tidyr, nlme, purrr, future, future.apply, forcats, patchwork, tibble, pracma, extraDistr, bookdown, knitr, kableExtra, rmarkdown, spelling, Hmisc, R.rsp, graphics, grDevices, ggtext, 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

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Language en-US

Repository https://sandhu-ss.r-universe.dev

RemoteUrl https://github.com/sandhu-ss/bsitar

RemoteRef HEAD

RemoteSha 431fc2ca958025737fe368cb16253070e14641a9

Contents

add_model_criterion.bgmfit 3
berkeley 6
berkeley_exdata 7
berkeley_exfit 8
bsitar 9
expose_model_functions.bgmfit 41
fitted_draws.bgmfit 43
getNsObject 49
growthparameters.bgmfit 50
growthparameters_comparison.bgmfit 57

loo_validation.bgmfit 69
marginal_comparison.bgmfit 73
marginal_draws.bgmfit 83
optimize_model.bgmfit 94
plot_conditional_effects.bgmfit 98
plot_curves.bgmfit 104
plot_ppc.bgmfit 114
predict_draws.bgmfit 118
update_model.bgmfit 124

Index **127**

`add_model_criterion.bgmfit`

Add model fit criteria to model

Description

The **add_model_criterion()** is a wrapper around the `brms::add_criterion()`. Note that arguments `compare` and `pointwise` are relevant only for brms::add_loo whereas arguments `summary`, `robust`, and `probs` 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_names = NULL,
  summary = TRUE,
  robust = FALSE,
  probs = c(0.025, 0.975),
  newdata = NULL,
  resp = NULL,
  cores = 1,
  deriv_model = NULL,
  verbose = FALSE,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  envir = NULL,
  ...
)

add_model_criterion(model, ...)
```

Arguments

model	An object of class bgmfit.
criterion	Names of model fit criteria to compute. Currently supported are "loo", "waic", "kfold", "loo_subsample", "bayes_R2" (Bayesian R-squared), "loo_R2" (LOO-adjusted R-squared), and "marglik" (log marginal likelihood).
ndraws	A positive integer indicating the number of posterior draws to be used in estimation. If NULL (default), all draws are used.
draw_ids	An integer indicating the specific posterior draw(s) to be used in estimation (default NULL).
compare	A flag indicating if the information criteria of the models should be compared to each other via loo_compare.
pointwise	A flag indicating whether to compute the full log-likelihood matrix at once or separately for each observation. The latter approach is usually considerably slower but requires much less working memory. Accordingly, if one runs into memory issues, pointwise = TRUE is the way to go.
model_names	If NULL (the default) will use model names derived from deparsing the call. Otherwise will use the passed values as model names.
summary	A logical indicating whether only the estimate should be computed (TRUE), or estimate along with SE and CI should be returned (FALSE, default). Setting summary as FALSE will increase the computation time. Note that summary = FALSE is must to get the correct estimates when re_formula = NULL.
robust	A logical to specify the summarize options. 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. 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 to be used in estimation. If NULL (default), the newdata is retrieved from the model.
resp	A character string (default NULL) to specify response variable when processing posterior draws for the univariate_by and multivariate models. See bsitar() for details on univariate_by and multivariate models
cores	Number of cores to be used when running the parallel computations (if future = TRUE). On non-Windows systems this argument can be set globally via the mc.cores option. For the default NULL option, the number of cores are set automatically by calling the future::availableCores(). The number of cores used are the maximum number of cores avaialble minus one, i.e., future::availableCores() - 1.
deriv_model	A logical to specify whether to estimate velocity curve from the derivative function, or the differentiation of the distance curve. The argument deriv_model is set to TRUE for those functions which need velocity curve such as growthparameters() and plot_curves(), and NULL for functions which explicitly use the distance curve (i.e., fitted values) such as loo_validation() and plot_ppc().

verbose — An optional argument (logical, default FALSE) to indicate whether to print information collected during setting up the object(s).

expose_function — An optional logical argument to indicate whether to expose Stan functions (default FALSE). Note that if user has already exposed Stan functions during model fit by setting expose_function = TRUE in the bsitar(), then those exposed functions are saved and can be used during post processing of the posterior draws and therefore expose_function is by default set as FALSE in all post processing functions except optimize_model(). For optimize_model(), the default setting is expose_function = NULL. The reason is that each optimized model has different Stan function and therefore it need to be re exposed and saved. The expose_function = NULL implies that the setting for expose_function is taken from the original model fit. Note that expose_function must be set to TRUE when adding fit criteria and/or bayes_R2 during model optimization.

usesavedfuns — A logical (default NULL) to indicate whether to use the already exposed and saved Stan functions. Depending on whether the user have exposed Stan functions within the bsitar() call via expose_functions argument in the bsitar(), the usesavedfuns is automatically set to TRUE (if expose_functions = TRUE) or FALSE (if expose_functions = FALSE). Therefore, manual setting of usesavedfuns as TRUE/FALSE is rarely needed. This is for internal purposes only and mainly used during the testing of the functions and therefore should not be used by users as it might lead to unreliable estimates.

clearenvfuns — A logical to indicate whether to clear the exposed function from the environment (TRUE) or not (FALSE). If NULL (default), then clearenvfuns is set as TRUE when usesavedfuns is TRUE, and FALSE if usesavedfuns is FALSE.

envir — Environment used for function evaluation. The default is NULL which will set parent.frame() as default environment. Note that since most of post processing functions are based on **brms**, the functions needed for evaluation should be in the .GlobalEnv. Therefore, it is strongly recommended to set envir = globalenv() (or envir = .GlobalEnv). This is particularly true for the derivatives such as velocity curve.

... — Further arguments passed to brms::fitted.brmsfit() and brms::predict() functions.

Value

An object of class class bgmfit with fit criteria added.

Author(s)

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

See Also

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

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'.

model <- berkeley_exfit

model <- add_model_criterion(model, criterion = c("waic"))
```

---

berkeley *Berkeley Child Guidance Study Data*

Description

Data provide longitudinal growth records for 136 children.

Usage

```
berkeley
```

Format

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

id factor with levels 201-278 for males, and 301-385 for females

age years, numeric vector

height cm, numeric vector

weight kg, numeric vector

stem.length cm, numeric vector

bi.acromial cm, numeric vector

bi.iliac cm, numeric vector

leg.circ cm, numeric vector

strength lb, numeric vector

sex factor with level 1 male and level 2 female

Details

Data 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), and later used as an example dataset in the **sitar** (Cole 2022) package after correcting for the transcription errors.

A detailed description of the data including the frequency of measurements per year is provided in the **sitar** package. (Cole 2022). Briefly, the data comprise of repeated growth measurements made on 66 boys and 70 girls (birth to 21 years). Children were born in 1928-29 (Berkeley, California) and were of north European ancestry. Measurements were made at the following ages: 0 (i.e, at birth), 0.085 year, 0.25 to 2 years (every 3 month), 2 to 8 years (annually), and 8 to 21 years (6-monthly). The children were measured for height, weight (undressed), stem length, biacromial diameter, bi-iliac diameter, leg circumference, and dynamo metric strength.

Value

A data frame with 10 columns.

Author(s)

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

References

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

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

---

berkeley_exdata *Berkeley Child Guidance Study Data for females*

Description

A subset of the berkeley data that contains longitudinal growth data for 70 females (8 to 18 years of age).

Usage

```
berkeley_exdata
```

Format

A data frame with following 3 variables:

id factor variable

age years, numeric vector

height cm, numeric vector

Details

A detailed description of the full data is provided 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 data (70 females, 8 to 18 years of age).

Usage

```
berkeley_exfit
```

Format

Model fit comprising summary of posterior draws.

Details

Data details are provided in the berkeley_exdata

Value

An object of class `bgmfit` with posterior draws.

Author(s)

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

bsitar *Fit Bayesian SITAR growth curve model*

Description

The **bsitar()** is the main function that fits the Bayesian version of the super imposition by translation and rotation (*SITAR*) model. The *SITAR* model is a nonlinear mixed effects model that has been used extensively to summarize growth processes (such as height and weight) from early childhood through the adulthood. The frequentist version of the *SITAR* model can be fit by using an already available R package, **sitar** (Cole 2022). Besides Bayesian implementation, the **bsitar** package greatly enhances the modelling capabilities of the *SITAR*. For example, in addition to the univariate analysis (i.e, modelling a single outcome), the **bsitar** allows univariate-by-subgroup and multivariate model fitting. In univariate-by-subgroup analysis, a single outcome is simultaneously analysed for subgroups defined by a factor variable such as gender. An advantage of univariate-by-subgroup analysis is that posterior draws for each sub group are part of a single model object that makes it possible to compare coefficients across groups and also test various hypotheses. The multivariate analysis involves simultaneous joint modelling of two or more outcomes.

Usage

```
bsitar(
  x,
  y,
  id,
  data,
  df = 4,
  knots = NA,
  fixed = a + b + c,
  random = a + b + c,
  xoffset = mean,
  bstart = xoffset,
  cstart = 0,
  xfun = NULL,
  yfun = NULL,
  bound = 0.04,
  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,
 sigmadf = 4,
 sigmaknots = NA,
 sigmafixed = a + b + c,
 sigmarandom = "",
 sigmaxoffset = mean,
 sigmabstart = sigmaxoffset,
 sigmacstart = 0,
 sigmaxfun = NULL,
 sigmabound = 0.04,
 dpar_formula = NULL,
 autocor_formula = NULL,
 family = gaussian(),
 custom_family = NULL,
 custom_stanvars = NULL,
group_arg = list(groupvar = NULL, by = NULL, cor = un, cov = NULL, dist = gaussian),
 sigma_group_arg = list(groupvar = NULL, by = NULL, cor = un, cov = NULL, dist =
   gaussian),
 univariate_by = list(by = NA, cor = un, terms = subset),
 multivariate = list(mvar = FALSE, cor = un, rescor = TRUE),
 a_prior_beta = normal(ymean, ysd, autoscale = TRUE),
 b_prior_beta = normal(0, 3.5, autoscale = FALSE),
 c_prior_beta = normal(0, 1.5, autoscale = FALSE),
 d_prior_beta = normal(0, 1, autoscale = TRUE),
 s_prior_beta = normal(lm, lm, autoscale = FALSE),
 a_cov_prior_beta = normal(0, 5, autoscale = FALSE),
 b_cov_prior_beta = normal(0, 1, autoscale = FALSE),
 c_cov_prior_beta = normal(0, 0.1, autoscale = FALSE),
 d_cov_prior_beta = normal(0, 1, autoscale = FALSE),
 s_cov_prior_beta = normal(lm, lm, autoscale = FALSE),
 a_prior_sd = normal(0, ysd, autoscale = TRUE),
 b_prior_sd = normal(0, 2, autoscale = FALSE),
 c_prior_sd = normal(0, 1.25, autoscale = FALSE),
 d_prior_sd = normal(0, 1, autoscale = TRUE),
 a_cov_prior_sd = normal(0, 5, autoscale = FALSE),
 b_cov_prior_sd = normal(0, 1, autoscale = FALSE),
 c_cov_prior_sd = normal(0, 0.1, autoscale = FALSE),
 d_cov_prior_sd = normal(0, 1, autoscale = FALSE),
 a_prior_sd_str = NULL,
 b_prior_sd_str = NULL,
 c_prior_sd_str = NULL,
```

```
d_prior_sd_str = NULL,
a_cov_prior_sd_str = NULL,
b_cov_prior_sd_str = NULL,
c_cov_prior_sd_str = NULL,
d_cov_prior_sd_str = NULL,
sigma_prior_beta = normal(0, 1, autoscale = FALSE),
sigma_cov_prior_beta = normal(0, 0.5, autoscale = FALSE),
sigma_prior_sd = normal(0, 0.25, autoscale = FALSE),
sigma_cov_prior_sd = normal(0, 0.15, autoscale = FALSE),
sigma_prior_sd_str = NULL,
sigma_cov_prior_sd_str = NULL,
rsd_prior_sigma = normal(0, ysd, autoscale = FALSE),
dpar_prior_sigma = normal(0, ysd, autoscale = TRUE),
dpar_cov_prior_sigma = normal(0, 1, autoscale = FALSE),
autocor_prior_acor = uniform(-1, 1, autoscale = FALSE),
autocor_prior_unstr_acor = lkj(1),
gr_prior_cor = lkj(1),
gr_prior_cor_str = lkj(1),
sigma_prior_cor = lkj(1),
sigma_prior_cor_str = lkj(1),
mvr_prior_rescor = lkj(1),
init = NULL,
init_r = NULL,
a_init_beta = random,
b_init_beta = random,
c_init_beta = random,
d_init_beta = random,
s_init_beta = random,
a_cov_init_beta = random,
b_cov_init_beta = random,
c_cov_init_beta = random,
d_cov_init_beta = random,
s_cov_init_beta = random,
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,
verbose = FALSE,
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 = 0.8, max_treedepth = 15),
empty = FALSE,
rename = TRUE,
pathfinder_args = NULL,
pathfinder_init = 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),
  parameterization = "ncp",
  ...
)
```

Arguments

x	Predictor variable (typically age in years). For `univariate` model, the `x` is a single variable whereas for `univariate_by` and `multivariate` models, the `x` can be same for sub models, or different for each sub model. For example, when fitting a bivariate model, the `x = list(x1, x2)` specifies that `x1` is the predictor variable for the first sub model, and `x2` for the second sub model. To specify `x1` as a common predictor variable for both sub models, the argument `x` is defined as `x = list(x1)` or simply `x = x1`.
y	Response variable (e.g., repeated height measurements). For `univariate` and `univariate_by` models, `y` is specified as a single variable. For `univariate_by` model, the response vector for each sub model is created and named internally based on the factor levels of the variable that is used to set up the `univariate_by` model. As an example, the model specified as `univariate_by = sex` creates response vectors `Female` and `Male` when `Female` is the first level and `Male` is the second level of the `sex` variable. For `multivariate` model, the response variables are specified as a list such as `y = list(y1, y2)` where `y1` is the response variable for the first sub model and `y2` for the second sub model. Note that for `multivariate` model, data are not stacked but rather response vectors are separate variables in the `data` and are of same length.
id	A factor variable uniquely identifying the groups (e.g., individuals) in the data frame. For `univariate_by` and `multivariate` models, the `id` can be same (typically) for sub models or different for each sub model (see argument `x` for details on setting different arguments for sub models).
data	Data frame containing variables such as `x`, `y`, `id` etc.
df	Degrees of freedom for the natural cubic spline design matrix (default `4`). The `df` is internally used to construct the knots (quantiles of `x` distribution) that are then used in the construction of the spline design matrix. For `univariate_by` and `multivariate` models, the `df` can be same (e.g., `df = 4`) for sub models or different for each sub model such as `df=list(4, 5)` where `df` is 4 is for the first sub model, and 5 for the second sub model.
knots	A numeric vector vector specifying the knots for the natural cubic spline design matrix (default `NULL`) Note that `df` and `knots` can not be specified together, and also both of them can not be `NULL`. In other words, either `df` or `knots` must be

specified. Like df, the knots can be same for sub models or different for each sub model when fitting univariate_by and multivariate models (see df for details).

fixed — A character string specifying the fixed effects structure (default 'a+b+c'). Note that different fixed effect structures can be specified when fitting univariate_by and multivariate models. As an example, fixed = list('a+b+c', 'a+b') implies that the fixed effect structure for the first sub model is 'a+b+c', and 'a+b' for the second sub model.

random — A character string specifying the random effects structure (default 'a+b+c'). The approach used in setting the random is same as described above for the fixed effects structure (see fixed).

xoffset — An optional character string, or a numeric value to set up the origin of the predictor variable, x (i.e., centering of x). The options available are 'mean' (mean of x, i.e., mean(x)), 'max' (maximum value of x, i.e., max(x)), 'min' (minimum value of x, i.e., min(x)), 'apv' (age at peak velocity estimated from the velocity curve derived from the simple linear model fit to the data), or any real number such as xoffset = 12. The default is xoffset = 'mean'. For univariate_by and multivariate models, the xoffset can be same for sub models or different for each sub model (see argument x for details on setting different arguments for sub models). Note that if xoffset is a numeric value, it will be internally transformed (log or sqrt) depending on the xfun argument. Similarly, when xoffset is 'mean', 'min' or 'max', then these value are retrieved after the log or sqrt transformation of the predictor variable, 'x'.

bstart — An optional character string, or a numeric value to set up the origin of the fixed effect parameter b. The argument bstart can be used to set up the location parameter for the location-scale based priors (such as normal()) via b_prior_beta argument and/or the initial value via the b_init_beta argument. The options available to set up the bstart are same as described above for the xoffset i.e., 'mean', 'min', 'max', 'apv' or a real number such as 12. The default is same as xoffset i.e., bstart = 'xoffset'. For univariate_by and multivariate models, the xoffset can be same for sub models (typically), or different for each sub model (see argument x for details on setting different arguments for sub models).

cstart — An optional character string, or a numeric value to set up the origin of the fixed effect parameter c. The argument cstart can be used to set up the location parameter for the location-scale based priors (such as normal()) via c_prior_beta argument and/or the initial value via the c_init_beta argument. The options available to set up the cstart are 'pv' (peak velocity estimated from the velocity curve derived from the simple linear model fit to the data), or a real number such as 1. Note that since parameter c is estimated on the exponential scale, the argument cstart should be adjusted accordingly. The default cstart is '0' i.e., cstart = '0'. For univariate_by and multivariate models, the xoffset can be same for sub models (typically), or different for each sub model (see argument x for details on setting different arguments for sub models).

xfun — An optional character string to specify the transformation of the predictor variable, The default is NULL indicating that no transformation is applied i.e., model

is fit to the data with original scale of the x. Available transformation options are 'log' (logarithmic transformation) and 'sqrt' (square root transformation). For univariate_by and multivariate models, the xfun can be same for sub models (typically), or different for each sub model (see argument x for details on setting different arguments for sub models).

yfun — An optional character string to specify the transformation of the response variable, The default is NULL, indicating that no transformation is applied i.e., model is fit to the data with original scale of the y. Available transformation options are 'log' (logarithmic transformation) and 'sqrt' (square root transformation). For univariate_by and multivariate models, the xfun can be same for sub models (typically), or different for each sub model (see argument x for details on setting different arguments for sub models).

bound — An optional real number to extend the span of the predictor variable x by a small value (default 0.04). See package sitar::sitar() for details. For univariate_by and multivariate models, the bound can be same for sub models (typically), or different for each sub model (see argument x for details on setting different arguments for sub models).

terms_rhs — An optional character string (default NULL) to specify terms on the right hand side of the response variable (separated by |) but before the formula tilde sign i.e., ~. The terms_rhs is used when fitting a measurement error model. As an example, consider fitting a model with measurement error in the response variable which is specified in the brms::brmsformula() as brmsformula(y | mi(sdy) ~ ..). In this example, the mi(sdy) is passed to brms::brmsformula() as terms_rhs = mi(sdy). For multivariate model, each outcome can have its own measurement error variable that can be specified as follows: terms_rhs = list(mi(sdy1), mi(sdy2)). Note that brms::brmsformula() does not allow combining mi() with the subset() formulation that is used for fitting univariate_by model.

a_formula — Formula for the fixed effect parameter, a (default ~ 1). User can specify different formula when fitting univariate_by and multivariate models. As an example a_formula = list(~1, ~1 + cov) implies that the a_formula for the first sub model includes an intercept only whereas the second sub model includes an intercept and a covariate, cov. The covariate(s) can be continuous variable(s) or factor variable(s). For factor covariates, dummy variables are created internally via the stats::model.matrix()). The formula can include any combination of continuous and factor variables as well as their interactions.

b_formula — Formula for the fixed effect parameter, b (default ~ 1). See a_formula for details.

c_formula — Formula for the fixed effect parameter, c (default ~ 1). See a_formula for details.

d_formula — Formula for the fixed effect parameter, d (default ~ 1). See a_formula for details.

s_formula — Formula for the fixed effect parameter, s (default ~ 1). The s_formula sets up the the spline design matrix. Typically, covariate(s) are not included in the s_formula to limit the population curve to be single curve for the whole data. In fact, the sitar::sitar() does not provide any option to include covariates

in the s_formula, However, **bsitar** package allows inclusion of covariates but the user need to justify the modelling of separate curves for each category when covariate is a factor variable.

a_formula_gr Formula for the random effect parameter, a (default ~ 1). Similar to a_formula, user can specify different formula when fitting univariate_by and multivariate models and formula can include continuous and/or factor variable(s) including their interactions as covariates (see a_formula for details). In addition to setting up the design matrix for the random effect parameter a, user can set up the group identifier and the correlation structure for random effects via the vertical bar || approach. For example, consider only an intercept for the random effects a, b, and c specified as a_formula_gr = ~1, b_formula_gr = ~1 and c_formula_gr = ~1. To specify the group identifier (e.g., id) and an unstructured correlation structure, the formula argument as specified as follows:
a_formula_gr = ~ (1|i|id)
b_formula_gr = ~ (1|i|id)
c_formula_gr = ~ (1|i|id)
where i within the vertical bars || is just a placeholder. A common identifier (i.e., i) shared across random effect formulas are modeled as unstructured correlated. For more details on the the vertical bar approach, please see brms::brm(). As explained below, an alternative approach to set up the group identifier and the correlation structure is to use group_by argument. In other words, to achieve the same set up as defined above by using the vertical bar approach, user can just specify the design matrix part of the formula as a_formula_gr = ~ 1
b_formula_gr = ~ 1
c_formula_gr = ~ 1
and use the group_by argument as group_by = list(groupvar = id, cor = un) where id specifies the group identifier and un sets up the unstructured correlation structure. See group_by argument for 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) when fitting a hierarchical model with three or more levels of hierarchy. An example is model applied to the data that comprise repeated measurements (level 1) on individuals (level 2) nested further within the growth studies (level 3). Note that When using a_formula_gr_str argument, only the vertical bar approach (see a_formula_gr) can be used to set up the group identifiers and the correlation structure. An example of setting up the formula for a three level model with random effect parameter a, b is as follows:
a_formula_gr_str = ~ (1|i|id:study) + (1|i2|study)
b_formula_gr_str = ~ (1|i|id:study) + (1|i2|study)
c_formula_gr_str = ~ (1|i|id:study) + (1|i2|study)
where |i| and |i2| set up the unstructured correlation structure for individual and study level random effects. Note that |i| and |i2| need to be distinct

because random effect parameters are not allowed to be correlated across different levels of hierarchy. It is worth mentioning that user can set up model with any number of hierarchical levels and include covariate into the random effect formula.

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

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

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

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

sigma_formula
: Formula for the fixed effect distributional parameter, sigma. The sigma_formula sets up the fixed effect design matrix that may include continuous and/or factor variables (and their interactions) as covariates(s) for the distributional parameter. In other words, setting up the covariates for sigma_formula is same as for any other fixed parameter such as a (see a_formula for details). Note that sigma_formula estimates sigma parameter at log scale. By default, the sigma_formula is NULL because the brms::brm() itself models the sigma as a residual standard deviation (RSD) parameter at the link scale. The sigma_formula along with the arguments sigma_formula_gr and sigma_formula_gr_str allow estimating the scale parameters as random effects for sigma. The set up to specify the fixed and random effects for sigma is similar to setting fixed and random effect structures for other model parameters such as a, b, and c. It is important to note that an alternative way to set up the fixed effect design matrix for the distributional parameter sigma is to use the dpar_formula argument. An advantage of dpar_formula over sigma_formula is that user can specify the linear and nonlinear formulation as allowed by the brms::lf() and brms::nlf() syntax. The brms::lf() and brms::nlf() offer flexibility in centering the predictors and also allows enabling/disabling of cell mean centering when excluding intercept via 0 + formulation. A disadvantage of dpar_formula approach is that it is not possible to include random effects for the sigma. Note that sigma_formula and dpar_formula can not be specified together. When either sigma_formula or dpar_formula is used, the default estimation of the RSD by brms::brm() is automatically turned off. Note that user can specify an external function such as poly but with only a single argument (predictor) i.e.

poly(age). The addition arguments must be specified externally. For example, if user wants to set degree as degree, then a copy of poly can be created which is then modified and used in the sigma_formula as mypoly = poly; formals(mypoly)[['degree']] <- 3; mypoly(age).

sigma_formula_gr
: Formula for the random effect parameter, sigma (default NULL). See a_formula_gr for details. Like sigma_formula, external function such as poly can used. Please above for details sigma_formula.

sigma_formula_gr_str
: Formula for the random effect parameter, sigma when fitting a hierarchical model with three or more levels of hierarchy. See a_formula_gr_str for details. Like sigma_formula, external function such as poly can used. Please above for details sigma_formula.

sigma_formula_manual
: Formula for the random effect parameter, sigma via a character string string that explicitly uses the brms::nlf() and brms::lf() functions (default NULL). An example is nlf(sigma ~ z) + lf(z ~ 1 + age + (1 + age |55| gr(id, by = NULL))). Another use case of sigma_formula_manual is to model location scale model for the SITAR model where same SITAR formula can be used to model the scale (sigma). An example is nlf(sigma ~ sigmaSITARFun(logage, 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) where sigmaSITARFun (and all other needed sub functions) are defined by the sigmax, sigmadf, sigmaknots, sigmafixed, sigmarandom, sigmaxoffset, sigmaxfun and sigmabound arguments. It is important to match the sigma_formula_manual code as sigmaSITARFun created by the above arguments.

  Note that for sigma_formula_manual, the priors need to set up manually via the add_self_priors argument. To see which all priors need to be added, the user can run the code by setting get_priors = TRUE. Also, no initial values are defined and therefore initials for these parameters can be either 0 or random.

sigmax
: Predictor for the sigma. See x for details. Ignored if sigma_formula_manual = NULL.

sigmadf
: Degree of freedom for the spline function for sigma. See df for details. Ignored if sigma_formula_manual = NULL.

sigmaknots
: Degree of freedom for the spline function for sigma. See knots for details. Ignored if sigma_formula_manual = NULL.

sigmafixed
: Fixed effect formula for the sigma structure. See fixed for details. Ignored if sigma_formula_manual = NULL.

sigmarandom
: Random effect formula for the sigma structure. See random for details. Ignored if sigma_formula_manual = NULL. Currently not used even when setting up the sigma_formula_manual.

sigmaxoffset
: Offset for the x for sigma structure. See xoffset for details. Ignored if sigma_formula_manual = NULL.

sigmabstart
: Start of b parameter for sigma structure. See bstart for details. Ignored if sigma_formula_manual = NULL. Currently not used even when setting up the sigma_formula_manual.

sigmacstart
: Start of c parameter for sigma structure. See cstart for details. Ignored if sigma_formula_manual = NULL. Currently not used even when setting up the sigma_formula_manual.

sigmaxfun
: Transformation of x for the sigma structure. See xfun for details. Ignored if sigma_formula_manual = NULL.

sigmabound
: Bounds for the x for sigma structure. See bound for details. Ignored if sigma_formula_manual = NULL.

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 are:

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

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

family
: Family distribution (default gaussian) and the link function (default identity). See brms::brm() for details on available distributions and link functions, and how to specify them. For univariate_by and multivariate models, the family can be same (e.g., family = gaussian()) for sub models 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 sub model. Please note that argument family is ignored when use specifies custom_family i.e., custom_family is not NULL.

custom_family
: Specify custom families (i.e. response distribution). Default NULL. Please see brms::custom_family() for details. It is important no note that user defined Stan functions must be expose by setting expose_functions = TRUE.

custom_stanvars
: Prepare and pass user-defined variables that need to be added to the Stan's program blocks (default NULL). This is primarily useful when defining custom_family. Please see brms::custom_family() for details on specifying stanvars. Note that custom_stanvars are passed directly without conducting any sanity checks.

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

  - The groupvar specifies the subject identifier. In case groupvar = NULL (default), the groupvar is automatically assigned based on the id argument.

- The dist specifies the distribution from which the random effects are drawn (default gaussian). As per the brms::brm() documentation, the gaussian distribution is the only available distribution (as of now).
- The by argument can be used to estimate separate variance covariance structure (i.e., standard deviation and correlation parameters) for random effect parameters (default NULL). If specified, variable used to set up the by argument must be a factor variable. For example, by = 'sex' implies that separate variance covariance structure are estimated for males and females.
- The cor is used to set up the covariance (i.e., correlation) structure for random effect parameters. The default covariance is unstructured (i.e, cor = un) for all three model settings, i.e., univariate, univariate_by and multivariate. The alternative correlation structure available for univariate and univariate_by models is diagonal. While the cor = un models the full unstructured variance covariance structure, the cor = diagonal estimates only the variance (i.e, standard deviation) parameters and the covariance (i.e., correlation) parameters are set to zero. For *multivariate* model, options include un, diagonal and un_s. The un sets up the unstructured correlation implying that the group level random effects across response variables are drawn for a joint multivariate normal distribution with shared correlation parameters. The cor = diagonal specifies that only the variance parameter are estimates for each sub model whereas the correlation parameters set to zero. Option cor = un_s allows for estimating unstructured variance covariance parameters separately for each response variable.

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

sigma_group_arg

Specify arguments for modelling distributional level random effects, sigma. The approach used in setting up the sigma_group_arg is exactly same as described above for the group level random effects (see group_arg for details).

univariate_by Set up the univariate-by-subgroup model fitting (default NULL) via a named list as described below:

- The by (an optional character string) is used to specify the variable (must be a factor variable) to define the sub models (default NA).
- The cor (an optional character string) specifies the correlation structure. The options available are un and diagonal. The un = un (default) models the full unstructured variance covariance structure, whereas the cor = diagonal estimates only the variance (i.e, standard deviation) parameters and the covariance (i.e., correlation) parameters are set to zero.
- The terms (an optional character string) specifies the method used in setting up the sub models. Options are 'subset' (default) and 'weights'. See

brms::`addition-terms` for details.

multivariate Set up the multivariate model fitting (default NULL) arguments as a named list:

- The mvar (logical, default FALSE) indicates whether to fit a multivariate model.
- The cor (an optional character string) sets up the correlation structure. The options available are un, diagonal and un_s. The un sets up the unstructured correlation implying that the group level random effects across response variables are drawn for a joint multivariate normal distribution with shared correlation parameters. The cor = diagonal specifies that only the variance parameter are estimates for each sub model whereas the correlation parameters set to zero. Option cor = un_s allows for estimating unstructured variance covariance parameters separately for each response variable.
- The rescor (logical, default TRUE) indicates whether to estimate the residual correlation between response variables.

a_prior_beta Specify priors for the fixed effect parameter, a. (default normal(ymean, ysd, autoscale = TRUE)). The key points in prior specification that are applicable for all parameters are highlighted below. For full details on prior specification, please see brms::prior().

- Allowed distributions are normal, student_t, cauchy, lognormal, uniform, exponential, gamma and inv_gamma (inverse gamma).
- For each distribution, upper and lower bounds can be set via options lb and ub (default NA for both lb and ub).
- For location-scale based distributions (such as normal, student_t, cauchy, and lognormal), an option autoscale (default FALSE) can be used to multiply the scale parameter by a numeric value. Both **brms** and **rstanarm** packages allow similar auto scaling under the hood. While **rstanarm** earlier used to set autoscale as TRUE which internally multiplied scale parameter by a value 2.5 (recently authors changed this behavior to FALSE), the **brms** package sets scaling factor as 1.0 or 2.5 depending on the standard deviation of the response variable (See brms::prior()). The **bsitar** package offers full flexibility in choosing the scaling factor as any real number instead of 1.0 or 2.5 (e.g., autoscale = 5.0). When autoscale = TRUE, 2.5 is the default scaling factor.
- For location-scale based distributions such as normal, options fxl (function location) and fxs (function scale) are available to apply any function such as log and sqrt, or a function defined in the R environment to transform the location and scale parameters. For example, prior normal(2, 5, fxl = 'log', fxs = 'sqrt') will be translated internally as normal(log(2), sqrt(5)) implying that the actually prior assigned will be normal(0.693, 2.23). The default for both fxl and fxs is NULL.
- Like fxl and fxs functions, another function fxls (function location scale) is available to transform location and scale parameters for the location-scale based distributions such as normal. Unlike fxl and fxs functions which transform location and scale parameters individually, the fxls function is used for those transformation for which both location and scale parameters are needed in the transformation of these parameters. For example, the transformation of location and scale parameters for the normal prior on

log scale is as follows:
log_location = log(location / sqrt(scale^2 / location^2 + 1)),
log_scale = sqrt(log(scale^2 / location^2 + 1)),
where location and scale are the original parameters supplied by the user and log_location and log_scale are the equivalent parameters on the log scale. The fxls can be set as a character string or a list comprised of two functions where first function of the list will be used to transform the location parameter and the second function will be for the scale transformation. If a character string is used such as fxls = 'log', then the above transformation for the log parametrization will be applied automatically. Note that if using a list, then the list must be crated within the R environment and then passed this to the fxls as:
location_fun <- function(location, scale) { log(location / sqrt(scale^2 / location^2 + 1)) }
scale_fun <- function(location, scale) { sqrt(log(scale^2 / location^2 + 1)) }
fxls_fun <- list(location_fun = location_fun, scale_fun = scale_fun)
fxls = 'fxls_fun'
As an example, normal(2, 5, fxls = 'fxls_fun'. The default for fxls is NULL.

- For strictly positive distributions such as exponential, gamma and inv_gamma, the lower bound (lb) is automatically set to zero i.e., lb = 0.
- For uniform distribution, an option addrange is available to symmetrically widen the prior range. For example, prior uniform(a, b, addrange = 5) implies that the lower and upper limits will be evaluated as uniform(a-5, b+5).
- For exponential distribution, the rate parameter is evaluated as inverse. In other words, prior set as exponential(10.0) is translated to 0.1 i.e., exponential(1.0/10.0).
- User need not to specify each option explicitly because the missing options are set to their default values automatically. For example, the prior specified as a_prior_beta = normal(location = 5, scale = 1, lb = NA, ub = NA, addrange = NA, autoscale = FALSE, fxl = NULL, fxs = NULL)) is same as a_prior_beta = normal(5, 1)).
- For univariate_by multivariate models, priors can be same for sub models (e.g., a_prior_beta = normal(5, 1)), or different for each sub such as a_prior_beta = list(normal(5,1), normal(10, 5)).

The location parameter for the location-scale based distributions can be specified as mean (by specifying 'ymean') or the median (by using 'ymedian') of the response variable. Similarly, the scale parameter can be set as the standard deviation (SD) or the median absolute deviation (MAD) of the response variable via 'ysd' and 'ymad' options. Another option available is to use the coefficients 'lm' from the simple linear model applied to the data (e.g., lm(y ~ age, data = data). This is true even when model has covariates i.e., lm(y ~ age + cov, data = data). A few examples of specifying priors using these options are:
a_prior_beta = normal(ymean, ysd),
a_prior_beta = normal(ymean, ysd),
a_prior_beta = normal(ymedian, ymad),

a_prior_beta = normal(lm, ysd),
Note that options 'ymean', 'ymedian', 'ysd', 'ymad', 'ymad' and 'lm' are available only for the fixed effect parameter, a and not for parameters b, c or d.

b_prior_beta Specify priors for the fixed effect parameter, b. (default normal(0, 3.5, autoscale = FALSE)). See a_prior_beta for details.

c_prior_beta Specify priors for the fixed effect parameter, c. (default normal(0, 1.5, autoscale = FALSE)). See a_prior_beta for details.

d_prior_beta Specify priors for the fixed effect parameter, d. (default normal(0, 1.0, autoscale = FALSE)). See a_prior_beta for details.

s_prior_beta Specify priors for the fixed effect parameter, s (i.e., spline coefficients). (default normal(0, 'lm', autoscale = TRUE)). The general approach is same as described earlier for the fixed effect parameters (see a_prior_beta for details). A few key points are highlighted below:

- When specifying location-scale based priors using 'lm' such as s_prior_beta = normal(lm, 'lm') , it sets spline coefficients obtained from the simple linear model fit as location parameter whereas scale parameter is based on the standard deviation of the spline design matrix. However, typically, the location parameter is set at '0' (default), and the autoscale option is set as TRUE.
- For location-scale based priors, an option sethp (logical, default FALSE) is available to set up the hierarchical priors. To set sethp as TRUE, the prior is specified as s_prior_beta = normal(0, 'lm', autoscale = TRUE, sethp = TRUE)). When sethp = TRUE, instead of setting prior as s ~ normal(0, 'lm') the hierarchical priors are set as s ~ normal(0, 'hp') where 'hp' is defined as hp ~ normal(0, 'lm'). Note that the scale parameter for the hp ~ normal(0, 'lm') is automatically taken from the s ~ normal(0, 'hp'). Setting sethp = TRUE implies that the scale for spline coefficients is estimated from the data itself. The distribution of hierarchical priors is automatically matched with the prior set for the s parameter, or else can be set by the same sethp option. For example, s_prior_beta = normal(0, 'lm', sethp = cauchy) will be translated to s ~ normal(0, 'lm'), hp ~ cauchy(0, 'lm').
- For uniform priors, the optionaddrange can be used to symmetrically expand the prior range.

It is observed that location scale based prior distributions (e.g, normal, student_t, and cauchy) perform well for the spline coefficients.

a_cov_prior_beta
Specify priors for the covariate(s) included in the fixed effect parameter, a (default normal(0, 5.0, autoscale = FALSE)). The approach is same as described earlier for the a_prior_beta except that options 'ymean', 'ymedian', 'ysd', and 'ymad' are not allowed. The Option 'lm' for the location parameter sets covariate(s) coefficient obtained from the simple linear model fit to the data. Note that option 'lm' is allowed only for the a_cov_prior_beta and not for the covariate(s) included in the other fixed or random effect parameters. Lastly, separate priors can be specified for sub models when fitting univariate_by and a_prior_beta models (see a_prior_beta).

b_cov_prior_beta
: Specify priors for the covariate(s) included in the fixed effect parameter, b (default normal(0, 1.0, autoscale = FALSE)). See a_cov_prior_beta for details.

c_cov_prior_beta
: Specify priors for the covariate(s) included in the fixed effect parameter, c (default normal(0, 0.1, autoscale = FALSE)). See a_cov_prior_beta for details.

d_cov_prior_beta
: Specify priors for the covariate(s) included in the fixed effect parameter, d (default normal(0, 1.0, autoscale = FALSE)). See a_cov_prior_beta for details.

s_cov_prior_beta
: Specify priors for the covariate(s) included in the fixed effect parameter, s (default normal(0, 10.0, autoscale = FALSE)). However, as described earlier, (see s_formual), the *SITAR* model does not allows for inclusion of covariate(s) in the spline design matrix. If and when covariate(s) are specified (see s_formual), the approach of setting priors for the covariate(s) included in the parameter, s via s_cov_prior_beta is same as described earlier for the fixed effect parameter a (see a_cov_prior_beta). For the location-scale based priors, the option 'lm' sets the location parameter same as the spline coefficients obtained from fitting a simple linear to the data.

a_prior_sd
: Specify priors for the random effect parameter, a. (default normal(0, 'ysd', autoscale = FALSE)). Note that prior is on the standard deviation (which is the square root of the variance) and not on the variance itself. The approach of setting the prior is same as described earlier for the fixed effect parameter, a (See a_prior_beta) with the exception that location parameter is always zero. The lower bound 0 is automatically set by the brms::brm(). For univariate_by and multivariate models, priors can be same for sub models or different for each sub model (See a_prior_beta).

b_prior_sd
: Specify priors for the random effect parameter, b (default normal(0, 2.0, autoscale = FALSE)). See a_prior_sd for details.

c_prior_sd
: Specify priors for the random effect parameter, c (default normal(0, 1.25, autoscale = FALSE)). See a_prior_sd for details.

d_prior_sd
: Specify priors for the random effect parameter, d (default normal(0, 1.0, autoscale = FALSE)). See a_prior_sd for details.

a_cov_prior_sd
: Specify priors for the covariate(s) included in the random effect parameter, a (default normal(0, 5.0, autoscale = FALSE)). The approach is same as described earlier for the a_cov_prior_beta except that no pre-defined option (e.g., 'lm') is allowed.

b_cov_prior_sd
: Specify priors for the covariate(s) included in the random effect parameter, b (default normal(0, 1.0, autoscale = FALSE)). See a_cov_prior_sd for details.

c_cov_prior_sd
: Specify priors for the covariate(s) included in the random effect parameter, c (default normal(0, 0.1, autoscale = FALSE)). See a_cov_prior_sd for details.

d_cov_prior_sd Specify priors for the covariate(s) included in the random effect parameter, d (default normal(0, 1.0, autoscale = FALSE)). See a_cov_prior_sd for details.

a_prior_sd_str Specify priors for the random effect parameter, a when fitting a hierarchical model with three or more levels of hierarchy (default NULL). The approach is same as described earlier (see the a_prior_sd).

b_prior_sd_str Specify priors for the random effect parameter, b when fitting a hierarchical model with three or more levels of hierarchy (default NULL). The approach is same as described earlier (see the a_prior_sd_str).

c_prior_sd_str Specify priors for the random effect parameter, c when fitting a hierarchical model with three or more levels of hierarchy (default NULL). The approach is same as described earlier (see the a_prior_sd_str).

d_prior_sd_str Specify priors for the random effect parameter, d when fitting a hierarchical model with three or more levels of hierarchy (default NULL). The approach is same as described earlier (see the a_prior_sd_str).

a_cov_prior_sd_str
Specify priors for the covariate(s) included in the random effect parameter, a when fitting a hierarchical model with three or more levels of hierarchy (default NULL). The approach is same as described earlier (see the a_cov_prior_sd).

b_cov_prior_sd_str
Specify priors for the covariate(s) included in the random effect parameter, b when fitting a hierarchical model with three or more levels of hierarchy (default NULL). The approach is same as described earlier (see the a_cov_prior_sd_str).

c_cov_prior_sd_str
Specify priors for the covariate(s) included in the random effect parameter, c when fitting a hierarchical model with three or more levels of hierarchy (default NULL). The approach is same as described earlier (see the a_cov_prior_sd_str).

d_cov_prior_sd_str
Specify priors for the covariate(s) included in the random effect parameter, d when fitting a hierarchical model with three or more levels of hierarchy (default NULL). The approach is same as described earlier (see the a_cov_prior_sd_str).

sigma_prior_beta
Specify priors for the fixed effect distributional parameter, sigma (default normal(0, 1.0, autoscale = FALSE)). The approach is same as described earlier for the fixed effect parameter, a (See a_prior_beta for details).

sigma_cov_prior_beta
Specify priors for the covariate(s) included in the fixed effect distributional parameter, sigma (default normal(0, 0.5, autoscale = FALSE)). The approach is same as described earlier for the covariate(s) included the fixed effect parameter, a (see a_cov_prior_beta for details).

sigma_prior_sd Specify priors for the random effect distributional parameter, sigma (default normal(0, 0.25, autoscale = FALSE)). The approach is same as described earlier the random effect parameter a (see a_prior_sd for details).

sigma_cov_prior_sd
Specify priors for the covariate(s) included in the random effect distributional parameter, sigma (default normal(0, 0.15, autoscale = FALSE)). The approach

is same as described earlier for the covariate(s) included in the random effect parameter a (see a_cov_prior_sd for details).

sigma_prior_sd_str
: Specify priors for the the random effect distributional parameter, sigma when fitting a hierarchical model with three or more levels of hierarchy (default NULL). The approach is same as described earlier for the random effect parameter, a (See a_prior_sd_str for details).

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

rsd_prior_sigma
: Specify priors for the residual standard deviation parameter sigma (default normal(0, 'ysd', autoscale = TRUE)). Note that this argument is evaluated only when both dpar_formula and sigma_formula are NULL. For location scale based distributions, user can use specify standard deviation (ysd) or the median absolute deviation (ymad) as scale parameter.

dpar_prior_sigma
: Specify priors for the fixed effect distributional parameter sigma (default normal(0, 'ysd', autoscale = TRUE)). The argument is evaluated only when sigma_formula is NULL.

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

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

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

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

gr_prior_cor_str
: Specify priors for the correlation parameter(s) of group-level random effects when fitting a hierarchical model with three or more levels of hierarchy (default lkj(1)). The approach is same as described above (See gr_prior_cor).

sigma_prior_cor
: Specify priors for the correlation parameter(s) of distributional random effects sigma (default lkj(1)). The only allowed distribution is lkj (see gr_prior_cor for details). Note that currently brms::brm() does not allow for setting different lkj priors for the group level and distributional random effects that share the same group identifier (id). Therefore, either create a copy of group identifier and use that but then this will not allow correlation parameter across group random effects and sigma.

sigma_prior_cor_str
: Specify priors for the correlation parameter(s) of distributional random effects sigma when fitting a hierarchical model with three or more levels of hierarchy (default lkj(1)). The approach is same as described above (See sigma_prior_cor).

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

init
: Initial values for the sampler. If init = '0', all parameters are initialized to zero. For init = 'random', **Stan** will randomly generate initial values for each parameter within a range specified by the init_r (see below), or between -2 and 2 in unconstrained space when init_r = NULL. Another available option is init = 'prior' which sets initial values based on the prior specified for each parameter. Lastly, when init = NULL (default), initial value for each parameter is specified by the corresponding init arguments defined see below.

init_r
: A positive real value to set range for the random generation of initial values (default NULL). This argument is evaluated only when init = 'random'.

a_init_beta
: Initial values for the fixed effect parameter, a (default 'random'). Options available are '0', 'random' and 'prior'. In addition, user can specify 'ymean' and 'ymedian' to set initial as the mean or the median of the response variable. Also, option 'lm' can be used to set coefficients obtained from the simple linear model fitted to the data as initial values for the fixed effect parameter, a. Note that this is similar to the location parameter for prior on the fixed effect parameter a (see a_prior_beta for details). These options ('ymean', 'ymedian', and 'lm') are available only for the fixed effect parameter a and not for other parameters described below. Lastly, For univariate_by and multivariate models, the initials can be same (e.g., a_init_beta = 0) for sub models or different for each sub model such as
list(a_init_beta = '0', a_init_beta = 'lm').

b_init_beta
: Initial values for the fixed effect parameter, b (default 'random'). See a_init_beta for details.

c_init_beta
: Initial values for the fixed effect parameter, c (default 'random'). See a_init_beta for details.

d_init_beta
: Initial values for the fixed effect parameter, d (default 'random'). See a_init_beta for details.

s_init_beta
: Initial values for the fixed effect parameter, s (default 'random'). Options available are '0', 'random', 'prior', and 'lm'.

a_cov_init_beta

Initial values for the covariate(s) included in the fixed effect parameter, a (default 'random'). Options available are '0', 'random', 'prior' and 'lm'. The option 'lm' is available only for the a_cov_init_beta and not for the covariate(s) included in other fixed effect parameters b, c, or d.

b_cov_init_beta

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

c_cov_init_beta

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

d_cov_init_beta

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

s_cov_init_beta

Initial values for covariate(s) included in the fixed effect parameter, s (default 'lm'). See a_cov_init_beta for details. The option 'lm' will set the spline coefficients obtained from the simple linear model fitted to the data. Note that s_cov_init_beta is only a placeholder and is not valuated because covariate(s) are not allowed for the s parameter. See s_formula for details.

a_init_sd

Initial value for the standard deviation of group level random effect parameter, a (default 'random'). Options available are 'random', 'random' and 'prior'. In addition, 'ysd', 'ymad', 'lme_sd_a', and 'lm_sd_a' can be used to specify initial values as described below:

- The 'ysd' sets standard deviation (sd) of the response variable as an initial value.
- The 'ymad' sets median absolute deviation (mad) of the response variable as an initial value.
- The 'lme_sd_a' sets initial value based on the standard deviation of random Intercept obtained from the linear mixed model (nlme::lme()) fitted to the data. Note that in case nlme::lme() fails to converge, the option 'lm_sd_a' (see below) is set automatically.
- The 'lm_sd_a' sets square root of the residual variance obtained from the simple linear model applied to the data as an initial value.

Note that these option described above ('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. Lastly, when fitting univariate_by and multivariate models, user can set same initials for sub models, or different for each sub model.

b_init_sd

Initial value for the standard deviation of group level random effect parameter, b (default 'random'). See a_init_sd for details.

c_init_sd

Initial values for the group level random effect parameter, c (default 'random'). See a_init_sd for details.

d_init_sd

Initial value for the standard deviation of group level random effect parameter, d (default 'random'). See a_init_sd for details.

a_cov_init_sd

Initial values for the covariate(s) included in the random effect parameter, a (default 'random'). Options available are 'random', 'random' and 'prior'.

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

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

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

sigma_init_beta
Initial values for the fixed effect distributional parameter, sigma (default 'random'). Options available are 'random', 'random' and 'prior'.

sigma_cov_init_beta
Initial values for the covariate(s) included in the fixed effect distributional parameter, sigma (default 'random')

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

sigma_cov_init_sd
Initial values for the covariate(s) included in the distributional random effect parameter, sigma (default 'random'). (See a_cov_init_sd for details).

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

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

rsd_init_sigma Initial values for the residual standard deviation parameter, sigma (default 'random'). Options available are '0', 'random' and 'prior'. In addition, options 'lme_rsd' and 'lm_rsd' can be used as follows. The lme_rsd sets initial value based on the standard deviation of residuals obtained from the linear mixed model (nlme::lme()) fitted to the data. The initial value set by the 'lm_rsd' is the square root of the residual variance from the simple linear model applied to the data. Note that in case nlme::lme() fails to converge, then option 'lm_sd_a' is set automatically. The argument rsd_init_sigma is evaluated when dpar_formula and sigma_formula are set to NULL.

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

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

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

autocor_init_unstr_acor
Initial values for unstructured residual autocorrelation parameters (default 'random'). Allowed options are '0', 'random', and 'prior'. Note that the approach to set initials for autocor_init_unstr_acor is identical to the gr_init_cor.

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

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

vcov_init_0
: A logical (default FALSE) to set initials for variance (i.e, standard deviation) and covariance (i.e., correlation) parameters as zero. This allows for setting custom initials for the fixed effects parameters but zero initials for the variance covariance parameters.

jitter_init_beta
: A value as proportion (between 0 and 1) to perturb the initial values for fixed effect parameters. The default is NULL indicating that same initials are used across all chains. A sensible option can be jitter_init_beta = 0.1 as it mildly perturb the initials. Note that jitter is not absolute but proportion of the specified initial value. For example, if initial value is 100, then jitter_init_beta = 0.1 implies that the perturbed initial value will be within 90 and 110. On the other hand, if initial values is 10, then the perturbed initial value will be within 9 and 11.

jitter_init_sd
: A value as proportion (between 0 and 1) to perturb the initials for standard deviation of random effect parameters. The default is NULL indicating that same initials are used across all chains. An option of setting jitter_init_beta = 0.01 looked good during early testing.

jitter_init_cor
: A value as proportion (between 0 and 1) to perturb the initials for correlation parameters of random effects. The default is NULL indicating that same initials are used across all chains. An option of setting jitter_init_beta = 0.001 looked good during early testing.

prior_data
: An optional argument (a named list, default NULL) that can be used to pass information to the prior arguments for each parameter (e.g., a_prior_beta). The prior_data is particularly helpful in passing a long vector or a matrix as priors. These vectors and matrices can be created in the R framework and then passed using the prior_data. For example, to pass a vector of location and scale parameters when setting priors for covariate coefficients (with 10 dummy variables) included in the fixed effects parameter a, the following steps can be used to set covariate priors that each has scale parameter (sd) as 5 but mean values are drawn from a normal distribution with mean = 0 and sd = 1:

  - create the named objects prior_a_cov_location and prior_a_cov_scale in the R environment as follows:
    prior_a_cov_location <- rnorm(n = 10, mean = 0, sd = 1)
    prior_a_cov_scale <- rep(5, 10)

  - specify the above created objects prior_a_cov_location and prior_a_cov_scale in the prior_data as follows:
    prior_data = list(prior_a_cov_location = prior_a_cov_location, prior_a_cov_scale = prior_a_cov_scale).

- now use the prior_data objects to set up the priors as: a_cov_prior_beta = normal(prior_a_cov_location, prior_a_cov_scale).

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

init_custom Specify a custom initials object (a named list). The named list is directly passed to the init argument without checking for the dimensions and name matching. Note that in case initials are set for some parameter by using parameter specific argument (e.g., a_init_beta = 0), then init_custom is only passed to those parameters for which initials are missing. If user want to override this behaviors i.e., to pass all init_custom ignoring parameter specific initials, then init should be set as init = 'custom'.

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.

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

get_stancode An optional argument (logical, default FALSE) to get the stancode (see brms::stancode() for details).

get_standata An optional argument (logical, default FALSE) to get the standata (see brms::standata() for details).

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

get_stanvars An optional argument (logical, default FALSE) to get the stanvars (see brms::stanvar() for details).

get_priors An optional argument (logical, default FALSE) to get the priors. (see brms::get_prior() for details).

get_priors_eval An optional argument (logical, default FALSE) to get the priors specified by the user.

get_init_eval An optional argument (logical, default FALSE) to get the initial values specified by the user.

validate_priors An optional argument (logical, default FALSE) to validate the specified priors. (see brms::validate_prior() for details).

set_self_priors An optional argument (default NULL) to manually specify the priors. Note that set_self_priors is passed directly to the brms::brm() without performing any checks.

add_self_priors An optional argument (default NULL) to append part of prior object. This is for internal use only.

set_replace_priors
: An optional argument (default NULL) to replace part of prior object. This is for internal use only.

set_same_priors_hierarchy
: An optional argument (default NULL) to replace part of the prior object. This is for internal use only.

outliers
: An optional argument (default NULL) to remove outliers. The argument should be a named list which is passed directly to the sitar::velout() and sitar::zapvelout() functions. This is for internal use only.

unused
: An optional formula that defines variables that are unused in the model but should still be stored in the model's data frame. This can be useful when variables are required during the post-processing.

chains
: Number of Markov chains (default 4).

iter
: Number of total iterations per chain, including warmup (default 2000)

warmup
: A positive integer specifying the number of warmup (aka burnin) iterations. This also specifies the number of iterations used for stepsize adaptation, so warmup draws should not be used for inference. The number of warmup should not be larger than iter and the default is iter/2.

thin
: A positive integer. Set thin > 1 to save memory and computation time if iter is large. The thin > 1 is often used in cases with high autocorrelation of MCMC draws An indication of high autocorrelation is poor mixing of chain (i.e., high rhat values) despite the fact that model recovers the parameters well. An easy diagnostic to check for autocorrelation of MCMC draws is to use the mcmc_acf function from the **bayesplot**.

cores
: Number of cores to be used when executing the chains in parallel. See brms::brm() for details. Note that unlike brms::brm(), which sets default cores argument as cores=getOption("mc.cores", 1), the default cores in **bsitar** package is cores=getOption("mc.cores", 'optimize') which optimizes the utilization of system resources. The maximum number of cores that can be deployed is calculated as the maximum number of available cores minus 1. When the number of available cores is greater than the number of chains (see chains), then number of cores is set equal to the number of chains. Another option is to set cores as getOption("mc.cores", 'maximise') which sets the number of cores as the maximum number of cores available from the system regardless of the number of chains specified. Note that the user can also set cores argument similar to the brms::brm() i.e., getOption("mc.cores", 1). All these three options can be set globally as options(mc.cores = x) where x can be 'optimize', 'maximise' or 1. Lastly, the cores can set by directly by specifying an integer e.g., cores = 4.

backend
: A character string naming the package to be used when executing the the Stan model. Options are "rstan" (the default) or "cmdstanr". Can be set globally for the current R session via the "brms.backend". See brms::brm() for details.

threads
: Number of threads to be used in within-chain parallelization. Note that unlike the brms::brm() which sets the threads argument as getOption("brms.threads", NULL) implying that no within-chain parallelization is used by default, the **bsitar** package, by default, sets threads as getOption("brms.threads", 'optimize')

to utilize the available resources from the modern computing systems. The number of threads per chain is set as the maximum number of cores available minus 1. Another option is to set `threads` as `getOption("brms.threads", 'maximise')` which set the number threads per chains same as the maximum number of cores available. User can also set the `threads` similar to the `brms` i.e., `getOption("brms.threads", NULL)`. All these three options can be set globally as `options(brms.threads = x)` where x can be `'optimize'`, `'maximise'` or `NULL`. Alternatively, the number of threads can be set directly as `threads = threading(x)` where `X` is an integer. Other arguments that can be passed to the `threads` are `grainsize` and the `static`. See `brms::brm()` for further details on within-chain parallelization.

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

`normalize` Indicates whether normalization constants should be included in the Stan code (default `TRUE`). Setting it to `FALSE` requires Stan version >= 2.25. If `FALSE`, sampling efficiency may be increased but some post processing functions such as `brms::bridge_sampler()` will not be available. This option can be controlled globally via the `brms.normalize` option.

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

`control` A named `list` to control the sampler's behavior. The default are same as `brms::brm()` with the exception that the `max_treedepth` has been increased form 10 to 12 to allow better exploration of typically challenging posterior geometry posed by the nonlinear model. However, another control parameter, the `adpat_delta` which is also often need to be increased for nonlinear model, has be set to default setting as in `brms::brm()` i.e, 0.8. This is to avoid unnecessarily increasing the sampling time. See `brms::brm()` for full details on control parameters and their default values.

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

`rename` For internal use only.

`pathfinder_args`
A named `list` of arguments passed to the `'pathfinder'` algorithm. This is used to set `'pathfinder'` based initial values for the `'MCMC'`. Note that `'pathfinder_args'` currently works only for `backend = "cmdstanr"`. Therefore, even when user specified `backend = "rstan"`, it will be automatically changed to `backend = "cmdstanr"` when `'pathfinder_args'` is not `NULL`.

`pathfinder_init`
A logical `default FALSE` to indicate whether to use initials from the `'pathfinder'`

when fitting the final model i.e, 'MCMC' sampling. Note that 'pathfinder_args' currently works only for backend = "cmdstanr". Therefore, even when user specified backend = "rstan", it will be automatically changed to backend = "cmdstanr" when 'pathfinder_args' is not NULL. The arguments passed to the 'pathfinder' algorithm are specified via the 'pathfinder_args'. If 'pathfinder_args' = NULL, then the default arguments set via the 'cmdstanr' are used.

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

save_pars
: An object generated by brms::save_pars() controlling which parameters should be saved in the model. The argument has no impact on the model fitting itself.

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

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

refresh
: An integer to set the printing of every nth iteration. Default NULL indicates that refresh will be set automatically by the brms::brm(). Setting refresh is useful especially when thin is greater than 1. In that case, the refresh is recalculated as (refresh * thin) / thin.

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

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

save_model
: A character string or NULL (default). If not NULL, then the model's Stan code is saved via in a text file named after the string supplied in save_model.

fit
: An instance of S3 class brmsfit derived from a previous fit; defaults to NA. If fit is of class brmsfit, the compiled model associated with the fitted result is re-used and all arguments modifying the model code or data are ignored. It is not recommended to use this argument directly, but to call the update method, instead.

file
: Either NULL or a character string. In the latter case, the fitted model object is saved via saveRDS in a file named after the string supplied in file. The .rds extension is added automatically. If the file already exists, brm will load and return the saved model object instead of refitting the model. Unless you specify

the file_refit argument as well, the existing files won't be overwritten, you have to manually remove the file in order to refit and save the model under an existing file name. The file name is stored in the brmsfit object for later usage.

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

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

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

parameterization
A character string to specify Non-centered parameterization, NCP ('ncp') or the Centered parameterization, CP ('cp') to draw group level random effect. The NCP is generally recommended when likelihood is not strong (e.g., a few number of observations per individual). The NCP is the default (and only) approach implemented in the brms::brm(). The CP parameterization, on the other hand, is often considered more efficient than NCP when a relatively large number of observations are available across individual. The 'relatively large number' is not defined in the literature and we follow a general approach wherein CP parameterization is used when each individual provides at least 10 repeated measurements and NCP otherwise. Note this automatic behavior is set only when the argument parameterization = NULL. To set CP parameterization, use parameterization = 'cp'. The default is parameterization = 'ncp'. Note that since brms::brm() does not offer CP parameterization, the brms::brm() generated stancode is first edited internally and then the model is fit using the rstan::rstan() or cmdstanr, depending on the backend choice. Therefore, we caution that CP parameterization should be considered experimental and it may fail if structure of the brms::brm() generated stancode changes in future.

... Further arguments passed to brms::brm(). Can also be used to pass some strictly internal use arguments such as match_sitar_a_form, match_sitar_d_form, sigmamatch_sitar_a_form, displayit, setcolh, setcolb etc.

Details

The *SITAR* is a shape-invariant nonlinear mixed effect growth curve model that fits a population average (i.e., mean average) curve to the data, and aligns each individual's growth trajectory to the

underlying population average curve via a set of (typically) three random effects: the `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 shape invariant model (SIM) was first described 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 is developed by Cole et al. (2010) and has been used extensively for modelling growth data (see Nembidzane et al. 2020 and Sandhu 2020).

The frequentist version of the *SITAR* model can be fit by using an already available R package, the **sitar** (Cole 2022). The framework of Bayesian implementation of the *SITAR* model in **bsitar** package is same as the **sitar** package with the exception that unlike the **sitar** package which uses B spline basis for the natural cubic spline design matrix (by calling the `splines::ns()`), the **bsitar** package uses the truncated power basis approach (see Harrell and others (2001), and Harrell Jr. (2022) for details) to construct the spline design matrix. Note that **bsitar** package builds the spline design matrix on the fly which is then included in the `functions` block of the **Stan** program and hence compiled (via the c++) during the model fit.

Like **sitar** package (Cole et al. 2010), the **bsitar** package fits *SITAR* model with (usually) up to three random effects: the size (parameter defined as `a`), the timing (parameter defined as `b`) and the intensity (parameter defined as `c`). In addition, there is a slope parameter (defined as `d`) that models the variability in the adult slope of the growth curve (See `sitar::sitar()` for details). Please note that author of the **sitar** package (Cole et al. 2010) enforces the inclusion of parameter `d` as a random effects only and therefore excludes it from the fixed fixed structure of the model. However, the **bsitar** package allows inclusion of parameter `d` in fixed and/or in the 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 that captures the deviation of individual trajectories from the population average curve is specified as `random = 'a+b+c'`. Note that user need not to include all the three parameters in the fixed or the random effect structure. For example, a fixed effect version of the *SITAR* model can be fit by setting randoms as an empty string i.e., `random = ''`. Furthermore, the fixed effect structure may include only a sub set of the parameters e.g., size and timing parameters (`fixed = 'a+b'`) or the size and the intensity parameters (`fixed = 'a+c'`). The four parameters version of the *SITAR* model is fit by including parameter `d` in the `fixed` and/or the `random` arguments. Similar to the three parameter *SITAR* model, user can fit model with a sub set of the fixed and/or the random effects.

The **sitar** package internally depends on the **brms** package (see Bürkner 2022; Bürkner 2021). The **brms** can fit a wide range of hierarchical linear and nonlinear regression models including multivariate models. The **brms** itself depends on the **Stan** software program full Bayesian inference (see Stan Development Team 2023; Gelman et al. 2015). Like **brms**, the **bsitar** package allows a wide range of prior specifications that encourage the users to specify priors that actually reflect their prior knowledge about the human growth processes, (such as timing and intensity of the growth spurt). For prior specification, we follow the carefully crafted approaches used in the **brms** and **rstanarm** packages. While **brms** packages use a combination of `normal` and `student_t` distribution for the regression coefficients and the standard deviation of group level random effects and the distributional parameter (`sigma`), the **rstanarm** uses `normal` distribution for regression coefficients and the group level random effects but sets `exponential` distribution for the distributional parameter (`sigma`). We follow use default`normal` distribution for all parameters i.e., regression coefficients and the standard deviation of group level random effects and the distributional parameter. Like **brms** and **rstanarm** packages, the **bsitar** package allows `'autoscaling'` of the scale parameter for location-scale based distributions (such as `normal` and `student_t`). While **rstan-**

arm earlier used to set `autoscale` as `2.5` (recently authors changed this behavior to `FALSE`), the **brms** package sets it as `1.0` or `2.5` depending on the standard deviation of the response variable (See `brms::prior()`). The **bsitar** package, on the other hand, offers full flexibility in choosing the scale factor as any real number (e.g., `autoscale = 5.0`). When `autoscale = TRUE`, the `2.5` is the default scaling factor. We strongly recommend to go through the well documented details on prior specifications used in **brms** and **rstanarm** packages.

Like **brms** package, the **bsitar** package offers a range of tools to evaluate the model fit that include posterior predictive check (see `brms::pp_check()`) and the leave one out (`loo`) cross validation (see `brms::loo()`). Furthermore, while the excellent post-processing support offered by the **brms** package is directly available to the users, the **bsitar** package includes many customized functions that allow for estimation and visualization of population average and individual specific distance (increase in size) and velocity (change in rate of growth), as well as computation of population average and individual specific growth parameters such as age at peak growth velocity (APGV) and the peak growth velocity (PGV).

Finally, the **bsitar** package allows three different types of model specifications: `'univariate'`, `'univariate_by'` and `'multivariate'`. A `'univariate'` fitting involves a single model applied to an outcome whereas both `'univariate_by'` and `'multivariate'` specifications comprise two or more sub models. The `'univariate_by'` fits two or more sub models to an outcome variable defined by a factor variable (e.g, sex). The data are typically stacked and the factor variable is used to set-up the sub models via the `'subset'` option available in the `brms::brm()`. The `'multivariate'` model allows simultaneous modelling of two or more outcomes with joint a distribution of the random effects. For both `'univariate_by'` and `'multivariate'` models, the **bsitar** package allows full flexibility in specifying separate arguments such as predictor variables (`x`), degree of freedom (`df`) for design matrix as well as the priors and the initial values. Furthermore, to enhance the ease of specifying different options and to make it user-friendly, there is no need to enclose the character option(s) in single or double quotes. For example to specify the `'univariate_by'` for sex, the `univariate_by = sex` is same as `univariate_by = 'sex'` or `univariate_by = "sex"`. The same applies for all character string options.

Value

An object of class `brmsfit, bsiatr`, that contains the posterior draws and other useful information about the model.

Note

The package is under continuous development and new models and post-processing features will be added soon.

Author(s)

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

References

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.

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.

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

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

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

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

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

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

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

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

```
# Examples below fits SITAR model to the 'berkeley_exdata' which is a subset
# of the  Berkley height data. The same subset of the  Berkley height data
# has been used as an example data in the vignette for the 'sitar' package.
#
# The Berkley height data comprise of repeated growth measurements made on
```

```
# 66 boys and 70 girls (birth to 21 years).
#
# The subset of the Berkley height data analysed here include growth
# measurements for 70 girls (8 to 18 years).
#
# See 'sitar' package documentation for details on Berkley height data
# (help file ?sitar::berkeley ). The details on subset data for 70 girls is
# provided in the vignette('Fitting_models_with_SITAR', package = 'sitar').

# Fit frequentist SITAR model with df = 5 by using the sitar package

# Get 'berkeley_exdata' data that has been already saved
berkeley_exdata <- getNsObject(berkeley_exdata)

model_ml <- sitar::sitar(x = age, y = height, id = id,
                         df = 5,
                         data = berkeley_exdata,
                         xoffset = 'mean',
                         fixed = 'a+b+c',
                         random = 'a+b+c',
                         a.formula = ~1,
                         b.formula = ~1,
                         c.formula = ~1
                         )


# Fit Bayesian SITAR model

# To avoid mode estimation which takes time, the Bayesian SITAR model fit to
# the 'berkeley_exdata' has been saved as an example fit ('berkeley_exfit').
# The model is fit using 2 chain (1000 iterations per) with thin set as 6 to
# save time and memory. This is because of the package size restrictions
# imposed by the CRAN. Due to the small number of draws, the example model
# fit shown here is only included as an illustration and therefore users
# are advised to refit model with default setting suggested by the Stan Team
# (4 chain with 2000 iterations per chain along with thin = 1).

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

 print(berkeley_exfit)
if(exists('berkeley_exfit')) {
  model <- berkeley_exfit
} else {
 # Fit model with default priors
 # See documentation for prior on each parameter
  model <- bsitar(x = age, y = height, id = id,
                  df = 3,
                  data = berkeley_exdata,
                  xoffset = 'mean',
                  fixed = 'a+b+c',
                  random = 'a+b+c',
```

```
                  a_formula = ~1,
                  b_formula = ~1,
                  c_formula = ~1,
                  threads = brms::threading(NULL),
                  chains = 2, cores = 2, iter = 6000, thin = 15)

# Note that we can test for the sensitivity to the priors by re fitting the
# above model with flat (i.e., uniform) priors on the regression coefficients
# for parameters a, b and c.
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,
                  a_prior_beta = flat,
                  b_prior_beta = flat,
                  c_prior_beta = flat,
                  threads = brms::threading(NULL),
                  chains = 2, cores = 2, iter = 6000, thin = 15)
}

# Generate model summary
summary(model)

# Compare model summary with the maximum likelihood SITAR model
print(model_ml)


# Check model fit via posterior predictive checks. The plot_ppc is a based
# on the pp_check function from the brms package.

plot_ppc(model, ndraws = 100)

# Plot distance and velocity curves using plot_conditional_effects() function.
# This function works exactly same as as conditional_effects() from the brms
# package with the exception that plot_conditional_effects allows for
# plotting velocity curve also.

# Distance
plot_conditional_effects(model, deriv = 0)

# Velocity
plot_conditional_effects(model, deriv = 1)

# Plot distance and velocity curve along with the parameter estimates using
# the plot_curves() function. This function works exactly the same way as
# plot.sitar from the sitar package

plot_curves(model, apv = TRUE)
```

```
# Compare plot with the maximum likelihood SITAR model

plot(model_ml)
```

`expose_model_functions.bgmfit` *Expose user defined Stan function for post-processing*

Description

The **expose_model_functions()** is a wrapper around the `rstan::expose_stan_functions()` to expose user defined Stan function(s). These exposed functions are needed during the post-processing of 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,
  envir = NULL,
  ...
)

expose_model_functions(model, ...)
```

Arguments

`model`	An object of class `bgmfit`.
`scode`	A character string (`Stan` code) containing the user-defined Stan function(s). If `NULL` (default), the `scode` is retrieved from the `model`.
`expose`	A logical (default `TRUE`) to indicate whether to expose functions and add them to the `model` as an attribute.
`select_model`	A character string (default `NULL`) to indicate the model name. This is for internal use only.
`returnobj`	A logical (default `TRUE`) to indicate whether to return the model object. When `expose = TRUE`, then it is advisable to set `returnobj = TRUE` too.

vectorize	A logical (default FALSE) to indicate whether the exposed functions should be vectorized via base::Vectorize(). Note that currently vectorize should be set to FALSE because setting it TRUE may not work as expected.
verbose	An optional argument (logical, default FALSE) to indicate whether to print information collected during setting up the object(s).
envir	Environment used for function evaluation. The default is NULL which will set parent.frame() as default environment. Note that since most of post processing functions are based on **brms**, the functions needed for evaluation should be in the .GlobalEnv. Therefore, it is strongly recommended to set envir = globalenv() (or envir = .GlobalEnv). This is particularly true for the derivatives such as velocity curve.
...	Additional arguments passed to the rstan::expose_stan_functions() function.

Value

An object of class bgmfit if returnobj=TRUE, otherwise invisible NULL.

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 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 avoid model compilation which often takes time

expose_model_functions(model, expose = FALSE)
```

`fitted_draws.bgmfit` *Fitted (expected) values from the posterior draws*

Description

The **fitted_draws()** is a wrapper around the `brms::fitted.brmsfit()` function to obtain fitted values (and their summary) from the posterior draws. See `brms::fitted.brmsfit()` for details.

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,
  ipts = 10,
  deriv = 0,
  deriv_model = TRUE,
  summary = TRUE,
  robust = FALSE,
  transform = 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,
  envir = NULL,
  ...
```

```
)

fitted_draws(model, ...)
```

Arguments

model	An object of class bgmfit.
newdata	An optional data frame to be used in estimation. If NULL (default), the newdata is retrieved from the model.
resp	A character string (default NULL) to specify response variable when processing posterior draws for the 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 be used in estimation. If NULL (default), all draws are used.
draw_ids	An integer indicating the specific posterior draw(s) to be used in estimation (default NULL).
re_formula	Option to indicate whether or not to include the individual/group-level effects in the estimation. When NA (default), the individual-level effects are excluded and therefore population average growth parameters are computed. When NULL, individual-level effects are included in the computation and hence the growth parameters estimates returned are individual-specific. In both situations, (i.e,, NA or NULL), continuous and factor covariate(s) are appropriately included in the estimation. The continuous covariates by default are set to their means (see numeric_cov_at for details) whereas factor covariates are left unaltered thereby allowing estimation of covariate specific population average and individual-specific growth parameter.
allow_new_levels	A flag indicating if new levels of group-level effects are allowed (defaults to FALSE). Only relevant if newdata is provided.
sample_new_levels	Indicates how to sample new levels for grouping factors specified in re_formula. This argument is only relevant if newdata is provided and allow_new_levels is set to TRUE. If "uncertainty" (default), each posterior sample for a new level is drawn from the posterior draws of a randomly chosen existing level. Each posterior sample for a new level may be drawn from a different existing level such that the resulting set of new posterior draws represents the variation across existing levels. If "gaussian", sample new levels from the (multivariate) normal distribution implied by the group-level standard deviations and correlations. This options may be useful for conducting Bayesian power analysis or predicting new levels in situations where relatively few levels where observed in the old_data. If "old_levels", directly sample new levels from the existing levels, where a new level is assigned all of the posterior draws of the same (randomly chosen) existing level.
incl_autocor	A flag indicating if correlation structures originally specified via autocor should be included in the predictions. Defaults to TRUE.

numeric_cov_at An optional (named list) argument to specify the value of continuous covariate(s). The default NULL option set the continuous covariate(s) at 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 varibale 'xx' at 2. The argument numeric_cov_at is ignored when no continuous covariate is included in the model.

levels_id An optional argument to specify the ids for hierarchical model (default NULL). It is used only when model is applied to the data with 3 or more levels of hierarchy. For a two level model, the levels_id is automatically inferred from the model fit. Even for 3 or higher level model, the levels_id is inferred from the model fit but under the assumption that hierarchy is specified from lowest to upper most level i.e, id followed by study where id is nested within the study Note that it is not guaranteed that the levels_id is sorted correctly, and therefore 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 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) such as avg_reffects = list(feby = 'study', reby = NULL, over = 'age').

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

ipts An integer to set the length of the predictor variable to get a smooth velocity curve. The NULL will return original values whereas an integer such as ipts = 10 (default) will interpolate the predictor. It is important to note that these interpolations do not alter the range of predictor when calculating population average and/or the individual specific growth curves.

deriv An integer to indicate whether to estimate distance curve or its derivative (i.e., velocity curve). The deriv = 0 (default) is for the distance curve whereas deriv = 1 for the velocity curve.

deriv_model A logical to specify whether to estimate velocity curve from the derivative function, or the differentiation of the distance curve. The argument deriv_model is set to TRUE for those functions which need velocity curve such as growthparameters() and plot_curves(), and NULL for functions which explicitly use the distance curve (i.e., fitted values) such as loo_validation() and plot_ppc().

summary A logical indicating whether only the estimate should be computed (TRUE), or estimate along with SE and CI should be returned (FALSE, default). Setting summary as FALSE will increase the computation time. Note that summary = FALSE is must to get the correct estimates when re_formula = NULL.

robust A logical to specify the summarize options. 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. Ignored if summary is FALSE.

transform — A function applied to individual draws from the posterior distribution, before computing summaries. The argument transform is based on the marginaleffects::predictions(). This should not be confused with the transform from the brms::posterior_predict() which is now deprecated.

probs — The percentiles to be computed by the quantile function. Only used if summary is TRUE.

xrange — An integer to set the predictor range (i.e., age) when executing the interpolation via ipts. The default NULL sets the individual specific predictor range whereas code xrange = 1 sets identical range for individuals within the same higher grouping variable (e.g., study). Code xrange = 2 sets the identical range across the entire sample. Lastly, a paired numeric values can be supplied e.g., xrange = c(6, 20) to set the range within those values.

xrange_search — A vector of length two, or a character string 'range' to set the range of predictor variable (x) within which growth parameters are searched. This is useful when there is more than one peak and user wants to summarize peak within a given range of the x variable. Default xrange_search = NULL.

parms_eval — A logical to specify whether or not to get growth parameters on the fly. This is for internal use only and mainly needed for compatibility across internal functions.

parms_method — A character to specify the method used to when evaluating parms_eval. The default is getPeak which uses the sitar::getPeak() function from the sitar package. The alternative option is findpeaks that uses this findpeaks from the pracma package. This is for internal use only and mainly needed for compatibility across internal functions.

idata_method — A character string to indicate the interpolation method. The number of of interpolation points is set up the ipts argument. Options available for idata_method are *method 1* (specified as 'm1') and *method 2* (specified as 'm2'). The *method 1* ('m1') is adapted from the the **iapvbs** package and is documented here https://rdrr.io/github/Zhiqiangcao/iapvbs/src/R/exdata.R whereas *method 2* ('m2') is based on the **JMbayes** package as documented here https://github.com/drizopoulos/JMbayes/blob/master/R/dynPred_lme.R. The 'm1' method works by internally constructing the data frame based on the model configuration whereas the method 'm2' uses the exact data frame used in model fit and can be accessed via fit$data. If idata_method = NULL, default, then method 'm2' is automatically set. Note that method 'm1' might fail in some cases when model involves covariates particularly when model is fit as univariate_by. Therefore, it is advised to switch to method 'm2' in case 'm1' results in error.

verbose — An optional argument (logical, default FALSE) to indicate whether to print information collected during setting up the object(s).

fullframe — A logical to indicate whether to return fullframe object in which newdata is bind to the summary estimates. Note that fullframe can not be combined with summary = FALSE. Furthermore, fullframe can only be used when idata_method = 'm2'. A particular use case is when fitting univariate_by model. The fullframe is mainly for internal use only.

dummy_to_factor — A named list (default NULL) that is used to convert dummy variables into a factor variable. The named elements are factor.dummy, factor.name, and

factor.level. The factor.dummy is a vector of character strings that need to be converted to a factor variable whereas the factor.name is a single character string that is used to name the newly created factor variable. The factor.level is used to name the levels of newly created factor. When factor.name is NULL, then the factor name is internally set as 'factor.var'. If factor.level is NULL, then names of factor levels are take from the factor.dummy i.e., the factor levels are assigned same name as factor.dummy. Note that when factor.level is not NULL, its length must be same as the length of the factor.dummy.

expose_function

An optional logical argument to indicate whether to expose Stan functions (default FALSE). Note that if user has already exposed Stan functions during model fit by setting expose_function = TRUE in the bsitar(), then those exposed functions are saved and can be used during post processing of the posterior draws and therefore expose_function is by default set as FALSE in all post processing functions except optimize_model(). For optimize_model(), the default setting is expose_function = NULL. The reason is that each optimized model has different Stan function and therefore it need to be re exposed and saved. The expose_function = NULL implies that the setting for expose_function is taken from the original model fit. Note that expose_function must be set to TRUE when adding fit criteria and/or bayes_R2 during model optimization.

usesavedfuns A logical (default NULL) to indicate whether to use the already exposed and saved Stan functions. Depending on whether the user have exposed Stan functions within the bsitar() call via expose_functions argument in the bsitar(), the usesavedfuns is automatically set to TRUE (if expose_functions = TRUE) or FALSE (if expose_functions = FALSE). Therefore, manual setting of usesavedfuns as TRUE/FALSE is rarely needed. This is for internal purposes only and mainly used during the testing of the functions and therefore should not be used by users as it might lead to unreliable estimates.

clearenvfuns A logical to indicate whether to clear the exposed function from the environment (TRUE) or not (FALSE). If NULL (default), then clearenvfuns is set as TRUE when usesavedfuns is TRUE, and FALSE if usesavedfuns is FALSE.

funlist A list (default NULL) to set function names. This is rarely used because required functions are automatically retrieved internally. A use case of funlist when sigma_formula, sigma_formula_gr, or sigma_formula_gr_str uses a function such as poly(age). See bsitar::bsitar() for details. The funlist provides list of function names which have been defined externally and are available in the base::globalenv(). Note that for functions that need distance and velocity curves such as plot_curves(..., opt = 'dv'), the funlist must include two functions where first will be used for the distance and second for the velocity.

envir Environment used for function evaluation. The default is NULL which will set parent.frame() as default environment. Note that since most of post processing functions are based on **brms**, the functions needed for evaluation should be in the .GlobalEnv. Therefore, it is strongly recommended to set envir = globalenv() (or envir = .GlobalEnv). This is particularly true for the derivatives such as velocity curve.

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

Details

The **fitted_draws()** computes the fitted values from the posterior draws. The brms::fitted.brmsfit() function from the **brms** package can used to get the fitted (distance) values when outcome (e.g., height) is untransformed. However, when the outcome is log or square root transformed, the brms::fitted.brmsfit() function will return the fitted curve on the log or square root scale whereas the **fitted_draws()** function returns the fitted values on the original scale. Furthermore, the **fitted_draws()** also compute the first derivative of (velocity) that too on the original scale after making required back-transformation. Except for these differences, both these functions (i.e., brms::fitted.brmsfit() and fitted_draws()) work in the same manner. In other words, user can specify all the options available in the brms::fitted.brmsfit().

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

brms::fitted.brmsfit()

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
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` *Check and get namespace object if exists*

Description

Check and get namespace object if exists

Usage

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

Arguments

`object`	An object to be retrieved. Note that `object` must be a symbol and not a character string.
`namespace`	A character string specifying the namespace to be checked.
`envir`	An environment to be used (default `NULL`).

Value

An object of same class as input `object`.

Author(s)

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

Examples

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

`growthparameters.bgmfit` *Estimate growth parameters from the model fit*

Description

The **growthparameters()** computes population average and and individual-specific growth parameters (such as age at peak growth velocity) and the uncertainty (standard error, SE and the credible interval, CI). Note that a better alternative is to use `growthparameters_comparison()` function that not only allows estimation of adjusted parameters but also makes it possible to compare these parameters across groups.

Usage

```
## S3 method for class 'bgmfit'
growthparameters(
  model,
  newdata = NULL,
  resp = NULL,
  dpar = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  summary = FALSE,
  robust = FALSE,
  transform = NULL,
  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,
  ipts = 10,
  deriv_model = 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,
  envir = NULL,
  ...
)

growthparameters(model, ...)
```

Arguments

model	An object of class bgmfit.
newdata	An optional data frame to be used in estimation. If NULL (default), the newdata is retrieved from the model.
resp	A character string (default NULL) to specify response variable when processing posterior draws for the 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 be used in estimation. If NULL (default), all draws are used.
draw_ids	An integer indicating the specific posterior draw(s) to be used in estimation (default NULL).
summary	A logical indicating whether only the estimate should be computed (TRUE), or estimate along with SE and CI should be returned (FALSE, default). Setting summary as FALSE will increase the computation time. Note that summary = FALSE is must to get the correct estimates when re_formula = NULL.
robust	A logical to specify the summarize options. 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. Ignored if summary is FALSE.
transform	A function applied to individual draws from the posterior distribution, before computing summaries. The argument transform is based on the marginaleffects::predictions(). This should not be confused with the transform from the brms::posterior_predict() which is now deprecated.
re_formula	Option to indicate whether or not to include the individual/group-level effects in the estimation. When NA (default), the individual-level effects are excluded

and therefore population average growth parameters are computed. When NULL, individual-level effects are included in the computation and hence the growth parameters estimates returned are individual-specific. In both situations, (i.e,, NA or NULL), continuous and factor covariate(s) are appropriately included in the estimation. The continuous covariates by default are set to their means (see numeric_cov_at for details) whereas factor covariates are left unaltered thereby allowing estimation of covariate specific population average and individual-specific growth parameter.

peak
: A logical (default TRUE) to indicate whether or not to calculate the age at peak velocity (APGV) and the peak velocity (PGV) parameters.

takeoff
: A logical (default FALSE) to indicate whether or not to calculate the age at takeoff velocity (ATGV) and the takeoff growth velocity (TGV) parameters.

trough
: A logical (default FALSE) to indicate whether or not to calculate the age at cessation of growth velocity (ACGV) and the cessation of growth velocity (CGV) parameters.

acgv
: A logical (default FALSE) to indicate whether or not to calculate the age at cessation of growth velocity from the velocity curve. If TRUE, age at cessation of growth velocity (ACGV) and the cessation growth velocity (CGV) are calculated based on the percentage of the peak growth velocity as defined by the acgv_velocity argument (see below). The acgv_velocity is typically set at 10 percent of the peak growth velocity. The ACGV and CGV are calculated along with the the uncertainty (SE and CI) around the ACGV and CGV parameters.

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

estimation_method
: A character string to specify the estimation method when calculating the velocity from the posterior draws. The 'fitted' method internally calls the fitted_draws() whereas the option predict calls the predict_draws(). See brms::fitted.brmsfit() and brms::predict.brmsfit() for derails.

allow_new_levels
: A flag indicating if new levels of group-level effects are allowed (defaults to FALSE). Only relevant if newdata is provided.

sample_new_levels
: Indicates how to sample new levels for grouping factors specified in re_formula. This argument is only relevant if newdata is provided and allow_new_levels is set to TRUE. If "uncertainty" (default), each posterior sample for a new level is drawn from the posterior draws of a randomly chosen existing level. Each posterior sample for a new level may be drawn from a different existing level such that the resulting set of new posterior draws represents the variation across existing levels. If "gaussian", sample new levels from the (multivariate) normal distribution implied by the group-level standard deviations and correlations. This options may be useful for conducting Bayesian power analysis or predicting new levels in situations where relatively few levels where observed in the old_data. If "old_levels", directly sample new levels from the existing levels, where a new level is assigned all of the posterior draws of the same (randomly chosen) existing level.

incl_autocor A flag indicating if correlation structures originally specified via autocor should be included in the predictions. Defaults to TRUE.

numeric_cov_at An optional (named list) argument to specify the value of continuous covariate(s). The default NULL option set the continuous covariate(s) at 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 varibale 'xx' at 2. The argument numeric_cov_at is ignored when no continuous covariate is included in the model.

levels_id An optional argument to specify the ids for hierarchical model (default NULL). It is used only when model is applied to the data with 3 or more levels of hierarchy. For a two level model, the levels_id is automatically inferred from the model fit. Even for 3 or higher level model, the levels_id is inferred from the model fit but under the assumption that hierarchy is specified from lowest to upper most level i.e, id followed by study where id is nested within the study Note that it is not guaranteed that the levels_id is sorted correctly, and therefore 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 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) such as avg_reffects = list(feby = 'study', reby = NULL, over = 'age').

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

ipts An integer to set the length of the predictor variable to get a smooth velocity curve. The NULL will return original values whereas an integer such as ipts = 10 (default) will interpolate the predictor. It is important to note that these interpolations do not alter the range of predictor when calculating population average and/or the individual specific growth curves.

deriv_model A logical to specify whether to estimate velocity curve from the derivative function, or the differentiation of the distance curve. The argument deriv_model is set to TRUE for those functions which need velocity curve such as growthparameters() and plot_curves(), and NULL for functions which explicitly use the distance curve (i.e., fitted values) such as loo_validation() and plot_ppc().

conf A numeric value (default 0.95) to compute CI. Internally, the conf is translated into a paired probability values as c((1 - conf)/2, 1 - (1 - conf) / 2). For conf = 0.95, this will compute 95% CI and the variables with lower and upper limits will be named as Q.2.5 and Q.97.5.

xrange An integer to set the predictor range (i.e., age) when executing the interpolation via ipts. The default NULL sets the individual specific predictor range whereas code xrange = 1 sets identical range for individuals within the same higher grouping variable (e.g., study). Code xrange = 2 sets the identical range across the entire sample. Lastly, a paired numeric values can be supplied e.g., xrange = c(6, 20) to set the range within those values.

xrange_search
: A vector of length two, or a character string 'range' to set the range of predictor variable (x) within which growth parameters are searched. This is useful when there is more than one peak and user wants to summarize peak within a given range of the x variable. Default xrange_search = NULL.

digits
: An integer (default 2) to set the decimal argument for the base::round() function.

seed
: An integer (default 123) that is passed to the estimation method.

future
: A logical (default FALSE) to specify whether or not to perform parallel computations. If set to TRUE, the future.apply::future_sapply() function is used to summarize draws.

future_session
: A character string to set the session type when future = TRUE. The 'multisession' (default) options sets the multisession whereas the 'multicore' sets the multicore session. Note that option 'multicore' is not supported on Windows systems. For more details, see future.apply::future_sapply().

cores
: Number of cores to be used when running the parallel computations (if future = TRUE). On non-Windows systems this argument can be set globally via the mc.cores option. For the default NULL option, the number of cores are set automatically by calling the future::availableCores(). The number of cores used are the maximum number of cores avaialble minus one, i.e., future::availableCores() - 1.

parms_eval
: A logical to specify whether or not to get growth parameters on the fly. This is for internal use only and mainly needed for compatibility across internal functions.

idata_method
: A character string to indicate the interpolation method. The number of of interpolation points is set up the ipts argument. Options available for idata_method are *method 1* (specified as 'm1') and *method 2* (specified as 'm2'). The *method 1* ('m1') is adapted from the the **iapvbs** package and is documented here https://rdrr.io/github/Zhiqiangcao/iapvbs/src/R/exdata.R whereas *method 2* ('m2') is based on the **JMbayes** package as documented here https://github.com/drizopoulos/JMbayes/blob/master/R/dynPred_lme.R. The 'm1' method works by internally constructing the data frame based on the model configuration whereas the method 'm2' uses the exact data frame used in model fit and can be accessed via fit$data. If idata_method = NULL, default, then method 'm2' is automatically set. Note that method 'm1' might fail in some cases when model involves covariates particularly when model is fit as univariate_by. Therefore, it is advised to switch to method 'm2' in case 'm1' results in error.

parms_method
: A character to specify the method used to when evaluating parms_eval. The default is getPeak which uses the sitar::getPeak() function from the sitar package. The alternative option is findpeaks that uses this findpeaks from the pracma package. This is for internal use only and mainly needed for compatibility across internal functions.

verbose
: An optional argument (logical, default FALSE) to indicate whether to print information collected during setting up the object(s).

fullframe
: A logical to indicate whether to return fullframe object in which newdata is bind to the summary estimates. Note that fullframe can not be combined with summary = FALSE. Furthermore, fullframe can only be used when idata_method

= 'm2'. A particular use case is when fitting univariate_by model. The fullframe is mainly for internal use only.

dummy_to_factor
: A named list (default NULL) that is used to convert dummy variables into a factor variable. The named elements are factor.dummy, factor.name, and factor.level. The factor.dummy is a vector of character strings that need to be converted to a factor variable whereas the factor.name is a single character string that is used to name the newly created factor variable. The factor.level is used to name the levels of newly created factor. When factor.name is NULL, then the factor name is internally set as 'factor.var'. If factor.level is NULL, then names of factor levels are take from the factor.dummy i.e., the factor levels are assigned same name as factor.dummy. Note that when factor.level is not NULL, its length must be same as the length of the factor.dummy.

expose_function
: An optional logical argument to indicate whether to expose Stan functions (default FALSE). Note that if user has already exposed Stan functions during model fit by setting expose_function = TRUE in the bsitar(), then those exposed functions are saved and can be used during post processing of the posterior draws and therefore expose_function is by default set as FALSE in all post processing functions except optimize_model(). For optimize_model(), the default setting is expose_function = NULL. The reason is that each optimized model has different Stan function and therefore it need to be re exposed and saved. The expose_function = NULL implies that the setting for expose_function is taken from the original model fit. Note that expose_function must be set to TRUE when adding fit criteria and/or bayes_R2 during model optimization.

usesavedfuns
: A logical (default NULL) to indicate whether to use the already exposed and saved Stan functions. Depending on whether the user have exposed Stan functions within the bsitar() call via expose_functions argument in the bsitar(), the usesavedfuns is automatically set to TRUE (if expose_functions = TRUE) or FALSE (if expose_functions = FALSE). Therefore, manual setting of usesavedfuns as TRUE/FALSE is rarely needed. This is for internal purposes only and mainly used during the testing of the functions and therefore should not be used by users as it might lead to unreliable estimates.

clearenvfuns
: A logical to indicate whether to clear the exposed function from the environment (TRUE) or not (FALSE). If NULL (default), then clearenvfuns is set as TRUE when usesavedfuns is TRUE, and FALSE if usesavedfuns is FALSE.

funlist
: A list (default NULL) to set function names. This is rarely used because required functions are automatically retrieved internally. A use case of funlist when sigma_formula, sigma_formula_gr, or sigma_formula_gr_str uses a function such as poly(age). See bsitar::bsitar() for details. The funlist provides list of function names which have been defined externally and are available in the base::globalenv(). Note that for functions that need distance and velocity curves such as plot_curves(..., opt = 'dv'), the funlist must include two functions where first will be used for the distance and second for the velocity.

envir
: Environment used for function evaluation. The default is NULL which will set parent.frame() as default environment. Note that since most of post processing functions are based on **brms**, the functions needed for evaluation should

be in the .GlobalEnv. Therefore, it is strongly recommended to set envir = globalenv() (or envir = .GlobalEnv). This is particularly true for the derivatives such as velocity curve.

... Further arguments passed to brms::fitted.brmsfit() and brms::predict() functions.

Details

The **growthparameters()** internally calls the fitted_draws() or the predict_draws() function to estimate the first derivative based growth parameters for each posterior draw. The growth parameters estimated are 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), and the cessation growth velocity (CGV). The APGV and PGV are estimated by calling the sitar::getPeak() function whereas the ATGV and TGV are estimated by using the sitar::getTakeoff() function. The sitar::getTrough() function is used to estimates ACGV and CGV parameters. The parameters obtained from each posterior draw are then summarized appropriately to get the estimates and the uncertainty (SEs and CIs) around these estimates. Please note that it is not always possible to estimate cessation and takeoff growth parameters when there are no distinct pre-peak or post-peak troughs.

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 each scenario (summary = TRUE/False) are 'Parameter' and 'Estimate' which define the name of the growth parameter (e.g., APGV, PGV etc), and estimate. When summary = TRUE, the three additional columns are 'Est.Error', and a paired vector of names defining the lower and upper limits of the CIs. The CI columns are named as Q with appropriate suffix taken from the percentiles used to construct these intervals (such as Q.2.5 and Q.97.5 where2.5 and 97.5 are the 0.025 and 0.975 percentiles used to compute by the 95% CI by calling the quantile function. When re_formual = NULL, an additional column is added that denotes the individual identifier (typically id).

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

# 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 the 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 the peak
# growth spurt (APGV, PGV. ATGV, TGV)

growthparameters(model, re_formula = NULL, peak = TRUE, takeoff = TRUE)
```

`growthparameters_comparison.bgmfit`

Estimate and compare growth parameters

Description

The **growthparameters_comparison()** function estimates and compare growth parameters such as peak growth velocity and the age at peak growth velocity. This function is a wrapper around the `marginaleffects::comparisons()` and `marginaleffects::avg_comparisons()`. The `marginaleffects::comparison` computes unit-level (conditional) estimates whereas `marginaleffects::avg_comparisons()` return average (marginal) estimates. A detailed explanation is available here. Note that for the current use case, i.e., to estimate and compare growth parameters, the arguments `variables` and `comparion` of `marginaleffects::comparisons()` and `marginaleffects::avg_comparisons()` are modified (see below). Furthermore, comparison of growth parameters is performed via the `hypothesis` argument of the `marginaleffects::comparisons()` and `marginaleffects::avg_comparisons()` functions. Note that **marginaleffects** package is highly flexible and therefore it is expected that user has a strong understanding of its working. Furthermore, since **marginaleffects** package is rapidly evolving, the results obtained from the current implementation should be considered experimental.

Usage

```
## S3 method for class 'bgmfit'
growthparameters_comparison(
  model,
  resp = NULL,
  dpar = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  newdata = NULL,
  datagrid = NULL,
  re_formula = NA,
  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 = NULL,
seed = 123,
future = FALSE,
future_session = "multisession",
usedtplyr = FALSE,
usecollapse = TRUE,
parallel = FALSE,
cores = NULL,
average = FALSE,
plot = FALSE,
showlegends = NULL,
variables = NULL,
deriv = NULL,
deriv_model = NULL,
method = "custom",
pdraws = FALSE,
pdrawsp = FALSE,
pdrawsh = FALSE,
comparison = "difference",
type = NULL,
by = FALSE,
bys = NULL,
conf_level = 0.95,
transform = 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,
  envir = NULL,
  ...
)

growthparameters_comparison(model, ...)
```

Arguments

model	An object of class bgmfit.
resp	A character string (default NULL) to specify response variable when processing posterior draws for the 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 be used in estimation. If NULL (default), all draws are used.
draw_ids	An integer indicating the specific posterior draw(s) to be used in estimation (default NULL).
newdata	An optional data frame to be used in estimation. If NULL (default), the newdata is retrieved from the model.
datagrid	Generate a grid of user-specified values for use in the newdata argument in various functions of the **marginaleffects** package. This is useful to define where in the predictor space we want to evaluate the quantities of interest. See marginaleffects::datagrid() for details. The default value for the datagrid is NULL implying that no custom grid is constructed. To set a data grid, the argument should be a data.frame constructed by using the marginaleffects::datagrid() function, or else a named list which are internally used for setting up the grid. For the user convenience, we also allow setting an empty list datagrid = list() in which case essential arguments such as model, newdata are taken up from the respective arguments specified elsewhere. Further, the level 1 predictor (such as age) and any covariate included in the model fit (e.g., gender) are also automatically inferred from the model object.
re_formula	Option to indicate whether or not to include the individual/group-level effects in the estimation. When NA (default), the individual-level effects are excluded and therefore population average growth parameters are computed. When NULL, individual-level effects are included in the computation and hence the growth parameters estimates returned are individual-specific. In both situations, (i.e,, NA or NULL), continuous and factor covariate(s) are appropriately included in the estimation. The continuous covariates by default are set to their means (see numeric_cov_at for details) whereas factor covariates are left unaltered thereby allowing estimation of covariate specific population average and individual-specific growth parameter.
allow_new_levels	A flag indicating if new levels of group-level effects are allowed (defaults to FALSE). Only relevant if newdata is provided.

sample_new_levels
: Indicates how to sample new levels for grouping factors specified in re_formula. This argument is only relevant if newdata is provided and allow_new_levels is set to TRUE. If "uncertainty" (default), each posterior sample for a new level is drawn from the posterior draws of a randomly chosen existing level. Each posterior sample for a new level may be drawn from a different existing level such that the resulting set of new posterior draws represents the variation across existing levels. If "gaussian", sample new levels from the (multivariate) normal distribution implied by the group-level standard deviations and correlations. This options may be useful for conducting Bayesian power analysis or predicting new levels in situations where relatively few levels where observed in the old_data. If "old_levels", directly sample new levels from the existing levels, where a new level is assigned all of the posterior draws of the same (randomly chosen) existing level.

parameter
: A single character string, or a character vector specifying the growth parameter(s) to be estimated. Options are 'tgv' (takeoff growth velocity), 'atgv' (age at takeoff growth velocity), 'pgv' (peak growth velocity), 'apgv' (age at peak growth velocity), 'cgv' (cessation growth velocity), and 'acgv' (age at cessation growth velocity), and 'all'. If parameter = NULL (default), age at peak growth velocity ('apgv') is estimated where when parameter = 'all', all six parameters are estimated. Note that option 'all' can not be used when argument by is TRUE.

xrange
: An integer to set the predictor range (i.e., age) when executing the interpolation via ipts. The default NULL sets the individual specific predictor range whereas code xrange = 1 sets identical range for individuals within the same higher grouping variable (e.g., study). Code xrange = 2 sets the identical range across the entire sample. Lastly, a paired numeric values can be supplied e.g., xrange = c(6, 20) to set the range within those values.

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

digits
: An integer (default 2) to set the decimal places for the estimated growth parameters. The digits is passed on to the base::round() function.

numeric_cov_at
: An optional (named list) argument to specify the value of continuous covariate(s). The default NULL option set the continuous covariate(s) at 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 varibale 'xx' at 2. The argument numeric_cov_at is ignored when no continuous covariate is 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. An indication to use aux_variables is when post processing functions throw an error such as variable 'x' not found either 'data' or 'data2'

levels_id
: An optional argument to specify the ids for hierarchical model (default NULL). It is used only when model is applied to the data with 3 or more levels of hierarchy. For a two level model, the levels_id is automatically inferred from the model fit. Even for 3 or higher level model, the levels_id is inferred from the model fit but under the assumption that hierarchy is specified from lowest to upper most level i.e, id followed by study where id is nested within the study Note that it is not guaranteed that the levels_id is sorted correctly, and therefore 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 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) such as avg_reffects = list(feby = 'study', reby = NULL, over = 'age').

idata_method
: A character string to indicate the interpolation method. The number of of interpolation points is set up the ipts argument. Options available for idata_method are *method 1* (specified as 'm1') and *method 2* (specified as 'm2'). The *method 1* ('m1') is adapted from the the **iapvbs** package and is documented here https://rdrr.io/github/Zhiqiangcao/iapvbs/src/R/exdata.R whereas *method 2* ('m2') is based on the **JMbayes** package as documented here https://github.com/drizopoulos/JMbayes/blob/master/R/dynPred_lme.R. The 'm1' method works by internally constructing the data frame based on the model configuration whereas the method 'm2' uses the exact data frame used in model fit and can be accessed via fit$data. If idata_method = NULL, default, then method 'm2' is automatically set. Note that method 'm1' might fail in some cases when model involves covariates particularly when model is fit as univariate_by. Therefore, it is advised to switch to method 'm2' in case 'm1' results in error.

ipts
: An integer to set the length of the predictor variable to get a smooth velocity curve. The NULL will return original values whereas an integer such as ipts = 10 (default) will interpolate the predictor. It is important to note that these interpolations do not alter the range of predictor when calculating population average and/or the individual specific growth curves.

seed
: An integer (default 123) that is passed to the estimation method.

future
: A logical (default FALSE) to specify whether or not to perform parallel computations. If set to TRUE, the future.apply::future_sapply() function is used to summarize draws.

future_session
: A character string to set the session type when future = TRUE. The 'multisession' (default) options sets the multisession whereas the 'multicore' sets the multicore session. Note that option 'multicore' is not supported on Windows systems. For more details, see future.apply::future_sapply().

usedtplyr
: A logical (default FALSE) to indicate whether to use the **dtplyr** package for summarizing the draws. The **dtplyr** package uses the **data.table** package as back-end. Note that usedtplyris useful only when the data has a large number of observation. For routine uses, the usedtplyr does not make a large difference in the performance because the **marginaleffects** package itself uses the **data.table** package. The usedtplyr argument is evaluated only when the method = 'custom'.

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

parallel — A logical (default FALSE) to indicate whether to use parallel computation (via **doParallel** and **foreach**) when **usecollapse = TRUE**. Note that when **parallel = TRUE**, the parallel::makeCluster() sets type as "PSOCK" which works on all operating systems including Windows. To set type as "FORK", which is fast (but does not works on Windows system), use **parallel = "FORK"**.

cores — A positive integer (default 1) to set up the number of cores to be used when **parallel = TRUE**. To automatically detect the number of cores, please use **cores = NULL**.

average — A logical to indicate whether to internally call the marginaleffects::comparisons() or the marginaleffects::avg_comparisons() function. If FALSE (default), marginaleffects::comparisons() is called otherwise marginaleffects::avg_comparisons() when average = TRUE.

plot — A logical to specify whether to plot comparisons by calling the marginaleffects::plot_comparisons(function (FALSE) or not (FALSE). If FALSE (default), then marginaleffects::comparisons() or marginaleffects::avg_comparisons() are called to compute predictions (see average for details).

showlegends — An argument to specify whether to show legends (TRUE) or not (FALSE). If NULL (default), then showlegends is internally set to TRUE if re_formula = NA, and FALSE if re_formula = NULL.

variables — For estimating growth parameters in the current use case, the variables is the level 1 predictor such as age/time. The variables is a named list where value is set via the esp argument (default 1e-6). If NULL, the variables is set internally by retrieving the relevant information from the model. Otherwise, user can define it as follows: variables = list('x' = 1e-6) where 'x' is the level 1 predictor. Note that variables = list('age' = 1e-6) is the default behavior for the **marginaleffects** because velocity is typically calculated by differentiating the distance curve via dydx approach, and therefore argument deriv is automatically set as 0 and deriv_model as FALSE. If user want to estimate parameters based on the model based first derivative, then argument deriv must be set as 1 and internally argument variables is defined as variables = list('age' = 0) i.e, original level 1 predictor variable, 'x'. It is important to consider that if default behavior is used i.e, deriv = 0 and variables = list('x' = 1e-6), then user can not pass additional arguments to the variables argument. On the other hand, alternative approach i.e, deriv = 0 and variables = list('x' = 0), additional options can be passed to the marginaleffects::comparisons() and marginaleffects::avg_comparisons() functions.

deriv — A numeric to specify whether to estimate parameters based on the differentiation of the distance curve or the model based first derivative. Please see argument variables for more details.

deriv_model — A logical to specify whether to estimate velocity curve from the derivative function, or the differentiation of the distance curve. The argument deriv_model is set to TRUE for those functions which need velocity curve such as growthparameters() and plot_curves(), and NULL for functions which explicitly use the distance curve (i.e., fitted values) such as loo_validation() and plot_ppc().

method — A character string to specify whether to make computation at post draw stage by using the 'marginaleffects' machinery i.e., marginaleffects::comparisons() (method = 'pkg') or via the custom functions written for efficiency and speed (method = 'custom', default). Note that method = 'pkg' does not work except for very simple cases and therefore and should be used cautiously. The method = 'custom' is the recommended and preferred method because it allows computation of more than one parameter simultaneously (such as 'apgv') and 'pgv', see 'parameter')), method = 'custom' is applied only during the post draw stage, all comparison of multiple parameters simultaneously via the hypothesis argument, and makes it possible to add or return draws (see pdraws and pdraws for details).

pdraws — A character string (default FALSE) to indicate whether to return raw draws (if pdraws = 'return'), add raw draws (if pdraws = 'add') to the final return object, return summary of draws (if pdraws = 'returns'), or add summary of draws (if pdraws = 'adds') to the final return object. See marginaleffects::posterior_draws() for details.

pdrawsp — A character string (default FALSE) to indicate whether to return the posterior draws for parameters (if pdrawsp = 'return'). Note that summary of posterior draws for parameters is the default returned object.

pdrawsh — A character string (default FALSE) to indicate whether to return the posterior draws for parameters (if pdrawsh = 'return'). Note that summary of posterior draws for parameters is the default returned object.

comparison — For estimating growth parameters in the current use case, options allowed for the comparison are 'difference' and 'differenceavg'. Note that comparison is a placeholder and is only used to setup the the internal function that estimates 'parameter' via sitar::getPeak(), sitar::getTakeoff() and sitar::getTrough() functions to estimate various growth parameters. Options 'difference' and 'differenceavg' are internally 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.

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) to specify variables over which parameters need to be summarized.

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

transform
: A function applied to individual draws from the posterior distribution, before computing summaries. The argument transform is based on the marginaleffects::predictions(). This should not be confused with the transform from the brms::posterior_predict() which is now deprecated.

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

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

hypothesis
: specify a hypothesis test or custom contrast using a numeric value, vector, or matrix; a string equation; string; a formula, or a function.
- Numeric:
  - Single value: the null hypothesis used in the computation of Z and p (before applying transform).
  - Vector: Weights to compute a linear combination of (custom contrast between) estimates. Length equal to the number of rows generated by the same function call, but without the hypothesis argument.
  - Matrix: Each column is a vector of weights, as describe above, used to compute a distinct linear combination of (contrast between) estimates. The column names of the matrix are used as labels in the output.
- String equation to specify linear or non-linear hypothesis tests. If the term column uniquely identifies rows, terms 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. 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

- String:
  - "pairwise": 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.
  - "revpairwise", "revreference", "revsequential": inverse of the corresponding hypotheses, as described above.
- Formula:
  - `comparison ~ pairs | group`
  - Left-hand side determines the type of comparison to conduct: `difference` or `ratio`. If the left-hand side is empty, `difference` is chosen.
  - Right-hand side determines the pairs of estimates to compare: `reference`, `sequential`, or `meandev`
  - Optional: Users can supply grouping variables after a vertical bar to conduct comparisons withing subsets.
  - Examples:
    - `~ reference`
    - `ratio ~ pairwise`
    - `difference ~ pairwise | groupid`
- 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 `posterior_draws()` to extract and manipulate the draws directly.
- See the Examples section below and the vignette: https://marginaleffects.com/vignettes/hypothesis.ht

`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. 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 contrast (post draw stage) via the hypothesis argument should be computed. Note that variable(s) specified in the constrats_by should be sub set of the variables included in the 'by' argument. If constrats_by = NULL, then all variables are copied from the 'by' argument (i.e., constrats_by = by) except the level-1 predictor (such as age). This automatic behavior of setting at 'unique' values can be turned off by using constrats_by = FALSE. The constrats_by argument is only evaluated when method = 'custom' and the hypothesis is not NULL.

constrats_at A named list (default FALSE) to specify the values at which estimates and contrast (post draw stage) via the hypothesis argument should be computed. The constrats_at can be specified as a one of the following strings 'max', 'min', 'unique', 'range' (e.g., constrats_at = list(age = 'min')) or else as a numeric value or a numeric vector (e.g., constrats_at = list(age = c(6, 7))). When constrats_at = NULL, any level-1 predictor (such as age) is be automatically set at its 'unique' values i.e., constrats_at = list(age = 'unique'). This automatic behavior of setting at 'unique' values can be turned off by using constrats_at = FALSE. Note that constrats_at only subsets the data that has been set up the marginaleffects::datagrid() or specified as the newdata argument. In case no match is found, an error will be triggered. The constrats_at argument is only evaluated when method = 'custom' and the hypothesis is not NULL.

constrats_subset A named list (default FALSE) to subset the estimates (post draw stage) at which contrast are computed via the hypothesis argument. The use of the constrats_subset argument is similar to the constrats_at with the exception that while constrats_at subsets the data based on the values of a variable, the constrats_at filters the character vector of a variables such as sub-setting individuals constrats_at = list(id = c('id1', 'id2')) where 'id1' and 'id1' are individual identifiers. The constrats_subset argument is only evaluated when method = 'custom' and the hypothesis is not NULL.

reformat A logical (default TRUE) to reformat the output returned by the marginaleffects as a data.frame with column names re-defined as follows: conf.low as Q2.5, and conf.high as Q97.5 (assuming that conf_int = 0.95). Also, following columns are dropped from the data frame: term, contrast, tmp_idx, predicted_lo, predicted_hi, predicted.

estimate_center A character string (default NULL) to specify whether to center estimate as 'mean' or as 'median'. Note that estimate_center is used to set the global options as follows:
options("marginaleffects_posterior_center" = "mean"), or
options("marginaleffects_posterior_center" = "median")
The pre-specified global options are restored on exit via the base::on.exit().

estimate_interval A character string (default NULL) to specify whether to compute credible intervals as equal-tailed intervals, 'eti' or highest density intervals, 'hdi'. Note that estimate_interval is used to set the global options as follows:
options("marginaleffects_posterior_interval" = "eti"), or

options("marginaleffects_posterior_interval" = "hdi")
The pre-specified global options are restored on exit via the base::on.exit().

dummy_to_factor
: A named list (default NULL) that is used to convert dummy variables into a factor variable. The named elements are factor.dummy, factor.name, and factor.level. The factor.dummy is a vector of character strings that need to be converted to a factor variable whereas the factor.name is a single character string that is used to name the newly created factor variable. The factor.level is used to name the levels of newly created factor. When factor.name is NULL, then the factor name is internally set as 'factor.var'. If factor.level is NULL, then names of factor levels are take from the factor.dummy i.e., the factor levels are assigned same name as factor.dummy. Note that when factor.level is not NULL, its length must be same as the length of the factor.dummy.

verbose
: An optional argument (logical, default FALSE) to indicate whether to print information collected during setting up the object(s).

expose_function
: An optional logical argument to indicate whether to expose Stan functions (default FALSE). Note that if user has already exposed Stan functions during model fit by setting expose_function = TRUE in the bsitar(), then those exposed functions are saved and can be used during post processing of the posterior draws and therefore expose_function is by default set as FALSE in all post processing functions except optimize_model(). For optimize_model(), the default setting is expose_function = NULL. The reason is that each optimized model has different Stan function and therefore it need to be re exposed and saved. The expose_function = NULL implies that the setting for expose_function is taken from the original model fit. Note that expose_function must be set to TRUE when adding fit criteria and/or bayes_R2 during model optimization.

usesavedfuns
: A logical (default NULL) to indicate whether to use the already exposed and saved Stan functions. Depending on whether the user have exposed Stan functions within the bsitar() call via expose_functions argument in the bsitar(), the usesavedfuns is automatically set to TRUE (if expose_functions = TRUE) or FALSE (if expose_functions = FALSE). Therefore, manual setting of usesavedfuns as TRUE/FALSE is rarely needed. This is for internal purposes only and mainly used during the testing of the functions and therefore should not be used by users as it might lead to unreliable estimates.

clearenvfuns
: A logical to indicate whether to clear the exposed function from the environment (TRUE) or not (FALSE). If NULL (default), then clearenvfuns is set as TRUE when usesavedfuns is TRUE, and FALSE if usesavedfuns is FALSE.

funlist
: A list (default NULL) to set function names. This is rarely used because required functions are automatically retrieved internally. A use case of funlist when sigma_formula, sigma_formula_gr, or sigma_formula_gr_str uses a function such as poly(age). See bsitar::bsitar() for details. The funlist provides list of function names which have been defined externally and are available in the base::globalenv(). Note that for functions that need distance and velocity curves such as plot_curves(..., opt = 'dv'), the funlist must include two functions where first will be used for the distance and second for the velocity.

envir	Environment used for function evaluation. The default is NULL which will set parent.frame() as default environment. Note that since most of post processing functions are based on **brms**, the functions needed for evaluation should be in the .GlobalEnv. Therefore, it is strongly recommended to set envir = globalenv() (or envir = .GlobalEnv). This is particularly true for the derivatives such as velocity curve.
...	Further arguments passed to brms::fitted.brmsfit() and brms::predict() functions.

Details

The growthparameters_comparison 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 starts and it indicates the beginning of the pubertal growth spurt. The cessation growth velocity indicates the end of the active pubertal growth spurt and is calculated as some percentage of the peak velocity (pgv). Typically, a 10 percent of the pgv is considered as a good indicator of the cessation of the active pubertal growth spurt (Hardin et al. 2022). The percentage is controlled via the acg_velocity argument which takes a positive real value bounded between 0 and 1 (default 0.1 implying 10 percent).

Value

A data frame objects with estimates and CIs for computed parameter(s)

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 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

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

---

loo_validation.bgmfit *Perform leave-one-out (loo) cross-validation*

Description

The **loo_validation()** 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,
```

```
  deriv_model = NULL,
  verbose = FALSE,
  dummy_to_factor = NULL,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  envir = NULL,
  ...
)

loo_validation(model, ...)
```

Arguments

model	An object of class bgmfit.
compare	A flag indicating if the information criteria of the models should be compared to each other via loo::loo_compare().
resp	Optional names of response variables. If specified, predictions are performed only for the specified response variables.
dpar	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
pointwise	A flag indicating whether to compute the full log-likelihood matrix at once or separately for each observation. The latter approach is usually considerably slower but requires much less working memory. Accordingly, if one runs into memory issues, pointwise = TRUE is the way to go.
moment_match	A logical argument to indicate whether loo::loo_moment_match() should be applied on problematic observations. Defaults to FALSE. For most models, moment matching will only work if you have set save_pars = save_pars(all = TRUE) when fitting the model with brms::brm(). See brms::loo_moment_match() for more details.
reloo	A logical argument to indicate whether brms::reloo() should be applied on problematic observations. Defaults to FALSE.
k_threshold	The Pareto k threshold for which observations loo_moment_match or reloo is applied if argument moment_match or reloo is TRUE. Defaults to 0.7. See pareto_k_ids for more details.
save_psis	Should the "psis" object created internally be saved in the returned object? For more details see loo.
moment_match_args	An optional list of additional arguments passed to loo::loo_moment_match().
reloo_args	An optional list of additional arguments passed to brms::reloo().
model_names	If NULL (the default) will use model names derived from deparsing the call. Otherwise will use the passed values as model names.
ndraws	A positive integer indicating the number of posterior draws to be used in estimation. If NULL (default), all draws are used.
draw_ids	An integer indicating the specific posterior draw(s) to be used in estimation (default NULL).

cores — Number of cores to be used when running the parallel computations (if future = TRUE). On non-Windows systems this argument can be set globally via the mc.cores option. For the default NULL option, the number of cores are set automatically by calling the future::availableCores(). The number of cores used are the maximum number of cores avaialble minus one, i.e., future::availableCores() - 1.

deriv_model — A logical to specify whether to estimate velocity curve from the derivative function, or the differentiation of the distance curve. The argument deriv_model is set to TRUE for those functions which need velocity curve such as growthparameters() and plot_curves(), and NULL for functions which explicitly use the distance curve (i.e., fitted values) such as loo_validation() and plot_ppc().

verbose — An optional argument (logical, default FALSE) to indicate whether to print information collected during setting up the object(s).

dummy_to_factor — A named list (default NULL) that is used to convert dummy variables into a factor variable. The named elements are factor.dummy, factor.name, and factor.level. The factor.dummy is a vector of character strings that need to be converted to a factor variable whereas the factor.name is a single character string that is used to name the newly created factor variable. The factor.level is used to name the levels of newly created factor. When factor.name is NULL, then the factor name is internally set as 'factor.var'. If factor.level is NULL, then names of factor levels are take from the factor.dummy i.e., the factor levels are assigned same name as factor.dummy. Note that when factor.level is not NULL, its length must be same as the length of the factor.dummy.

expose_function — An optional logical argument to indicate whether to expose Stan functions (default FALSE). Note that if user has already exposed Stan functions during model fit by setting expose_function = TRUE in the bsitar(), then those exposed functions are saved and can be used during post processing of the posterior draws and therefore expose_function is by default set as FALSE in all post processing functions except optimize_model(). For optimize_model(), the default setting is expose_function = NULL. The reason is that each optimized model has different Stan function and therefore it need to be re exposed and saved. The expose_function = NULL implies that the setting for expose_function is taken from the original model fit. Note that expose_function must be set to TRUE when adding fit criteria and/or bayes_R2 during model optimization.

usesavedfuns — A logical (default NULL) to indicate whether to use the already exposed and saved Stan functions. Depending on whether the user have exposed Stan functions within the bsitar() call via expose_functions argument in the bsitar(), the usesavedfuns is automatically set to TRUE (if expose_functions = TRUE) or FALSE (if expose_functions = FALSE). Therefore, manual setting of usesavedfuns as TRUE/FALSE is rarely needed. This is for internal purposes only and mainly used during the testing of the functions and therefore should not be used by users as it might lead to unreliable estimates.

clearenvfuns — A logical to indicate whether to clear the exposed function from the environment (TRUE) or not (FALSE). If NULL (default), then clearenvfuns is set as TRUE when usesavedfuns is TRUE, and FALSE if usesavedfuns is FALSE.

envir — Environment used for function evaluation. The default is NULL which will set parent.frame() as default environment. Note that since most of post processing functions are based on **brms**, the functions needed for evaluation should be in the .GlobalEnv. Therefore, it is strongly recommended to set envir = globalenv() (or envir = .GlobalEnv). This is particularly true for the derivatives such as velocity curve.

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

Details

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

Value

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

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


loo_validation(model, cores = 1)
```

`marginal_comparison.bgmfit` *Estimate and compare growth curves*

Description

The **marginal_comparison()** function estimates and compare growth curves such as distance and velocity. This function is a wrapper around the `marginaleffects::comparisons()` and `marginaleffects::avg_comparis` The `marginaleffects::comparisons()` computes unit-level (conditional) estimates whereas `marginaleffects::avg_com` return average (marginal) estimates. A detailed explanation is available here. Note that **marginaleffects** package is highly flexible and therefore it is expected that user has a strong understanding of its working. Furthermore, since **marginaleffects** package is rapidly evolving, the results obtained from the current implementation should be considered experimental.

Usage

```
## S3 method for class 'bgmfit'
marginal_comparison(
  model,
  resp = NULL,
  dpar = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  newdata = NULL,
  datagrid = NULL,
  re_formula = NA,
  allow_new_levels = FALSE,
  sample_new_levels = "gaussian",
  xrange = 1,
  digits = 2,
  numeric_cov_at = NULL,
  aux_variables = NULL,
  levels_id = NULL,
  avg_reffects = NULL,
  idata_method = NULL,
  ipts = NULL,
  seed = 123,
  future = FALSE,
  future_session = "multisession",
  cores = NULL,
  average = FALSE,
  plot = FALSE,
  showlegends = NULL,
  variables = NULL,
  deriv = NULL,
  deriv_model = NULL,
  method = "pkg",
```

```
  comparison = "difference",
  type = NULL,
  by = FALSE,
  conf_level = 0.95,
  transform = 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,
  usedtplyr = FALSE,
  dummy_to_factor = NULL,
  verbose = FALSE,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  funlist = NULL,
  envir = NULL,
  ...
)

marginal_comparison(model, ...)
```

Arguments

model	An object of class bgmfit.
resp	A character string (default NULL) to specify response variable when processing posterior draws for the 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 be used in estimation. If NULL (default), all draws are used.
draw_ids	An integer indicating the specific posterior draw(s) to be used in estimation (default NULL).
newdata	An optional data frame to be used in estimation. If NULL (default), the newdata is retrieved from the model.
datagrid	Generate a grid of user-specified values for use in the newdata argument in various functions of the **marginaleffects** package. This is useful to define where in the predictor space we want to evaluate the quantities of interest. See marginaleffects::datagrid() for details. The default value for the datagrid is NULL implying that no custom grid is constructed. To set a data grid, the argument should be a data.frame

constructed by using the marginaleffects::datagrid() function, or else a named list which are internally used for setting up the grid. For the user convenience, we also allow setting an empty list datagrid = list() in which case essential arguments such as model, newdata are taken up from the respective arguments specified elsewhere. Further, the level 1 predictor (such as age) and any covariate included in the model fit (e.g., gender) are also automatically inferred from the model object.

re_formula Option to indicate whether or not to include the individual/group-level effects in the estimation. When NA (default), the individual-level effects are excluded and therefore population average growth parameters are computed. When NULL, individual-level effects are included in the computation and hence the growth parameters estimates returned are individual-specific. In both situations, (i.e,, NA or NULL), continuous and factor covariate(s) are appropriately included in the estimation. The continuous covariates by default are set to their means (see numeric_cov_at for details) whereas factor covariates are left unaltered thereby allowing estimation of covariate specific population average and individual-specific growth parameter.

allow_new_levels
A flag indicating if new levels of group-level effects are allowed (defaults to FALSE). Only relevant if newdata is provided.

sample_new_levels
Indicates how to sample new levels for grouping factors specified in re_formula. This argument is only relevant if newdata is provided and allow_new_levels is set to TRUE. If "uncertainty" (default), each posterior sample for a new level is drawn from the posterior draws of a randomly chosen existing level. Each posterior sample for a new level may be drawn from a different existing level such that the resulting set of new posterior draws represents the variation across existing levels. If "gaussian", sample new levels from the (multivariate) normal distribution implied by the group-level standard deviations and correlations. This options may be useful for conducting Bayesian power analysis or predicting new levels in situations where relatively few levels where observed in the old_data. If "old_levels", directly sample new levels from the existing levels, where a new level is assigned all of the posterior draws of the same (randomly chosen) existing level.

xrange An integer to set the predictor range (i.e., age) when executing the interpolation via ipts. The default NULL sets the individual specific predictor range whereas code xrange = 1 sets identical range for individuals within the same higher grouping variable (e.g., study). Code xrange = 2 sets the identical range across the entire sample. Lastly, a paired numeric values can be supplied e.g., xrange = c(6, 20) to set the range within those values.

digits An integer (default 2) to set the decimal places for the estimates. The digits is passed on to the base::round() function.

numeric_cov_at An optional (named list) argument to specify the value of continuous covariate(s). The default NULL option set the continuous covariate(s) at 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 varibale 'xx' at 2. The argument numeric_cov_at is ignored when no continuous covariate is 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. An indication to use aux_variables is when post processing functions throw an error such as variable 'x' not found either 'data' or 'data2'

levels_id
: An optional argument to specify the ids for hierarchical model (default NULL). It is used only when model is applied to the data with 3 or more levels of hierarchy. For a two level model, the levels_id is automatically inferred from the model fit. Even for 3 or higher level model, the levels_id is inferred from the model fit but under the assumption that hierarchy is specified from lowest to upper most level i.e, id followed by study where id is nested within the study Note that it is not guaranteed that the levels_id is sorted correctly, and therefore 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 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) such as avg_reffects = list(feby = 'study', reby = NULL, over = 'age').

idata_method
: A character string to indicate the interpolation method. The number of of interpolation points is set up the ipts argument. Options available for idata_method are *method 1* (specified as 'm1') and *method 2* (specified as 'm2'). The *method 1* ('m1') is adapted from the the **iapvbs** package and is documented here https://rdrr.io/github/Zhiqiangcao/iapvbs/src/R/exdata.R whereas *method 2* ('m2') is based on the **JMbayes** package as documented here https://github.com/drizopoulos/JMbayes/blob/master/R/dynPred_lme.R. The 'm1' method works by internally constructing the data frame based on the model configuration whereas the method 'm2' uses the exact data frame used in model fit and can be accessed via fit$data. If idata_method = NULL, default, then method 'm2' is automatically set. Note that method 'm1' might fail in some cases when model involves covariates particularly when model is fit as univariate_by. Therefore, it is advised to switch to method 'm2' in case 'm1' results in error.

ipts
: An integer to set the length of the predictor variable to get a smooth velocity curve. The NULL will return original values whereas an integer such as ipts = 10 (default) will interpolate the predictor. It is important to note that these interpolations do not alter the range of predictor when calculating population average and/or the individual specific growth curves.

seed
: An integer (default 123) that is passed to the estimation method.

future
: A logical (default FALSE) to specify whether or not to perform parallel computations. If set to TRUE, the future.apply::future_sapply() function is used to summarize draws.

future_session
: A character string to set the session type when future = TRUE. The 'multisession' (default) options sets the multisession whereas the 'multicore' sets the multicore session. Note that option 'multicore' is not supported on Windows systems. For more details, see future.apply::future_sapply().

cores
: Number of cores to be used when running the parallel computations (if future = TRUE). On non-Windows systems this argument can be set globally via the

mc.cores option. For the default NULL option, the number of cores are set automatically by calling the future::availableCores(). The number of cores used are the maximum number of cores avaialble minus one, i.e., future::availableCores() - 1.

average — A logical to indicate whether to internally call the marginaleffects::comparisons() or the marginaleffects::avg_comparisons() function. If FALSE (default), marginaleffects::comparisons() is called otherwise marginaleffects::avg_comparisons() when average = TRUE.

plot — A logical to specify whether to plot comparisons by calling the marginaleffects::plot_comparisons(function (FALSE) or not (FALSE). If FALSE (default), then marginaleffects::comparisons() or marginaleffects::avg_comparisons() are called to compute predictions (see average for details).

showlegends — An argument to specify whether to show legends (TRUE) or not (FALSE). If NULL (default), then showlegends is internally set to TRUE if re_formula = NA, and FALSE if re_formula = NULL.

variables — For estimating growth parameters in the current use case, the variables is the level 1 predictor such as age/time. The variables is a named list where value is set via the esp argument (default 1e-6). If NULL, the variables is set internally by retrieving the relevant information from the model. Otherwise, user can define it as follows: variables = list('x' = 1e-6) where 'x' is the level 1 predictor. Note that variables = list('age' = 1e-6) is the default behavior for the **marginaleffects** because velocity is typically calculated by differentiating the distance curve via dydx approach, and therefore argument deriv is automatically set as 0 and deriv_model as FALSE. If user want to estimate parameters based on the model based first derivative, then argument deriv must be set as 1 and internally argument variables is defined as variables = list('age' = 0) i.e, original level 1 predictor variable, 'x'. It is important to consider that if default behavior is used i.e, deriv = 0 and variables = list('x' = 1e-6), then user can not pass additional arguments to the variables argument. On the other hand, alternative approach i.e, deriv = 0 and variables = list('x' = 0), additional options can be passed to the marginaleffects::comparisons() and marginaleffects::avg_comparisons() functions.

deriv — A numeric to specify whether to estimate parameters based on the differentiation of the distance curve or the model based first derivative. Please see argument variables for more details.

deriv_model — A logical to specify whether to estimate velocity curve from the derivative function, or the differentiation of the distance curve. The argument deriv_model is set to TRUE for those functions which need velocity curve such as growthparameters() and plot_curves(), and NULL for functions which explicitly use the distance curve (i.e., fitted values) such as loo_validation() and plot_ppc().

method — A character string to specify whether to make computation at post draw stage by using the 'marginaleffects' machinery i.e., marginaleffects::comparisons() (method = 'pkg') or via the custom functions written for efficiency and speed (method = 'custom', default). Note that method = 'custom' is useful and rather needed when testing hypotheses. Note that when method = 'custom', marginaleffects::predictions and not the marginaleffects::comparisons() is used internally.

`comparison` For estimating growth parameters in the current use case, options allowed for the `comparison` are `'difference'` and `'differenceavg'`. Note that `comparison` is a placeholder and is only used to setup the the internal function that estimates `'parameter'` via `sitar::getPeak()`, `sitar::getTakeoff()` and `sitar::getTrough()` functions to estimate various growth parameters. Options `'difference'` and `'differenceavg'` are internally 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.

`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` A function applied to individual draws from the posterior distribution, before computing summaries. The argument `transform` is based on the `marginaleffects::predictions()`. This should not be confused with the `transform` from the `brms::posterior_predict()` which is now deprecated.

`cross`

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

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

- string: column name of the weights variable in `newdata`. When supplying a column name to `wts`, it is recommended to supply the original data (including the weights variable) explicitly to `newdata`.
- numeric: vector of length equal to the number of rows in the original data or in `newdata` (if supplied).
- FALSE: Equal weights.

- TRUE: Extract weights from the fitted object with `insight::find_weights()` and use them when taking weighted averages of estimates. Warning: `newdata=datagrid()` returns a single average weight, which is equivalent to using `wts=FALSE`

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

- Numeric:
  - Single value: the null hypothesis used in the computation of Z and p (before applying `transform`).
  - Vector: Weights to compute a linear combination of (custom contrast between) estimates. Length equal to the number of rows generated by the same function call, but without the `hypothesis` argument.
  - Matrix: Each column is a vector of weights, as describe above, used to compute a distinct linear combination of (contrast between) estimates. The column names of the matrix are used as labels in the output.
- String equation to specify linear or non-linear hypothesis tests. If the `term` column uniquely identifies rows, terms 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. 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`
- String:
  - "pairwise": 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.
  - "revpairwise", "revreference", "revsequential": inverse of the corresponding hypotheses, as described above.
- Formula:
  - `comparison ~ pairs | group`
  - Left-hand side determines the type of comparison to conduct: `difference` or `ratio`. If the left-hand side is empty, `difference` is chosen.
  - Right-hand side determines the pairs of estimates to compare: `reference`, `sequential`, or `meandev`
  - Optional: Users can supply grouping variables after a vertical bar to conduct comparisons withing subsets.
  - Examples:

    `* ~ reference`

    - ratio ~ pairwise
    - difference ~ pairwise | groupid
  - 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 posterior_draws() to extract and manipulate the draws directly.
  - See the Examples section below and the vignette: https://marginaleffects.com/vignettes/hypothesis.ht

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. See Details section below.

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

constrats_by
: A character vector (default NULL) specifying the variable(s) by which hypotheses (post draw stage) should be tested (see hypothesis argument). Note that variable(s) specified in the constrats_by should be sub set of the variables included in the 'by' argument.

constrats_at
: A character vector (default NULL) specifying the variable(s) at which hypotheses (post draw stage) should be tested (see hypothesis argument). Note that variable(s) specified in the constrats_at should be sub set of the variables included in the 'by' argument. The constrats_at is particularly useful when number of rows in the estimates is large. This is because the **marginaleffects** does not allow hypotheses testing when the number of rows in the estimates is more that 25.

reformat
: A logical (default TRUE) to reformat the output returned by the marginaleffects as a data.frame with column names re-defined as follows: conf.low as Q2.5, and conf.high as Q97.5 (assuming that conf_int = 0.95). Also, following columns are dropped from the data frame: term, contrast, tmp_idx, predicted_lo, predicted_hi, predicted.

estimate_center
: A character string (default NULL) to specify whether to center estimate as 'mean' or as 'median'. Note that estimate_center is used to set the global options as follows:
options("marginaleffects_posterior_center" = "mean"), or
options("marginaleffects_posterior_center" = "median")
The pre-specified global options are restored on exit via the base::on.exit().

estimate_interval
: A character string (default NULL) to specify whether to compute credible intervals as equal-tailed intervals, 'eti' or highest density intervals, 'hdi'. Note

that estimate_interval is used to set the global options as follows: options("marginaleffects_posterior_interval" = "eti"), or options("marginaleffects_posterior_interval" = "hdi") The pre-specified global options are restored on exit via the base::on.exit().

usedtplyr A logical (default FALSE) to indicate whether to use the **dtplyr** package for summarizing the draws. The **dtplyr** package uses the **data.table** package as back-end. Note that usedtplyris useful only when the data has a large number of observation. For routine uses, the usedtplyr does not make a large difference in the performance because the **marginaleffects** package itself uses the **data.table** package. The usedtplyr argument is evaluated only when the method = 'custom'.

dummy_to_factor

A named list (default NULL) that is used to convert dummy variables into a factor variable. The named elements are factor.dummy, factor.name, and factor.level. The factor.dummy is a vector of character strings that need to be converted to a factor variable whereas the factor.name is a single character string that is used to name the newly created factor variable. The factor.level is used to name the levels of newly created factor. When factor.name is NULL, then the factor name is internally set as 'factor.var'. If factor.level is NULL, then names of factor levels are take from the factor.dummy i.e., the factor levels are assigned same name as factor.dummy. Note that when factor.level is not NULL, its length must be same as the length of the factor.dummy.

verbose An optional argument (logical, default FALSE) to indicate whether to print information collected during setting up the object(s).

expose_function

An optional logical argument to indicate whether to expose Stan functions (default FALSE). Note that if user has already exposed Stan functions during model fit by setting expose_function = TRUE in the bsitar(), then those exposed functions are saved and can be used during post processing of the posterior draws and therefore expose_function is by default set as FALSE in all post processing functions except optimize_model(). For optimize_model(), the default setting is expose_function = NULL. The reason is that each optimized model has different Stan function and therefore it need to be re exposed and saved. The expose_function = NULL implies that the setting for expose_function is taken from the original model fit. Note that expose_function must be set to TRUE when adding fit criteria and/or bayes_R2 during model optimization.

usesavedfuns A logical (default NULL) to indicate whether to use the already exposed and saved Stan functions. Depending on whether the user have exposed Stan functions within the bsitar() call via expose_functions argument in the bsitar(), the usesavedfuns is automatically set to TRUE (if expose_functions = TRUE) or FALSE (if expose_functions = FALSE). Therefore, manual setting of usesavedfuns as TRUE/FALSE is rarely needed. This is for internal purposes only and mainly used during the testing of the functions and therefore should not be used by users as it might lead to unreliable estimates.

clearenvfuns A logical to indicate whether to clear the exposed function from the environment (TRUE) or not (FALSE). If NULL (default), then clearenvfuns is set as TRUE when usesavedfuns is TRUE, and FALSE if usesavedfuns is FALSE.

funlist	A list (default NULL) to set function names. This is rarely used because required functions are automatically retrieved internally. A use case of funlist when sigma_formula, sigma_formula_gr, or sigma_formula_gr_str uses a function such as poly(age). See bsitar::bsitar() for details. The funlist provides list of function names which have been defined externally and are available in the base::globalenv(). Note that for functions that need distance and velocity curves such as plot_curves(..., opt = 'dv'), the funlist must include two functions where first will be used for the distance and second for the velocity.
envir	Environment used for function evaluation. The default is NULL which will set parent.frame() as default environment. Note that since most of post processing functions are based on **brms**, the functions needed for evaluation should be in the .GlobalEnv. Therefore, it is strongly recommended to set envir = globalenv() (or envir = .GlobalEnv). This is particularly true for the derivatives such as velocity curve.
...	Further arguments passed to brms::fitted.brmsfit() and brms::predict() functions.

Value

A data frame objects with estimates and CIs for computed parameter(s), or a list when method = 'custom' is 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 that since no covariate is part of the model, the 'marginal_comparison'
# doesn't make sense here. It's included only as a dummy example.

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

```
model <- berkeley_exfit

marginal_comparison(model, draw_ids = 1)
```

`marginal_draws.bgmfit` *Estimate growth curves*

Description

The **marginal_draws()** function estimates and plots growth curves (distance and velocity) by using **marginaleffects** package as back-end. This function can compute growth curves (via `marginaleffects::predictions()`), average growth curves (via `marginaleffects::avg_predictions()`) or plot growth curves (via `marginaleffects::plot_predictions()`). Please see here for details. Note that **marginaleffects** package is highly flexible and therefore it is expected that user has a strong understanding of its working. Furthermore, since **marginaleffects** package is rapidly evolving, the results obtained from the current implementation should be considered experimental.

Usage

```
## S3 method for class 'bgmfit'
marginal_draws(
  model,
  resp = NULL,
  dpar = NULL,
  ndraws = NULL,
  draw_ids = NULL,
  newdata = NULL,
  datagrid = NULL,
  re_formula = NA,
  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 = NULL,
  seed = 123,
  future = FALSE,
  future_session = "multisession",
  usedtplyr = FALSE,
```

```
  usecollapse = TRUE,
  cores = NULL,
  fullframe = FALSE,
  average = FALSE,
  plot = FALSE,
  showlegends = NULL,
  variables = NULL,
  condition = NULL,
  deriv = 0,
  deriv_model = TRUE,
  method = "pkg",
  pdrawsp = FALSE,
  pdrawsh = FALSE,
  type = NULL,
  by = NULL,
  conf_level = 0.95,
  transform = NULL,
  byfun = NULL,
  wts = NULL,
  hypothesis = NULL,
  equivalence = 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,
  envir = NULL,
  ...
)

marginal_draws(model, ...)
```

Arguments

model	An object of class bgmfit.
resp	A character string (default NULL) to specify response variable when processing posterior draws for the 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 be used in estimation. If NULL (default), all draws are used.

draw_ids
: An integer indicating the specific posterior draw(s) to be used in estimation (default NULL).

newdata
: An optional data frame to be used in estimation. If NULL (default), the newdata is retrieved from the model.

datagrid
: Generate a grid of user-specified values for use in the newdata argument in various functions of the **marginaleffects** package. This is useful to define where in the predictor space we want to evaluate the quantities of interest. See marginaleffects::datagrid() for details. The default value for the datagrid is NULL implying that no custom grid is constructed. To set a data grid, the argument should be a data.frame constructed by using the marginaleffects::datagrid() function, or else a named list which are internally used for setting up the grid. For the user convenience, we also allow setting an empty list datagrid = list() in which case essential arguments such as model, newdata are taken up from the respective arguments specified elsewhere. Further, the level 1 predictor (such as age) and any covariate included in the model fit (e.g., gender) are also automatically inferred from the model object.

re_formula
: Option to indicate whether or not to include the individual/group-level effects in the estimation. When NA (default), the individual-level effects are excluded and therefore population average growth parameters are computed. When NULL, individual-level effects are included in the computation and hence the growth parameters estimates returned are individual-specific. In both situations, (i.e,, NA or NULL), continuous and factor covariate(s) are appropriately included in the estimation. The continuous covariates by default are set to their means (see numeric_cov_at for details) whereas factor covariates are left unaltered thereby allowing estimation of covariate specific population average and individual-specific growth parameter.

allow_new_levels
: A flag indicating if new levels of group-level effects are allowed (defaults to FALSE). Only relevant if newdata is provided.

sample_new_levels
: Indicates how to sample new levels for grouping factors specified in re_formula. This argument is only relevant if newdata is provided and allow_new_levels is set to TRUE. If "uncertainty" (default), each posterior sample for a new level is drawn from the posterior draws of a randomly chosen existing level. Each posterior sample for a new level may be drawn from a different existing level such that the resulting set of new posterior draws represents the variation across existing levels. If "gaussian", sample new levels from the (multivariate) normal distribution implied by the group-level standard deviations and correlations. This options may be useful for conducting Bayesian power analysis or predicting new levels in situations where relatively few levels where observed in the old_data. If "old_levels", directly sample new levels from the existing levels, where a new level is assigned all of the posterior draws of the same (randomly chosen) existing level.

parameter
: A single character string, or a character vector specifying the growth parameter(s) to be estimated. Options are 'tgv' (takeoff growth velocity), 'atgv' (age at takeoff growth velocity), 'pgv' (peak growth velocity), 'apgv' (age at peak growth velocity), 'cgv' (cessation growth velocity), and 'acgv' (age at

cessation growth velocity), and 'all'. If parameter = NULL (default), age at peak growth velocity ('apgv') is estimated where when parameter = 'all', all six parameters are estimated. Note that option 'all' can not be used when argument by is TRUE.

xrange An integer to set the predictor range (i.e., age) when executing the interpolation via ipts. The default NULL sets the individual specific predictor range whereas code xrange = 1 sets identical range for individuals within the same higher grouping variable (e.g., study). Code xrange = 2 sets the identical range across the entire sample. Lastly, a paired numeric values can be supplied e.g., xrange = c(6, 20) to set the range within those values.

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

digits An integer (default 2) to set the decimal argument for the base::round() function.

numeric_cov_at An optional (named list) argument to specify the value of continuous covariate(s). The default NULL option set the continuous covariate(s) at 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 varibale 'xx' at 2. The argument numeric_cov_at is ignored when no continuous covariate is 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. An indication to use aux_variables is when post processing functions throw an error such as variable 'x' not found either 'data' or 'data2'

levels_id An optional argument to specify the ids for hierarchical model (default NULL). It is used only when model is applied to the data with 3 or more levels of hierarchy. For a two level model, the levels_id is automatically inferred from the model fit. Even for 3 or higher level model, the levels_id is inferred from the model fit but under the assumption that hierarchy is specified from lowest to upper most level i.e, id followed by study where id is nested within the study Note that it is not guaranteed that the levels_id is sorted correctly, and therefore 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 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) such as avg_reffects = list(feby = 'study', reby = NULL, over = 'age').

idata_method A character string to indicate the interpolation method. The number of of interpolation points is set up the ipts argument. Options available for idata_method

are *method 1* (specified as 'm1') and *method 2* (specified as 'm2'). The *method 1* ('m1') is adapted from the the **iapvbs** package and is documented here https://rdrr.io/github/Zhiqiangcao/iapvbs/src/R/exdata.R whereas *method 2* ('m2') is based on the **JMbayes** package as documented here https://github.com/drizopoulos/JMbayes/blob/master/R/dynPred_lme.R. The 'm1' method works by internally constructing the data frame based on the model configuration whereas the method 'm2' uses the exact data frame used in model fit and can be accessed via fit$data. If idata_method = NULL, default, then method 'm2' is automatically set. Note that method 'm1' might fail in some cases when model involves covariates particularly when model is fit as univariate_by. Therefore, it is advised to switch to method 'm2' in case 'm1' results in error.

ipts — An integer to set the length of the predictor variable to get a smooth velocity curve. The NULL will return original values whereas an integer such as ipts = 10 (default) will interpolate the predictor. It is important to note that these interpolations do not alter the range of predictor when calculating population average and/or the individual specific growth curves.

seed — An integer (default 123) that is passed to the estimation method.

future — A logical (default FALSE) to specify whether or not to perform parallel computations. If set to TRUE, the future.apply::future_sapply() function is used to summarize draws.

future_session — A character string to set the session type when future = TRUE. The 'multisession' (default) options sets the multisession whereas the 'multicore' sets the multicore session. Note that option 'multicore' is not supported on Windows systems. For more details, see future.apply::future_sapply().

usedtplyr — A logical (default FALSE) to indicate whether to use the **dtplyr** package for summarizing the draws. The **dtplyr** package uses the **data.table** package as back-end. Note that usedtplyris useful only when the data has a large number of observation. For routine uses, the usedtplyr does not make a large difference in the performance because the **marginaleffects** package itself uses the **data.table** package. The usedtplyr argument is evaluated only when the method = 'custom'.

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

cores — Number of cores to be used when running the parallel computations (if future = TRUE). On non-Windows systems this argument can be set globally via the mc.cores option. For the default NULL option, the number of cores are set automatically by calling the future::availableCores(). The number of cores used are the maximum number of cores avaialble minus one, i.e., future::availableCores() - 1.

fullframe — A logical to indicate whether to return fullframe object in which newdata is bind to the summary estimates. Note that fullframe can not be combined with summary = FALSE. Furthermore, fullframe can only be used when idata_method = 'm2'. A particular use case is when fitting univariate_by model. The fullframe is mainly for internal use only.

average — A logical to indicate whether to internally call the marginaleffects::predictions() or the marginaleffects::avg_predictions() function. If FALSE (default),

marginaleffects::predictions() is called otherwise marginaleffects::avg_predictions() when average = TRUE.

plot — A logical to specify whether to plot predictions by calling the marginaleffects::plot_predictions() function (FALSE) or not (FALSE). If FALSE (default), then marginaleffects::predictions() or marginaleffects::avg_predictions() are called to compute predictions (see average for details).

showlegends — An argument to specify whether to show legends (TRUE) or not (FALSE). If NULL (default), then showlegends is internally set to TRUE if re_formula = NA, and FALSE if re_formula = NULL.

variables — For estimating growth parameters in the current use case, the variables is the level 1 predictor such as age/time. The variables is a named list where value is set via the esp argument (default 1e-6). If NULL, the variables is set internally by retrieving the relevant information from the model. Otherwise, user can define it as follows: variables = list('x' = 1e-6) where 'x' is the level 1 predictor. Note that variables = list('age' = 1e-6) is the default behavior for the **marginaleffects** because velocity is typically calculated by differentiating the distance curve via dydx approach, and therefore argument deriv is automatically set as 0 and deriv_model as FALSE. If user want to estimate parameters based on the model based first derivative, then argument deriv must be set as 1 and internally argument variables is defined as variables = list('age' = 0) i.e, original level 1 predictor variable, 'x'. It is important to consider that if default behavior is used i.e, deriv = 0 and variables = list('x' = 1e-6), then user can not pass additional arguments to the variables argument. On the other hand, alternative approach i.e, deriv = 0 and variables = list('x' = 0), additional options can be passed to the marginaleffects::comparisons() and marginaleffects::avg_comparisons() functions.

condition — Conditional predictions

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

deriv — An integer to indicate whether to estimate distance curve or its derivative (i.e., velocity curve). The deriv = 0 (default) is for the distance curve whereas deriv = 1 for the velocity curve.

deriv_model — A logical to specify whether to estimate velocity curve from the derivative function, or the differentiation of the distance curve. The argument deriv_model is set to TRUE for those functions which need velocity curve such as growthparameters()

and plot_curves(), and NULL for functions which explicitly use the distance curve (i.e., fitted values) such as loo_validation() and plot_ppc().

method — A character string to specify whether to make computation at post draw stage by using the 'marginaleffects' machinery i.e., marginaleffects::comparisons() (method = 'pkg') or via the custom functions written for efficiency and speed (method = 'custom', default). Note that method = 'custom' is useful and rather needed when testing hypotheses. Note that when method = 'custom', marginaleffects::predictions and not the marginaleffects::comparisons() is used internally.

pdrawsp — A character string (default FALSE) to indicate whether to return the posterior draws for parameters (if pdrawsp = 'return'). Note that summary of posterior draws for parameters is the default returned object.

pdrawsh — A character string (default FALSE) to indicate whether to return the posterior draws for parameters (if pdrawsh = 'return'). Note that summary of posterior draws for parameters is the default returned object.

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.

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 — A function applied to individual draws from the posterior distribution, before computing summaries. The argument transform is based on the marginaleffects::predictions(). This should not be confused with the transform from the brms::posterior_predict() which is now deprecated.

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

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

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

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

- Numeric:
  - Single value: the null hypothesis used in the computation of Z and p (before applying `transform`).
  - Vector: Weights to compute a linear combination of (custom contrast between) estimates. Length equal to the number of rows generated by the same function call, but without the `hypothesis` argument.
  - Matrix: Each column is a vector of weights, as describe above, used to compute a distinct linear combination of (contrast between) estimates. The column names of the matrix are used as labels in the output.
- String equation to specify linear or non-linear hypothesis tests. If the `term` column uniquely identifies rows, terms 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. 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`
- String:
  - "pairwise": 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.
  - "revpairwise", "revreference", "revsequential": inverse of the corresponding hypotheses, as described above.
- Formula:
  - `comparison ~ pairs | group`
  - Left-hand side determines the type of comparison to conduct: `difference` or `ratio`. If the left-hand side is empty, `difference` is chosen.

  - Right-hand side determines the pairs of estimates to compare: `reference`, `sequential`, or `meandev`
  - Optional: Users can supply grouping variables after a vertical bar to conduct comparisons withing subsets.
  - Examples:
    * `~ reference`
    * `ratio ~ pairwise`
    * `difference ~ pairwise | groupid`
- 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 `posterior_draws()` to extract and manipulate the draws directly.
- See the Examples section below and the vignette: https://marginaleffects.com/vignettes/hypothesis.ht

`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. See Details section below.

`constrats_by` A character vector (default `NULL`) specifying the variable(s) by which hypotheses (post draw stage) should be tested (see `hypothesis` argument). Note that variable(s) specified in the `constrats_by` should be sub set of the variables included in the `'by'` argument.

`constrats_at` A character vector (default `NULL`) specifying the variable(s) at which hypotheses (post draw stage) should be tested (see `hypothesis` argument). Note that variable(s) specified in the `constrats_at` should be sub set of the variables included in the `'by'` argument. The `constrats_at` is particularly useful when number of rows in the estimates is large. This is because the **marginaleffects** does not allow hypotheses testing when the number of rows in the estimates is more that 25.

`reformat` A logical (default `TRUE`) to reformat the output returned by the `marginaleffects` as a data.frame with column names re-defined as follows: `conf.low` as `Q2.5`, and `conf.high` as `Q97.5` (assuming that `conf_int = 0.95`). Also, following columns are dropped from the data frame: `term`, `contrast`, `tmp_idx`, `predicted_lo`, `predicted_hi`, `predicted`.

`estimate_center`

A character string (default `NULL`) to specify whether to center estimate as `'mean'` or as `'median'`. Note that `estimate_center` is used to set the global options as follows:
`options("marginaleffects_posterior_center" = "mean")`, or
`options("marginaleffects_posterior_center" = "median")`
The pre-specified global options are restored on exit via the `base::on.exit()`.

estimate_interval
: A character string (default NULL) to specify whether to compute credible intervals as equal-tailed intervals, 'eti' or highest density intervals, 'hdi'. Note that estimate_interval is used to set the global options as follows: options("marginaleffects_posterior_interval" = "eti"), or options("marginaleffects_posterior_interval" = "hdi") The pre-specified global options are restored on exit via the base::on.exit().

dummy_to_factor
: A named list (default NULL) that is used to convert dummy variables into a factor variable. The named elements are factor.dummy, factor.name, and factor.level. The factor.dummy is a vector of character strings that need to be converted to a factor variable whereas the factor.name is a single character string that is used to name the newly created factor variable. The factor.level is used to name the levels of newly created factor. When factor.name is NULL, then the factor name is internally set as 'factor.var'. If factor.level is NULL, then names of factor levels are take from the factor.dummy i.e., the factor levels are assigned same name as factor.dummy. Note that when factor.level is not NULL, its length must be same as the length of the factor.dummy.

verbose
: An optional argument (logical, default FALSE) to indicate whether to print information collected during setting up the object(s).

expose_function
: An optional logical argument to indicate whether to expose Stan functions (default FALSE). Note that if user has already exposed Stan functions during model fit by setting expose_function = TRUE in the bsitar(), then those exposed functions are saved and can be used during post processing of the posterior draws and therefore expose_function is by default set as FALSE in all post processing functions except optimize_model(). For optimize_model(), the default setting is expose_function = NULL. The reason is that each optimized model has different Stan function and therefore it need to be re exposed and saved. The expose_function = NULL implies that the setting for expose_function is taken from the original model fit. Note that expose_function must be set to TRUE when adding fit criteria and/or bayes_R2 during model optimization.

usesavedfuns
: A logical (default NULL) to indicate whether to use the already exposed and saved Stan functions. Depending on whether the user have exposed Stan functions within the bsitar() call via expose_functions argument in the bsitar(), the usesavedfuns is automatically set to TRUE (if expose_functions = TRUE) or FALSE (if expose_functions = FALSE). Therefore, manual setting of usesavedfuns as TRUE/FALSE is rarely needed. This is for internal purposes only and mainly used during the testing of the functions and therefore should not be used by users as it might lead to unreliable estimates.

clearenvfuns
: A logical to indicate whether to clear the exposed function from the environment (TRUE) or not (FALSE). If NULL (default), then clearenvfuns is set as TRUE when usesavedfuns is TRUE, and FALSE if usesavedfuns is FALSE.

funlist
: A list (default NULL) to set function names. This is rarely used because required functions are automatically retrieved internally. A use case of funlist when sigma_formula, sigma_formula_gr, or sigma_formula_gr_str uses a function such as poly(age). See bsitar::bsitar() for details. The funlist

provides list of function names which have been defined externally and are available in the base::globalenv(). Note that for functions that need distance and velocity curves such as plot_curves(..., opt = 'dv'), the funlist must include two functions where first will be used for the distance and second for the velocity.

envir — Environment used for function evaluation. The default is NULL which will set parent.frame() as default environment. Note that since most of post processing functions are based on **brms**, the functions needed for evaluation should be in the .GlobalEnv. Therefore, it is strongly recommended to set envir = globalenv() (or envir = .GlobalEnv). This is particularly true for the derivatives such as velocity curve.

... — Additional arguments passed to the brms::fitted.brmsfit() function. Please see brms::fitted.brmsfit() for details on various options available.

Details

The **marginal_draws()** 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
marginal_draws(model, deriv = 0, re_formula = NA)
```

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

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

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

`optimize_model.bgmfit` *Optimize SITAR model*

Description

Select the best fitting SITAR model that involves choosing the optimum degrees of freedom (`df`) for the natural cubic-spline curve and the appropriate transformations of the predictor `x` and response `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 = c("beta", "sd", "rsd", "sigma", "dpar"),
  transform_beta_coef = c("b", "c", "d"),
  transform_sd_coef = c("b", "c", "d"),
  exclude_default_funs = TRUE,
  add_fit_criteria = NULL,
  add_bayes_R = NULL,
  byresp = FALSE,
  model_name = NULL,
  overwrite = FALSE,
  file = NULL,
  force_save = FALSE,
  digits = 2,
  cores = 1,
  verbose = FALSE,
  expose_function = NULL,
  usesavedfuns = FALSE,
  clearenvfuns = NULL,
  envir = NULL,
  ...
```

```
)

optimize_model(model, ...)
```

Arguments

model	An object of class bgmfit.
newdata	An optional data frame to be used in estimation. If NULL (default), the newdata is retrieved from the model.
optimize_df	A list of integers specifying the degree of freedom (df) values to be optimized. If NULL (default), the df is taken from the original model. For optimization over different df, say for example df 4 and df 5, the corresponding code is optimize_df = list(4,5). For univariate_by and multivariate models, optimize_df can be a single integer (e.g., optimize_df = 4) or a list (e.g., optimize_df = list(4,5)), or a a list of lists. As an example, consider optimization over df 4 and df 5 for the first sub model, and df 5 and df 6 for the second sub model, the corresponding code is optimize_df = list(list(4,5), list(5,6)).
optimize_x	A vector specifying the transformations for the predictor variable (i.e., x). The options available are NULL, 'log', 'sqrt', or their combinations. Note that user need not to enclose these options in a single or double quotes as they are take care of internally. The default setting is to explore all possible combination i.e., optimize_x = list(NULL, log, sqrt). Similar to the optimize_df, user can specify different optimize_x for univariate_by and multivariate sub models.
optimize_y	A vector specifying the transformations of the the response variable (i.e., y). The approach and options available for optimize_y are same as described above for the optimize_x.
transform_prior_class	A character vector (default NULL) specifying the transformations of location-scale based priors (such as normal()) when response variable (i.e., y) is 'log' or 'sqrt' transformed (currently available only for 'log' transformed y). The prior types that can be transformed are 'beta', 'sd', 'rsd', 'sigma' and 'dpar'. Each prior type (i.e., 'beta', 'sd', 'rsd', 'sigma', 'dpar') specified via transform_prior_class is log transformed as follows: log_location = log(location / sqrt(scale^2 / location^2 + 1)), log_scale = sqrt(log(scale^2 / location^2 + 1)), where location and scale are the original parameters supplied by the user and the log_location and log_scale are the equivalent parameters on the log scale. For more details, see a_prior_beta argument in bsitar() function. Note that transform_prior_class is used on an experimental basis and therefore results may not be what user intended. Thus we recommend to explicitly set the desired prior on y scale.
transform_beta_coef	A character vector (default NULL) specifying the transformations of location-scale based priors for specific regression coefficient(s) when response variable (i.e., y) is 'log' or 'sqrt' transformed. The coefficient that could be transformed are 'a', 'b', 'c', 'd' and 's'. The default is transform_beta_coef

= c('b',' b', 'd') which implies that parameters 'a', 'a' and 'a' will be transformed whereas parameter 'a' will be left unchanged because default prior for parameter 'a' is based on outcome y itself (e.g., a_prior_beta = normal(ymean, ysd)) which has be transformed. Note that transform_beta_coef is used on an experimental basis and therefore results may not be what user intended. Thus we recommend to explicitly set the desired prior on y scale.

transform_sd_coef

A character vector (default NULL) specifying the transformations of location-scale based priors for specific group level coefficient(s) when response variable (i.e., y) is 'log' or 'sqrt' transformed. The coefficient that could be transformed are 'a', 'b', 'c', 'd' and 's'. The default is transform_beta_coef = c('b',' b', 'd'). Note that transform_sd_coef is used on an experimental basis and therefore results may not be what user intended. Thus we recommend to explicitly set the desired prior on y scale.

exclude_default_funs

A logical to indicate 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 the optimize_x and optimize_y. From example, if original model is fit with xvar = log and yvar = NULL, then optimize_x is translated into optimize_x = list(NULL, sqrt), and similarly optimize_y is reset as optimize_y = list(log, sqrt).

add_fit_criteria

An optional argument (default NULL) to indicate whether to add fit criteria to the returned model fit. Options available are 'loo' and 'waic'. Please see brms::add_criterion() for details.

add_bayes_R An optional argument (default NULL) to indicate whether to add Bayesian R square to the returned model fit. To estimate and add bayes_R2 to the model fit, the argument add_bayes_R is set as add_bayes_R = 'bayes_R2'.

byresp A logical (default FALSE) to indicate if response wise fit criteria to be calculated. This argument is evaluated only for the multivariate model in which user can select whether to get joint calculation of point wise log likelihood (byresp = FALSE) or response specific (byresp = TRUE). For, univariate_by model, the only option available is to calculate separate point wise log likelihood for each sub-model, 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.

digits — An integer (default 2) to set the decimal argument for the base::round() function.

cores — The number of cores to used in parallel processing (default 1). The argument cores is passed to the brms::add_criterion().

verbose — An optional argument (logical, default FALSE) to indicate whether to print information collected during setting up the object(s).

expose_function — An optional logical argument to indicate whether to expose Stan functions (default FALSE). Note that if user has already exposed Stan functions during model fit by setting expose_function = TRUE in the bsitar(), then those exposed functions are saved and can be used during post processing of the posterior draws and therefore expose_function is by default set as FALSE in all post processing functions except optimize_model(). For optimize_model(), the default setting is expose_function = NULL. The reason is that each optimized model has different Stan function and therefore it need to be re exposed and saved. The expose_function = NULL implies that the setting for expose_function is taken from the original model fit. Note that expose_function must be set to TRUE when adding fit criteria and/or bayes_R2 during model optimization.

usesavedfuns — A logical (default NULL) to indicate whether to use the already exposed and saved Stan functions. Depending on whether the user have exposed Stan functions within the bsitar() call via expose_functions argument in the bsitar(), the usesavedfuns is automatically set to TRUE (if expose_functions = TRUE) or FALSE (if expose_functions = FALSE). Therefore, manual setting of usesavedfuns as TRUE/FALSE is rarely needed. This is for internal purposes only and mainly used during the testing of the functions and therefore should not be used by users as it might lead to unreliable estimates.

clearenvfuns — A logical to indicate whether to clear the exposed function from the environment (TRUE) or not (FALSE). If NULL (default), then clearenvfuns is set as TRUE when usesavedfuns is TRUE, and FALSE if usesavedfuns is FALSE.

envir — Environment used for function evaluation. The default is NULL which will set parent.frame() as default environment. Note that since most of post processing functions are based on **brms**, the functions needed for evaluation should be in the .GlobalEnv. Therefore, it is strongly recommended to set envir = globalenv() (or envir = .GlobalEnv). This is particularly true for the derivatives such as velocity curve.

... — Other arguments passed to update_model.

Value

A list containing the optimized models of class bgmfit, and the the summary statistics if add_fit_criteria and/or add_bayes_R are specified.

Author(s)

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

See Also

brms::add_criterion()

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

# Below example shows dummy call to optimization to save time.
# Note that in case degree of freedom and both  optimize_x and optimize_y are
# NULL (i.e., nothing to optimize), the original model object is returned.
# To explicitly get this information whether model is being optimized 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 <- optimize_model(model,
  optimize_df = NULL,
  optimize_x = NULL,
  optimize_y = NULL,
  verbose = TRUE)
```

---

plot_conditional_effects.bgmfit

Visualize conditional effects of predictor

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 = "posterior_epred",
  transform = NULL,
  resolution = 100,
  select_points = 0,
  too_far = 0,
  prob = 0.95,
  robust = TRUE,
  newdata = NULL,
  ndraws = NULL,
  dpar = NULL,
  draw_ids = NULL,
  levels_id = NULL,
  resp = NULL,
  ipts = 10,
  deriv = 0,
  deriv_model = NULL,
  idata_method = NULL,
  verbose = FALSE,
  dummy_to_factor = NULL,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  funlist = NULL,
  envir = NULL,
  ...
)

plot_conditional_effects(model, ...)
```

Arguments

model	An object of class bgmfit.
effects	An optional character vector naming effects (main effects or interactions) for which to compute conditional plots. Interactions are specified by a : between variable names. If NULL (the default), plots are generated for all main effects and two-way interactions estimated in the model. When specifying effects manually, *all* two-way interactions (including grouping variables) may be plotted even if not originally modeled.
conditions	An optional data.frame containing variable values to condition on. Each effect defined in effects will be plotted separately for each row of conditions. Val-

ues in the cond__ column will be used as titles of the subplots. If cond__ is not given, the row names will be used for this purpose instead. It is recommended to only define a few rows in order to keep the plots clear. See make_conditions for an easy way to define conditions. If NULL (the default), numeric variables will be conditionalized by using their means and factors will get their first level assigned. NA values within factors are interpreted as if all dummy variables of this factor are zero. This allows, for instance, to make predictions of the grand mean when using sum coding.

int_conditions An optional named list whose elements are vectors of values of the variables specified in effects. At these values, predictions are evaluated. The names of int_conditions have to match the variable names exactly. Additionally, the elements of the vectors may be named themselves, in which case their names appear as labels for the conditions in the plots. Instead of vectors, functions returning vectors may be passed and are applied on the original values of the corresponding variable. If NULL (the default), predictions are evaluated at the $mean$ and at $mean + / - sd$ for numeric predictors and at all categories for factor-like predictors.

re_formula A formula containing group-level effects to be considered in the conditional predictions. If NULL, include all group-level effects; if NA (default), include no group-level effects.

spaghetti Logical. Indicates if predictions should be visualized via spaghetti plots. Only applied for numeric predictors. If TRUE, it is recommended to set argument ndraws to a relatively small value (e.g., 100) in order to reduce computation time.

surface Logical. Indicates if interactions or two-dimensional smooths should be visualized as a surface. Defaults to FALSE. The surface type can be controlled via argument stype of the related plotting method.

categorical Logical. Indicates if effects of categorical or ordinal models should be shown in terms of probabilities of response categories. Defaults to FALSE.

ordinal (Deprecated) Please use argument categorical. Logical. Indicates if effects in ordinal models should be visualized as a raster with the response categories on the y-axis. Defaults to FALSE.

method Method used to obtain predictions. Can be set to "posterior_epred" (the default), "posterior_predict", or "posterior_linpred". For more details, see the respective function documentations.

transform A function or a character string naming a function to be applied on the predicted responses before summary statistics are computed. Only allowed if method = "posterior_predict".

resolution Number of support points used to generate the plots. Higher resolution leads to smoother plots. Defaults to 100. If surface is TRUE, this implies 10000 support points for interaction terms, so it might be necessary to reduce resolution when only few RAM is available.

select_points Positive number. Only relevant if points or rug are set to TRUE: Actual data points of numeric variables that are too far away from the values specified in conditions can be excluded from the plot. Values are scaled into the unit interval and then points more than select_points from the values in conditions are excluded. By default, all points are used.

too_far — Positive number. For surface plots only: Grid points that are too far away from the actual data points can be excluded from the plot. too_far determines what is too far. The grid is scaled into the unit square and then grid points more than too_far from the predictor variables are excluded. By default, all grid points are used. Ignored for non-surface plots.

prob — A value between 0 and 1 indicating the desired probability to be covered by the uncertainty intervals. The default is 0.95.

robust — If TRUE (the default) the median is used as the measure of central tendency. If FALSE the mean is used instead.

newdata — An optional data frame to be used in estimation. If NULL (default), the newdata is retrieved from the model.

ndraws — A positive integer indicating the number of posterior draws to be used in estimation. If NULL (default), all draws are used.

dpar — Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.

draw_ids — An integer indicating the specific posterior draw(s) to be used in estimation (default NULL).

levels_id — An optional argument to specify the ids for hierarchical model (default NULL). It is used only when model is applied to the data with 3 or more levels of hierarchy. For a two level model, the levels_id is automatically inferred from the model fit. Even for 3 or higher level model, the levels_id is inferred from the model fit but under the assumption that hierarchy is specified from lowest to upper most level i.e, id followed by study where id is nested within the study Note that it is not guaranteed that the levels_id is sorted correctly, and therefore 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 response variable when processing posterior draws for the univariate_by and multivariate models. See bsitar() for details on univariate_by and multivariate models

ipts — An integer to set the length of the predictor variable to get a smooth velocity curve. The NULL will return original values whereas an integer such as ipts = 10 (default) will interpolate the predictor. It is important to note that these interpolations do not alter the range of predictor when calculating population average and/or the individual specific growth curves.

deriv — An integer to indicate whether to estimate distance curve or its derivative (i.e., velocity curve). The deriv = 0 (default) is for the distance curve whereas deriv = 1 for the velocity curve.

deriv_model — A logical to specify whether to estimate velocity curve from the derivative function, or the differentiation of the distance curve. The argument deriv_model is set to TRUE for those functions which need velocity curve such as growthparameters() and plot_curves(), and NULL for functions which explicitly use the distance curve (i.e., fitted values) such as loo_validation() and plot_ppc().

idata_method — A character string to indicate the interpolation method. The number of of interpolation points is set up the ipts argument. Options available for idata_method are *method 1* (specified as 'm1') and *method 2* (specified as 'm2'). The *method 1* ('m1') is adapted from the the **iapvbs** package and is documented here https:

//rdrr.io/github/Zhiqiangcao/iapvbs/src/R/exdata.R whereas *method 2* ('m2') is based on the **JMbayes** package as documented here https://github.com/drizopoulos/JMbayes/blob/master/R/dynPred_lme.R. The 'm1' method works by internally constructing the data frame based on the model configuration whereas the method 'm2' uses the exact data frame used in model fit and can be accessed via fit$data. If idata_method = NULL, default, then method 'm2' is automatically set. Note that method 'm1' might fail in some cases when model involves covariates particularly when model is fit as univariate_by. Therefore, it is advised to switch to method 'm2' in case 'm1' results in error.

verbose An optional argument (logical, default FALSE) to indicate whether to print information collected during setting up the object(s).

dummy_to_factor
A named list (default NULL) that is used to convert dummy variables into a factor variable. The named elements are factor.dummy, factor.name, and factor.level. The factor.dummy is a vector of character strings that need to be converted to a factor variable whereas the factor.name is a single character string that is used to name the newly created factor variable. The factor.level is used to name the levels of newly created factor. When factor.name is NULL, then the factor name is internally set as 'factor.var'. If factor.level is NULL, then names of factor levels are take from the factor.dummy i.e., the factor levels are assigned same name as factor.dummy. Note that when factor.level is not NULL, its length must be same as the length of the factor.dummy.

expose_function
An optional logical argument to indicate whether to expose Stan functions (default FALSE). Note that if user has already exposed Stan functions during model fit by setting expose_function = TRUE in the bsitar(), then those exposed functions are saved and can be used during post processing of the posterior draws and therefore expose_function is by default set as FALSE in all post processing functions except optimize_model(). For optimize_model(), the default setting is expose_function = NULL. The reason is that each optimized model has different Stan function and therefore it need to be re exposed and saved. The expose_function = NULL implies that the setting for expose_function is taken from the original model fit. Note that expose_function must be set to TRUE when adding fit criteria and/or bayes_R2 during model optimization.

usesavedfuns A logical (default NULL) to indicate whether to use the already exposed and saved Stan functions. Depending on whether the user have exposed Stan functions within the bsitar() call via expose_functions argument in the bsitar(), the usesavedfuns is automatically set to TRUE (if expose_functions = TRUE) or FALSE (if expose_functions = FALSE). Therefore, manual setting of usesavedfuns as TRUE/FALSE is rarely needed. This is for internal purposes only and mainly used during the testing of the functions and therefore should not be used by users as it might lead to unreliable estimates.

clearenvfuns A logical to indicate whether to clear the exposed function from the environment (TRUE) or not (FALSE). If NULL (default), then clearenvfuns is set as TRUE when usesavedfuns is TRUE, and FALSE if usesavedfuns is FALSE.

funlist A list (default NULL) to set function names. This is rarely used because required functions are automatically retrieved internally. A use case of funlist

when sigma_formula, sigma_formula_gr, or sigma_formula_gr_str uses a function such as poly(age). See bsitar::bsitar() for details. The funlist provides list of function names which have been defined externally and are available in the base::globalenv(). Note that for functions that need distance and velocity curves such as plot_curves(..., opt = 'dv'), the funlist must include two functions where first will be used for the distance and second for the velocity.

envir — Environment used for function evaluation. The default is NULL which will set parent.frame() as default environment. Note that since most of post processing functions are based on **brms**, the functions needed for evaluation should be in the .GlobalEnv. Therefore, it is strongly recommended to set envir = globalenv() (or envir = .GlobalEnv). This is particularly true for the derivatives such as velocity curve.

... — 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 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 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, user can specify all the arguments which are available in the brms::conditional_effects().

Value

An object of class 'brms_conditional_effects' which is a named list with one data.frame per effect containing all information required to generate conditional effects plots. See brms::conditional_effects for details.

Author(s)

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

See Also

brms::conditional_effects()

Examples

```
# Fit Bayesian SITAR model

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

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

model <- berkeley_exfit

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


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

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

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

`plot_curves.bgmfit` *Plot growth curves*

Description

The **plot_curves()** provides visualization of six different types of growth curves that are plotted by using the **ggplot2** package. The **plot_curves()** also allows users to make their own detailed plots from the data returned as a `data.frame`. Note that an alternative approach is to use `marginal_draws()` function that not only allows estimation of adjusted curves but also makes it possible to compare them across groups by 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,
  levels_id = NULL,
  avg_reffects = NULL,
  ipts = 10,
  deriv_model = TRUE,
  xrange = NULL,
  xrange_search = NULL,
  takeoff = FALSE,
  trough = FALSE,
  acgv = FALSE,
  acgv_velocity = 0.1,
  seed = 123,
  estimation_method = "fitted",
  allow_new_levels = FALSE,
  sample_new_levels = "uncertainty",
  incl_autocor = TRUE,
  robust = FALSE,
  transform = NULL,
  future = FALSE,
  future_session = "multisession",
  cores = NULL,
  trim = 0,
  layout = "single",
  linecolor = NULL,
  linecolor1 = NULL,
  linecolor2 = NULL,
  label.x = NULL,
  label.y = NULL,
  legendpos = NULL,
  linetype.apv = NULL,
  linewidth.main = NULL,
  linewidth.apv = NULL,
  linetype.groupby = NA,
  color.groupby = NA,
  band.alpha = NULL,
  show_age_takeoff = TRUE,
  show_age_peak = TRUE,
  show_age_cessation = TRUE,
  show_vel_takeoff = FALSE,
  show_vel_peak = FALSE,
  show_vel_cessation = FALSE,
  returndata = FALSE,
  returndata_add_parms = FALSE,
  parms_eval = FALSE,
  idata_method = NULL,
  parms_method = "getPeak",
  verbose = FALSE,
```

```
  fullframe = NULL,
  dummy_to_factor = NULL,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  funlist = NULL,
  envir = NULL,
  ...
)

plot_curves(model, ...)
```

Arguments

model	An object of class bgmfit.
opt	A character string containing letter(s) corresponding to the following plotting options: 'd' for population average distance curve, 'v' for population average velocity curve, 'D' for individual-specific distance curves, 'V' for individual-specific velocity curves, 'u' for unadjusted individual-specific distance curves, and 'a' for adjusted individual-specific distance curves (adjusted for the random effects). Options 'd' and 'D' can not be specified simultaneously. Likewise, Options 'v' and 'V' can not be specified simultaneously. All other combinations are allowed. For example, dvau', Dvau', dVau', DVau', or dvau'.
apv	An optional logical (default FALSE) specifying whether or not to calculate and plot the age at peak velocity (APGV) when opt) includes 'v' or 'V'.
bands	A character string containing letter(s), or NULL (default) to indicate if CI bands to be plotted around the distance and velocity curves (and also the APGV). If NULL, no band plotted. Alternatively, user can specify a string with any one of the following or their combination(s): 'd' for band around the distance curve, 'v} for band around the velocity curve, and \code{'p for band around the the vertical line denoting the APGV parameter. The 'dvp' will include CI bands for distance and velocity curves, and the APGV.
conf	A numeric value (default 0.95) to be used to compute the CI and hence the width of the bands. See growthparameters() for further details.
resp	A character string (default NULL) to specify response variable when processing posterior draws for the 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 be used in estimation. If NULL (default), all draws are used.
draw_ids	An integer indicating the specific posterior draw(s) to be used in estimation (default NULL).
newdata	An optional data frame to be used in estimation. If NULL (default), the newdata is retrieved from the model.

summary — A logical indicating whether only the estimate should be computed (TRUE), or estimate along with SE and CI should be returned (FALSE, default). Setting summary as FALSE will increase the computation time. Note that summary = FALSE is must to get the correct estimates when re_formula = NULL.

digits — An integer (default 2) to set the decimal argument for the base::round() function.

re_formula — Option to indicate whether or not to include the individual/group-level effects in the estimation. When NA (default), the individual-level effects are excluded and therefore population average growth parameters are computed. When NULL, individual-level effects are included in the computation and hence the growth parameters estimates returned are individual-specific. In both situations, (i.e,, NA or NULL), continuous and factor covariate(s) are appropriately included in the estimation. The continuous covariates by default are set to their means (see numeric_cov_at for details) whereas factor covariates are left unaltered thereby allowing estimation of covariate specific population average and individual-specific growth parameter.

numeric_cov_at — An optional (named list) argument to specify the value of continuous covariate(s). The default NULL option set the continuous covariate(s) at 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 varibale 'xx' at 2. The argument numeric_cov_at is ignored when no continuous covariate is included in the model.

aux_variables — An optional argument to specify the variables to be passed to the ipts argument. This is useful when fitting location scale models and the measurement error models.

levels_id — An optional argument to specify the ids for hierarchical model (default NULL). It is used only when model is applied to the data with 3 or more levels of hierarchy. For a two level model, the levels_id is automatically inferred from the model fit. Even for 3 or higher level model, the levels_id is inferred from the model fit but under the assumption that hierarchy is specified from lowest to upper most level i.e, id followed by study where id is nested within the study Note that it is not guaranteed that the levels_id is sorted correctly, and therefore 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 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) such as avg_reffects = list(feby = 'study', reby = NULL, over = 'age').

ipts — An integer to set the length of the predictor variable to get a smooth velocity curve. The NULL will return original values whereas an integer such as ipts = 10 (default) will interpolate the predictor. It is important to note that these interpolations do not alter the range of predictor when calculating population average and/or the individual specific growth curves.

deriv_model — A logical to specify whether to estimate velocity curve from the derivative function, or the differentiation of the distance curve. The argument deriv_model is

set to TRUE for those functions which need velocity curve such as growthparameters() and plot_curves(), and NULL for functions which explicitly use the distance curve (i.e., fitted values) such as loo_validation() and plot_ppc().

xrange
: An integer to set the predictor range (i.e., age) when executing the interpolation via ipts. The default NULL sets the individual specific predictor range whereas code xrange = 1 sets identical range for individuals within the same higher grouping variable (e.g., study). Code xrange = 2 sets the identical range across the entire sample. Lastly, a paired numeric values can be supplied e.g., xrange = c(6, 20) to set the range within those values.

xrange_search
: A vector of length two, or a character string 'range' to set the range of predictor variable (x) within which growth parameters are searched. This is useful when there is more than one peak and user wants to summarize peak within a given range of the x variable. Default xrange_search = NULL.

takeoff
: A logical (default FALSE) to indicate whether or not to calculate the age at takeoff velocity (ATGV) and the takeoff growth velocity (TGV) parameters.

trough
: A logical (default FALSE) to indicate whether or not to calculate the age at cessation of growth velocity (ACGV) and the cessation of growth velocity (CGV) parameters.

acgv
: A logical (default FALSE) to indicate whether or not to calculate the age at cessation of growth velocity from the velocity curve. If TRUE, age at cessation of growth velocity (ACGV) and the cessation growth velocity (CGV) are calculated based on the percentage of the peak growth velocity as defined by the acgv_velocity argument (see below). The acgv_velocity is typically set at 10 percent of the peak growth velocity. The ACGV and CGV are calculated along with the the uncertainty (SE and CI) around the ACGV and CGV parameters.

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

seed
: An integer (default 123) that is passed to the estimation method.

estimation_method
: A character string to specify the estimation method when calculating the velocity from the posterior draws. The 'fitted' method internally calls the fitted_draws() whereas the option predict calls the predict_draws(). See brms::fitted.brmsfit() and brms::predict.brmsfit() for derails.

allow_new_levels
: A flag indicating if new levels of group-level effects are allowed (defaults to FALSE). Only relevant if newdata is provided.

sample_new_levels
: Indicates how to sample new levels for grouping factors specified in re_formula. This argument is only relevant if newdata is provided and allow_new_levels is set to TRUE. If "uncertainty" (default), each posterior sample for a new level is drawn from the posterior draws of a randomly chosen existing level. Each posterior sample for a new level may be drawn from a different existing level such that the resulting set of new posterior draws represents the variation across existing levels. If "gaussian", sample new levels from the (multivariate) normal distribution implied by the group-level standard deviations and correlations.

This options may be useful for conducting Bayesian power analysis or predicting new levels in situations where relatively few levels where observed in the old_data. If "old_levels", directly sample new levels from the existing levels, where a new level is assigned all of the posterior draws of the same (randomly chosen) existing level.

incl_autocor A flag indicating if correlation structures originally specified via autocor should be included in the predictions. Defaults to TRUE.

robust A logical to specify the summarize options. 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. Ignored if summary is FALSE.

transform A function applied to individual draws from the posterior distribution, before computing summaries. The argument transform is based on the marginaleffects::predictions(). This should not be confused with the transform from the brms::posterior_predict() which is now deprecated.

future A logical (default FALSE) to specify whether or not to perform parallel computations. If set to TRUE, the future.apply::future_sapply() function is used to summarize draws.

future_session A character string to set the session type when future = TRUE. The 'multisession' (default) options sets the multisession whereas the 'multicore' sets the multicore session. Note that option 'multicore' is not supported on Windows systems. For more details, see future.apply::future_sapply().

cores Number of cores to be used when running the parallel computations (if future = TRUE). On non-Windows systems this argument can be set globally via the mc.cores option. For the default NULL option, the number of cores are set automatically by calling the future::availableCores(). The number of cores used are the maximum number of cores avaialble minus one, i.e., future::availableCores() - 1.

trim A number (default 0) of long line segments to be excluded from plot with option 'u' or 'a'. See sitar::plot.sitar for details.

layout A character string defining the layout structure of the plot. A 'single' (default) layout provides overlaid distance and velocity curves on a single plot when opt includes 'dv', 'Dv', 'dV' or 'DV' options. Similarly, when opt includes 'au', the adjusted and unadjusted curves are plotted as a single plot. When opt is a single letter (e.g., 'd'. 'v' 'D', 'V', 'a', 'u'), the 'single' optiion is ignored. The alternative layout option, the 'facet' uses the facet_wrap from the ggplot2. to map and draw plot when opt include two or more letters.

linecolor The color of line used when layout is 'facet'. The default is NULL which internally set the linecolor as 'grey50'.

linecolor1 The color of first line when layout is 'single'. For example, for opt = 'dv', the color of distance line is controlled by the linecolor1. Default NULL will internally set linecolor1 as 'orange2'.

linecolor2 The color of second line when layout is 'single'. For example, for opt = 'dv', the color of velocity line is controlled by the linecolor2. Default NULL sets the color 'green4' for linecolor2.

label.x An optional character string to label the x axis. When NULL (default), the x axis label is taken from the predictor (e.g., age).

label.y An optional character string to label the y axis. When NULL (default), the y axis label is taken from the type of plot (e.g., distance, velocity etc.). Note that when layout option is 'facet', then y axis label is removed and instead the same label is used as a title.

legendpos An optional character string to specify the position of legends. When NULL (default), the legend position is set as 'bottom' for distance and velocity curves with 'single' layout option for the population average curves, and 'none' for the individual specific curves. The 'none' suppress all legends that helps in avoiding printing legends for each individual.

linetype.apv An optional character string to specify the type of the vertical line drawn to mark the APGV. Default NULL sets the linetype as dotted.

linewidth.main An optional character string to specify the width of the the line for the distance and velocity curves. The default NULL will set it as 0.35.

linewidth.apv An optional character string to specify the width of the the vertical line drawn to mark the APGV. The default NULL will set it as 0.25.

linetype.groupby
An optional argument to specify the line type for the distance and velocity curves when drawing plots for a model that includes factor covariate(s) or when visualising individual specific distance/velocity curves (default NA). Setting it to NULL will automatically sets the linetype for each factor level or individual This will also add legends for the factor level covariate or individuals whereas NA will set a 'solid' line type and suppress legends. It is recommended to keep the default NULL option when plotting population average curves for when model included factor covariates because this would appropriately set the legends otherwise it is difficult to differentiate which curve belongs to which level of factor. For individual specific curves, the line type can be set to NULL when the number of individuals is small. However, when the number of individuals is large, NA is a better choice which prevents printing a large number of legends for each individual.

color.groupby An optional argument to specify the line color for distance and velocity curves when drawing plots for a model that includes factor covariate(s), or when visualising individual specific distance/velocity curves (default NA). Setting it to NULL will automatically sets the line color for each factor level or individual. This will also add legends for the factor level covariate or individuals. However, setting it as NA will set a 'solid' line type and suppress legends. It is recommended to keep the default NULL option when plotting population average curves for factor covariates because this would appropriately set the legends otherwise it is difficult to differentiate which curve belongs to which level of the factor. For individual specific curves, the line color can be set to NULL when the number of individuals is small. However, when the number of individuals is large, NA is a better choice which prevents printing a large number of legends for each individual.

band.alpha An optional numeric value to specify the transparency of the CI band(s) around the distance curve, velocity curve and the line indicating the APGV. The default NULL will set this value to 0.4.

show_age_takeoff
: A logical (default TRUE) to indicate whether to display the ATGV line(s) on the plot.

show_age_peak
: A logical (default TRUE) to indicate whether to display the APGV line(s) on the plot.

show_age_cessation
: A logical (default TRUE) to indicate whether to display the ACGV line(s) on the plot.

show_vel_takeoff
: A logical (default FALSE) to indicate whether to display the TGV line(s) on the plot.

show_vel_peak
: A logical (default FALSE) to indicate whether to display the PGV line(s) on the plot.

show_vel_cessation
: A logical (default FALSE) to indicate whether to display the CGV line(s) on the plot.

returndata
: A logical (default FALSE) indicating whether to plot the data or return the data. If TRUE, the data is returned as a data.frame.

returndata_add_parms
: A logical (default FALSE) indicating whether add growth parameters to the returndata. The returndata_add_parms is ignored when returndata = FALSE. If TRUE, the growth parameters such as APGV and PGV are added to the returned data.frame. Note that growth parameters are estimated only when 'opt' argument include either 'v' or 'V' option and the argument 'apv' is set to TRUE. If any of these conditions are missing, then returndata_add_parms will ignored ignored.

parms_eval
: A logical to specify whether or not to get growth parameters on the fly. This is for internal use only and mainly needed for compatibility across internal functions.

idata_method
: A character string to indicate the interpolation method. The number of of interpolation points is set up the ipts argument. Options available for idata_method are *method 1* (specified as 'm1') and *method 2* (specified as 'm2'). The *method 1* ('m1') is adapted from the the **iapvbs** package and is documented here https://rdrr.io/github/Zhiqiangcao/iapvbs/src/R/exdata.R whereas *method 2* ('m2') is based on the **JMbayes** package as documented here https://github.com/drizopoulos/JMbayes/blob/master/R/dynPred_lme.R. The 'm1' method works by internally constructing the data frame based on the model configuration whereas the method 'm2' uses the exact data frame used in model fit and can be accessed via fit$data. If idata_method = NULL, default, then method 'm2' is automatically set. Note that method 'm1' might fail in some cases when model involves covariates particularly when model is fit as univariate_by. Therefore, it is advised to switch to method 'm2' in case 'm1' results in error.

parms_method
: A character to specify the method used to when evaluating parms_eval. The default is getPeak which uses the sitar::getPeak() function from the sitar package. The alternative option is findpeaks that uses this findpeaks from the pracma package. This is for internal use only and mainly needed for compatibility across internal functions.

verbose An optional argument (logical, default FALSE) to indicate whether to print information collected during setting up the object(s).

fullframe A logical to indicate whether to return fullframe object in which newdata is bind to the summary estimates. Note that fullframe can not be combined with summary = FALSE. Furthermore, fullframe can only be used when idata_method = 'm2'. A particular use case is when fitting univariate_by model. The fullframe is mainly for internal use only.

dummy_to_factor
A named list (default NULL) that is used to convert dummy variables into a factor variable. The named elements are factor.dummy, factor.name, and factor.level. The factor.dummy is a vector of character strings that need to be converted to a factor variable whereas the factor.name is a single character string that is used to name the newly created factor variable. The factor.level is used to name the levels of newly created factor. When factor.name is NULL, then the factor name is internally set as 'factor.var'. If factor.level is NULL, then names of factor levels are take from the factor.dummy i.e., the factor levels are assigned same name as factor.dummy. Note that when factor.level is not NULL, its length must be same as the length of the factor.dummy.

expose_function
An optional logical argument to indicate whether to expose Stan functions (default FALSE). Note that if user has already exposed Stan functions during model fit by setting expose_function = TRUE in the bsitar(), then those exposed functions are saved and can be used during post processing of the posterior draws and therefore expose_function is by default set as FALSE in all post processing functions except optimize_model(). For optimize_model(), the default setting is expose_function = NULL. The reason is that each optimized model has different Stan function and therefore it need to be re exposed and saved. The expose_function = NULL implies that the setting for expose_function is taken from the original model fit. Note that expose_function must be set to TRUE when adding fit criteria and/or bayes_R2 during model optimization.

usesavedfuns A logical (default NULL) to indicate whether to use the already exposed and saved Stan functions. Depending on whether the user have exposed Stan functions within the bsitar() call via expose_functions argument in the bsitar(), the usesavedfuns is automatically set to TRUE (if expose_functions = TRUE) or FALSE (if expose_functions = FALSE). Therefore, manual setting of usesavedfuns as TRUE/FALSE is rarely needed. This is for internal purposes only and mainly used during the testing of the functions and therefore should not be used by users as it might lead to unreliable estimates.

clearenvfuns A logical to indicate whether to clear the exposed function from the environment (TRUE) or not (FALSE). If NULL (default), then clearenvfuns is set as TRUE when usesavedfuns is TRUE, and FALSE if usesavedfuns is FALSE.

funlist A list (default NULL) to set function names. This is rarely used because required functions are automatically retrieved internally. A use case of funlist when sigma_formula, sigma_formula_gr, or sigma_formula_gr_str uses a function such as poly(age). See bsitar::bsitar() for details. The funlist provides list of function names which have been defined externally and are available in the base::globalenv(). Note that for functions that need distance and

velocity curves such as `plot_curves(..., opt = 'dv')`, the `funlist` must include two functions where first will be used for the distance and second for the velocity.

`envir` Environment used for function evaluation. The default is `NULL` which will set `parent.frame()` as default environment. Note that since most of post processing functions are based on **brms**, the functions needed for evaluation should be in the `.GlobalEnv`. Therefore, it is strongly recommended to set `envir = globalenv()` (or `envir = .GlobalEnv`). This is particularly true for the derivatives such as velocity curve.

`...` Further arguments passed to `brms::fitted.brmsfit()` and `brms::predict()` functions.

Details

The **plot_curves()** is a generic function that allows visualization of following six curves: population average distance curve, population average velocity curve, individual-specific distance curves, individual-specific velocity curves, unadjusted individual growth curves (i.e, observed growth curves), and the adjusted individual growth curves (adjusted for the model estimated random effects). The **plot_curves()** internally calls the `growthparameters()` function to estimate and summaries the distance and velocity curves and to estimate growth parameters such as the age at peak growth velocity (APGV). The **plot_curves()** in turn calls the `fitted_draws()` or the `predict_draws()` functions to make inference from the posterior draws. Thus, **plot_curves()** allows plotting fitted or predicted curves. See `fitted_draws()` and `predict_draws()` for details on these functions and 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 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 and velocity curves with default options
plot_curves(model, opt = 'dv')


# Individual-specific distance and velocity curves with default options
# Note that legendpos = 'none' will suppress the legend positions. This
# suppression is useful when plotting individual-specific curves

plot_curves(model, opt = 'DV')

# Population average distance and velocity curves with APGV

plot_curves(model, opt = 'dv', apv = TRUE)

# Individual-specific distance and velocity curves with APGV

plot_curves(model, opt = 'DV', apv = TRUE)

# Population average distance curve, velocity curve, and APGV with CI bands
# To construct CI bands, growth parameters are first calculated for each
# posterior draw and then summarized across draws. Therefore,summary
# option must be set to FALSE

plot_curves(model, opt = 'dv', apv = TRUE, bands = 'dvp', summary = FALSE)

# Adjusted and unadjusted individual curves
# Note ipts = NULL (i.e., no interpolation of predictor (i.e., age) to plot a
# smooth curve). This is because it does not a make sense to interploate data
# when estimating adjusted curves. Also, layout = 'facet' (and not default
# layout = 'single') is used for the ease of visualizing the plotted
# adjusted and unadjusted individual curves. However, these lines can be
# superimposed on each other by setting the set layout = 'single'.
# For other plots shown above, layout can be set as 'single' or 'facet'

# Separate plots for adjusted and unadjusted curves (layout = 'facet')
plot_curves(model, opt = 'au', ipts = NULL, layout = 'facet')

# Superimposed adjusted and unadjusted curves (layout = 'single')
plot_curves(model, opt = 'au', ipts = NULL, layout = 'single')
```

`plot_ppc.bgmfit` *Perform posterior predictive distribution checks*

Description

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

Usage

```
## S3 method for class 'bgmfit'
plot_ppc(
  model,
  type,
  ndraws = NULL,
  dpar = NULL,
  draw_ids = NULL,
  prefix = c("ppc", "ppd"),
  group = NULL,
  x = NULL,
  newdata = NULL,
  resp = NULL,
  size = 0.25,
  alpha = 0.7,
  trim = FALSE,
  bw = "nrd0",
  adjust = 1,
  kernel = "gaussian",
  n_dens = 1024,
  pad = TRUE,
  discrete = FALSE,
  binwidth = NULL,
  bins = NULL,
  breaks = NULL,
  freq = TRUE,
  y_draw = c("violin", "points", "both"),
  y_size = 1,
  y_alpha = 1,
  y_jitter = 0.1,
  verbose = FALSE,
  deriv_model = NULL,
  dummy_to_factor = NULL,
  expose_function = FALSE,
  usesavedfuns = NULL,
  clearenvfuns = NULL,
  envir = NULL,
  ...
)

plot_ppc(model, ...)
```

Arguments

model	An object of class bgmfit.
type	Type of the ppc plot as given by a character string. See PPC for an overview of currently supported types. You may also use an invalid type (e.g. type = "xyz") to get a list of supported types in the resulting error message.

ndraws	A positive integer indicating the number of posterior draws to be used in estimation. If NULL (default), all draws are used.
dpar	Optional name of a predicted distributional parameter. If specified, expected predictions of this parameters are returned.
draw_ids	An integer indicating the specific posterior draw(s) to be used in estimation (default NULL).
prefix	The prefix of the **bayesplot** function to be applied. Either ‘"ppc"‘ (posterior predictive check; the default) or ‘"ppd"‘ (posterior predictive distribution), the latter being the same as the former except that the observed data is not shown for ‘"ppd"‘.
group	Optional name of a factor variable in the model by which to stratify the ppc plot. This argument is required for ppc *_grouped types and ignored otherwise.
x	Optional name of a variable in the model. Only used for ppc types having an x argument and ignored otherwise.
newdata	An optional data frame to be used in estimation. If NULL (default), the newdata is retrieved from the model.
resp	A character string (default NULL) to specify response variable when processing posterior draws for the univariate_by and multivariate models. See bsitar() for details on univariate_by and multivariate models
size, alpha	Passed to the appropriate geom to control the appearance of the predictive distributions.
trim	A logical scalar passed to ggplot2::geom_density().
bw, adjust, kernel, n_dens	Optional arguments passed to stats::density() to override default kernel density estimation parameters. n_dens defaults to 1024.
pad	A logical scalar passed to ggplot2::stat_ecdf().
discrete	For ppc_ecdf_overlay(), should the data be treated as discrete? The default is FALSE, in which case geom="line" is passed to ggplot2::stat_ecdf(). If discrete is set to TRUE then geom="step" is used.
binwidth	Passed to ggplot2::geom_histogram() to override the default binwidth.
bins	Passed to ggplot2::geom_histogram() to override the default binwidth.
breaks	Passed to ggplot2::geom_histogram() as an alternative to binwidth.
freq	For histograms, freq=TRUE (the default) puts count on the y-axis. Setting freq=FALSE puts density on the y-axis. (For many plots the y-axis text is off by default. To view the count or density labels on the y-axis see the yaxis_text() convenience function.)
y_draw	For ppc_violin_grouped(), a string specifying how to draw y: "violin" (default), "points" (jittered points), or "both".
y_jitter, y_size, y_alpha	For ppc_violin_grouped(), if y_draw is "points" or "both" then y_size, y_alpha, and y_jitter are passed to to the size, alpha, and width arguments of ggplot2::geom_jitter() to control the appearance of y points. The default of y_jitter=NULL will let **ggplot2** determine the amount of jitter.

verbose — An optional argument (logical, default FALSE) to indicate whether to print information collected during setting up the object(s).

deriv_model — A logical to specify whether to estimate velocity curve from the derivative function, or the differentiation of the distance curve. The argument deriv_model is set to TRUE for those functions which need velocity curve such as growthparameters() and plot_curves(), and NULL for functions which explicitly use the distance curve (i.e., fitted values) such as loo_validation() and plot_ppc().

dummy_to_factor — A named list (default NULL) that is used to convert dummy variables into a factor variable. The named elements are factor.dummy, factor.name, and factor.level. The factor.dummy is a vector of character strings that need to be converted to a factor variable whereas the factor.name is a single character string that is used to name the newly created factor variable. The factor.level is used to name the levels of newly created factor. When factor.name is NULL, then the factor name is internally set as 'factor.var'. If factor.level is NULL, then names of factor levels are take from the factor.dummy i.e., the factor levels are assigned same name as factor.dummy. Note that when factor.level is not NULL, its length must be same as the length of the factor.dummy.

expose_function — An optional logical argument to indicate whether to expose Stan functions (default FALSE). Note that if user has already exposed Stan functions during model fit by setting expose_function = TRUE in the bsitar(), then those exposed functions are saved and can be used during post processing of the posterior draws and therefore expose_function is by default set as FALSE in all post processing functions except optimize_model(). For optimize_model(), the default setting is expose_function = NULL. The reason is that each optimized model has different Stan function and therefore it need to be re exposed and saved. The expose_function = NULL implies that the setting for expose_function is taken from the original model fit. Note that expose_function must be set to TRUE when adding fit criteria and/or bayes_R2 during model optimization.

usesavedfuns — A logical (default NULL) to indicate whether to use the already exposed and saved Stan functions. Depending on whether the user have exposed Stan functions within the bsitar() call via expose_functions argument in the bsitar(), the usesavedfuns is automatically set to TRUE (if expose_functions = TRUE) or FALSE (if expose_functions = FALSE). Therefore, manual setting of usesavedfuns as TRUE/FALSE is rarely needed. This is for internal purposes only and mainly used during the testing of the functions and therefore should not be used by users as it might lead to unreliable estimates.

clearenvfuns — A logical to indicate whether to clear the exposed function from the environment (TRUE) or not (FALSE). If NULL (default), then clearenvfuns is set as TRUE when usesavedfuns is TRUE, and FALSE if usesavedfuns is FALSE.

envir — Environment used for function evaluation. The default is NULL which will set parent.frame() as default environment. Note that since most of post processing functions are based on **brms**, the functions needed for evaluation should be in the .GlobalEnv. Therefore, it is strongly recommended to set envir = globalenv() (or envir = .GlobalEnv). This is particularly true for the derivatives such as velocity curve.

... Additional arguments passed to the brms::pp_check.brmsfit() function. Please see brms::pp_check.brmsfit() for details.

Details

The **plot_ppc()** is a wrapper around the brms::pp_check().

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 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

plot_ppc(model, ndraws = 100)
```

---

predict_draws.bgmfit *Predicted values from the posterior predictive distribution*

Description

The **predict_draws()** is a wrapper around the brms::predict.brmsfit() function to obtain predicted values (and their summary) from the posterior distribution. See brms::predict.brmsfit() for details.

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,
  ipts = 10,
  deriv = 0,
  deriv_model = TRUE,
  summary = TRUE,
  robust = FALSE,
  transform = 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,
  envir = NULL,
  ...
)

predict_draws(model, ...)
```

Arguments

model	An object of class bgmfit.
newdata	An optional data frame to be used in estimation. If NULL (default), the newdata is retrieved from the model.
resp	A character string (default NULL) to specify response variable when processing posterior draws for the 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 be used in estimation. If NULL (default), all draws are used.
draw_ids	An integer indicating the specific posterior draw(s) to be used in estimation (default NULL).

re_formula Option to indicate whether or not to include the individual/group-level effects in the estimation. When NA (default), the individual-level effects are excluded and therefore population average growth parameters are computed. When NULL, individual-level effects are included in the computation and hence the growth parameters estimates returned are individual-specific. In both situations, (i.e,, NA or NULL), continuous and factor covariate(s) are appropriately included in the estimation. The continuous covariates by default are set to their means (see numeric_cov_at for details) whereas factor covariates are left unaltered thereby allowing estimation of covariate specific population average and individual-specific growth parameter.

allow_new_levels
A flag indicating if new levels of group-level effects are allowed (defaults to FALSE). Only relevant if newdata is provided.

sample_new_levels
Indicates how to sample new levels for grouping factors specified in re_formula. This argument is only relevant if newdata is provided and allow_new_levels is set to TRUE. If "uncertainty" (default), each posterior sample for a new level is drawn from the posterior draws of a randomly chosen existing level. Each posterior sample for a new level may be drawn from a different existing level such that the resulting set of new posterior draws represents the variation across existing levels. If "gaussian", sample new levels from the (multivariate) normal distribution implied by the group-level standard deviations and correlations. This options may be useful for conducting Bayesian power analysis or predicting new levels in situations where relatively few levels where observed in the old_data. If "old_levels", directly sample new levels from the existing levels, where a new level is assigned all of the posterior draws of the same (randomly chosen) existing level.

incl_autocor A flag indicating if correlation structures originally specified via autocor should be included in the predictions. Defaults to TRUE.

numeric_cov_at An optional (named list) argument to specify the value of continuous covariate(s). The default NULL option set the continuous covariate(s) at 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 varibale 'xx' at 2. The argument numeric_cov_at is ignored when no continuous covariate is included in the model.

levels_id An optional argument to specify the ids for hierarchical model (default NULL). It is used only when model is applied to the data with 3 or more levels of hierarchy. For a two level model, the levels_id is automatically inferred from the model fit. Even for 3 or higher level model, the levels_id is inferred from the model fit but under the assumption that hierarchy is specified from lowest to upper most level i.e, id followed by study where id is nested within the study Note that it is not guaranteed that the levels_id is sorted correctly, and therefore 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 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) such as `avg_reffects = list(feby = 'study', reby = NULL, over = 'age')`.

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

`ipts` An integer to set the length of the predictor variable to get a smooth velocity curve. The `NULL` will return original values whereas an integer such as `ipts = 10` (default) will interpolate the predictor. It is important to note that these interpolations do not alter the range of predictor when calculating population average and/or the individual specific growth curves.

`deriv` An integer to indicate whether to estimate distance curve or its derivative (i.e., velocity curve). The `deriv = 0` (default) is for the distance curve whereas `deriv = 1` for the velocity curve.

`deriv_model` A logical to specify whether to estimate velocity curve from the derivative function, or the differentiation of the distance curve. The argument `deriv_model` is set to `TRUE` for those functions which need velocity curve such as `growthparameters()` and `plot_curves()`, and `NULL` for functions which explicitly use the distance curve (i.e., fitted values) such as `loo_validation()` and `plot_ppc()`.

`summary` A logical indicating whether only the estimate should be computed (`TRUE`), or estimate along with SE and CI should be returned (`FALSE`, default). Setting `summary` as `FALSE` will increase the computation time. Note that `summary = FALSE` is must to get the correct estimates when `re_formula = NULL`.

`robust` A logical to specify the summarize options. 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. Ignored if `summary` is `FALSE`.

`transform` A function applied to individual draws from the posterior distribution, before computing summaries. The argument `transform` is based on the `marginaleffects::predictions()`. This should not be confused with the `transform` from the `brms::posterior_predict()` which is now deprecated.

`probs` The percentiles to be computed by the `quantile` function. Only used if `summary` is `TRUE`.

`xrange` An integer to set the predictor range (i.e., age) when executing the interpolation via `ipts`. The default `NULL` sets the individual specific predictor range whereas code `xrange = 1` sets identical range for individuals within the same higher grouping variable (e.g., study). Code `xrange = 2` sets the identical range across the entire sample. Lastly, a paired numeric values can be supplied e.g., `xrange = c(6, 20)` to set the range within those values.

`xrange_search` A vector of length two, or a character string `'range'` to set the range of predictor variable (`x`) within which growth parameters are searched. This is useful when there is more than one peak and user wants to summarize peak within a given range of the `x` variable. Default `xrange_search = NULL`.

parms_eval
: A logical to specify whether or not to get growth parameters on the fly. This is for internal use only and mainly needed for compatibility across internal functions.

parms_method
: A character to specify the method used to when evaluating parms_eval. The default is getPeak which uses the sitar::getPeak() function from the sitar package. The alternative option is findpeaks that uses this findpeaks from the pracma package. This is for internal use only and mainly needed for compatibility across internal functions.

idata_method
: A character string to indicate the interpolation method. The number of of interpolation points is set up the ipts argument. Options available for idata_method are *method 1* (specified as 'm1') and *method 2* (specified as 'm2'). The *method 1* ('m1') is adapted from the the **iapvbs** package and is documented here https://rdrr.io/github/Zhiqiangcao/iapvbs/src/R/exdata.R whereas *method 2* ('m2') is based on the **JMbayes** package as documented here https://github.com/drizopoulos/JMbayes/blob/master/R/dynPred_lme.R. The 'm1' method works by internally constructing the data frame based on the model configuration whereas the method 'm2' uses the exact data frame used in model fit and can be accessed via fit$data. If idata_method = NULL, default, then method 'm2' is automatically set. Note that method 'm1' might fail in some cases when model involves covariates particularly when model is fit as univariate_by. Therefore, it is advised to switch to method 'm2' in case 'm1' results in error.

verbose
: An optional argument (logical, default FALSE) to indicate whether to print information collected during setting up the object(s).

fullframe
: A logical to indicate whether to return fullframe object in which newdata is bind to the summary estimates. Note that fullframe can not be combined with summary = FALSE. Furthermore, fullframe can only be used when idata_method = 'm2'. A particular use case is when fitting univariate_by model. The fullframe is mainly for internal use only.

dummy_to_factor
: A named list (default NULL) that is used to convert dummy variables into a factor variable. The named elements are factor.dummy, factor.name, and factor.level. The factor.dummy is a vector of character strings that need to be converted to a factor variable whereas the factor.name is a single character string that is used to name the newly created factor variable. The factor.level is used to name the levels of newly created factor. When factor.name is NULL, then the factor name is internally set as 'factor.var'. If factor.level is NULL, then names of factor levels are take from the factor.dummy i.e., the factor levels are assigned same name as factor.dummy. Note that when factor.level is not NULL, its length must be same as the length of the factor.dummy.

expose_function
: An optional logical argument to indicate whether to expose Stan functions (default FALSE). Note that if user has already exposed Stan functions during model fit by setting expose_function = TRUE in the bsitar(), then those exposed functions are saved and can be used during post processing of the posterior draws and therefore expose_function is by default set as FALSE in all post processing functions except optimize_model(). For optimize_model(), the default setting is expose_function = NULL. The reason is that each optimized

model has different Stan function and therefore it need to be re exposed and saved. The `expose_function = NULL` implies that the setting for `expose_function` is taken from the original `model` fit. Note that `expose_function` must be set to `TRUE` when adding `fit criteria` and/or `bayes_R2` during model optimization.

`usesavedfuns` A logical (default `NULL`) to indicate whether to use the already exposed and saved `Stan` functions. Depending on whether the user have exposed Stan functions within the `bsitar()` call via `expose_functions` argument in the `bsitar()`, the `usesavedfuns` is automatically set to `TRUE` (if `expose_functions = TRUE`) or `FALSE` (if `expose_functions = FALSE`). Therefore, manual setting of `usesavedfuns` as `TRUE`/`FALSE` is rarely needed. This is for internal purposes only and mainly used during the testing of the functions and therefore should not be used by users as it might lead to unreliable estimates.

`clearenvfuns` A logical to indicate whether to clear the exposed function from the environment (`TRUE`) or not (`FALSE`). If `NULL` (default), then `clearenvfuns` is set as `TRUE` when `usesavedfuns` is `TRUE`, and `FALSE` if `usesavedfuns` is `FALSE`.

`funlist` A list (default `NULL`) to set function names. This is rarely used because required functions are automatically retrieved internally. A use case of `funlist` when `sigma_formula`, `sigma_formula_gr`, or `sigma_formula_gr_str` uses a function such as `poly(age)`. See `bsitar::bsitar()` for details. The `funlist` provides list of function names which have been defined externally and are available in the `base::globalenv()`. Note that for functions that need distance and velocity curves such as `plot_curves(..., opt = 'dv')`, the `funlist` must include two functions where first will be used for the distance and second for the velocity.

`envir` Environment used for function evaluation. The default is `NULL` which will set `parent.frame()` as default environment. Note that since most of post processing functions are based on **brms**, the functions needed for evaluation should be in the `.GlobalEnv`. Therefore, it is strongly recommended to set `envir = globalenv()` (or `envir = .GlobalEnv`). This is particularly true for the derivatives such as velocity curve.

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

Details

The **predict_draws()** function computed the fitted values from the posterior distribution. The `brms::predict.brmsfit()` function from the **brms** package can used to get the predicted (distance) values when outcome (e.g., height) is untransformed. However, when the outcome is log or square root transformed, the `brms::predict.brmsfit()` function will return the fitted curve on the log or square root scale whereas the **predict_draws()** function returns the fitted values on the original scale. Furthermore, the **predict_draws()** also compute the first derivative of (velocity) that too on the original scale after making required back-transformation. Except for these differences, both these functions (i.e., `brms::predict.brmsfit()` and `predict_draws()`) work in the same manner. In other words, user can specify all the options available in the `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 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
predict_draws(model, deriv = 0, re_formula = NA)


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

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

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

---

update_model.bgmfit *Update model*

Description

The **update_model()** is a wrapper around the `update()` function in the **brms** package which refits the model as per the user specified updated arguments.

Usage

```
## S3 method for class 'bgmfit'
update_model(
  model,
  newdata = NULL,
```

```
  recompile = NULL,
  expose_function = FALSE,
  verbose = FALSE,
  check_newargs = FALSE,
  envir = NULL,
  ...
)

update_model(model, ...)
```

Arguments

model	An object of class bgmfit.
newdata	An optional data.frame to be used when updating the model. If NULL (default), the data used in the original model fit is re used. Note that data-dependent default priors are not updated automatically.
recompile	A logical to indicate whether the Stan model should be recompiled. When NULL (default), **update_model()** tries to figure out internally whether recompilation is required or not. Setting recompile to FALSE will ignore Stan code changing arguments.
expose_function	An optional logical argument to indicate whether to expose Stan functions (default FALSE). Note that if user has already exposed Stan functions during model fit by setting expose_function = TRUE in the bsitar(), then those exposed functions are saved and can be used during post processing of the posterior draws and therefore expose_function is by default set as FALSE in all post processing functions except optimize_model(). For optimize_model(), the default setting is expose_function = NULL. The reason is that each optimized model has different Stan function and therefore it need to be re exposed and saved. The expose_function = NULL implies that the setting for expose_function is taken from the original model fit. Note that expose_function must be set to TRUE when adding fit criteria and/or bayes_R2 during model optimization.
verbose	An optional argument (logical, default FALSE) to indicate whether to print information collected during setting up the object(s).
check_newargs	A logical (default FALSE) to check whether arguments in the original model fit and the update_model are same. When check_newargs = TRUE and arguments are same, it implies that update is not needed and hence the original model object is returned along with the message if verbose = TRUE.
envir	Environment used for function evaluation. The default is NULL which will set parent.frame() as default environment. Note that since most of post processing functions are based on **brms**, the functions needed for evaluation should be in the .GlobalEnv. Therefore, it is strongly recommended to set envir = globalenv() (or envir = .GlobalEnv). This is particularly true for the derivatives such as velocity curve.
...	Other arguments passed to brms::brm().

Details

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

Value

An updated object of class `brmsfit`.

Author(s)

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

Examples

```
# Fit Bayesian SITAR model

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

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

model <- berkeley_exfit

# Update model
# Note that in case all arguments supplied to the update_model() call are
# same as the original model fit (checked via check_newargs = TRUE), then
# original model object is returned.
# To explicitly get this information whether model is being updated or not,
# user can set verbose = TRUE. The verbose = TRUE also useful in getting the
# information regarding what all arguments have been changed as compared to
# the original model.

model2 <- update_model(model, df = 5, check_newargs = TRUE, verbose = TRUE)
```

Index

∗ **datasets**
berkeley, 6
berkeley_exdata, 7
berkeley_exfit, 8

add_model_criterion *(*add_model_criterion.bgmfit*)*, 3
add_model_criterion.bgmfit, 3

base::globalenv(), *47*, *55*, *67*, *82*, *93*, *103*, *112*, *123*
base::on.exit(), *66, 67*, *80, 81*, *91, 92*
base::round(), *54*, *60*, *75*, *86*, *97*, *107*
base::Vectorize(), *42*
berkeley, 6, *7, 8*
berkeley_exdata, 7, *8*
berkeley_exfit, 8
brms::add_criterion(), *3*, *72*, *96–98*
brms::add_ic(), *5*
brms::add_loo, *3*, *5*
brms::add_waic(), *5*
brms::bayes_R2(), *3*, *5*
brms::bridge_sampler(), *33*
brms::brm(), *16, 17*, *19, 20*, *30–35*, *37, 38*, *70*, *125*
brms::brmsfit_needs_refit(), *35*
brms::brmsformula(), *15*, *31*, *34*, *38*
brms::conditional_effects, *103*
brms::conditional_effects(), *103*
brms::custom_family(), *19*
brms::fitted.brmsfit, *48*, *93*
brms::fitted.brmsfit(), *43*, *48*, *52*, *93*, *108*
brms::hypothesis(), *34*
brms::lf(), *17, 18*
brms::loo(), *37*, *69*, *72*
brms::loo_moment_match(), *70*
brms::nlf(), *17, 18*
brms::opencl(), *33*
brms::posterior_predict(), *46*, *51*, *64*, *78*, *89*, *109*, *121*
brms::pp_check(), *37*, *118*
brms::pp_check.brmsfit(), *118*
brms::predict.brmsfit(), *52*, *93*, *108*, *118*, *123, 124*
brms::prior(), *21*, *26*, *37, 38*
brms::reloo(), *70*
brms::save_pars(), *34*
brms::set_prior(), *34*
brms::stancode(), *31*
brms::standata(), *31*
brms::stanvar(), *31*
brms::validate_prior(), *31*
bsitar, 9
bsitar(), *4, 5*, *44*, *47*, *51*, *55*, *59*, *67*, *71*, *74*, *81*, *84*, *92*, *95*, *97*, *101, 102*, *106*, *112*, *116, 117*, *119*, *122, 123*, *125*

expose_model_functions *(*expose_model_functions.bgmfit*)*, 41
expose_model_functions.bgmfit, 41

fitted_draws, *113*
fitted_draws *(*fitted_draws.bgmfit*)*, 43
fitted_draws(), *48*, *52*, *56*, *108*, *113*
fitted_draws.bgmfit, 43
future, *35*
future.apply::future_sapply(), *54*, *61*, *76*, *87*, *109*
future::availableCores(), *4*, *54*, *71*, *77*, *87*, *109*

getNsObject, 49
ggplot2::geom_density(), *116*
ggplot2::geom_histogram(), *116*
ggplot2::geom_jitter(), *116*
ggplot2::stat_ecdf(), *116*

growthparameters (growthparameters.bgmfit), 50
growthparameters(), *106*, *113*
growthparameters.bgmfit, 50
growthparameters_comparison (growthparameters_comparison.bgmfit), *57*
growthparameters_comparison(), *50*
growthparameters_comparison.bgmfit, 57

loo, *70*
loo::loo_compare(), *70*, *72*
loo::loo_moment_match(), *70*
loo_compare, *4*
loo_moment_match, *70*
loo_validation (loo_validation.bgmfit), 69
loo_validation.bgmfit, 69

make_conditions, *100*
marginal_comparison (marginal_comparison.bgmfit), 73
marginal_comparison.bgmfit, 73
marginal_draws (marginal_draws.bgmfit), 83
marginal_draws(), *104*
marginal_draws.bgmfit, 83
marginaleffects::avg_comparisons(), *57*, *62*, *68*, *73*, *77*, *82*, *88*
marginaleffects::avg_predictions(), *83*, *87*, *88*, *93*
marginaleffects::comparisons(), *57*, *62*, *63*, *68*, *73*, *77*, *82*, *88*, *89*
marginaleffects::datagrid(), *59*, *66*, *74*, *75*, *85*
marginaleffects::plot_comparisons(), *62*, *68*, *77*, *82*
marginaleffects::plot_predictions(), *83*, *88*, *93*
marginaleffects::posterior_draws(), *63*
marginaleffects::predictions(), *46*, *51*, *64*, *77*, *78*, *83*, *87–89*, *93*, *109*, *121*

optimize_model (optimize_model.bgmfit), 94
optimize_model(), *5*, *47*, *55*, *67*, *71*, *81*, *92*, *97*, *102*, *112*, *117*, *122*, *125*
optimize_model.bgmfit, 94
options, *33*, *35*

parallel::makeCluster(), *62*
pareto_k_ids, *70*
plan, *35*
plot_conditional_effects (plot_conditional_effects.bgmfit), 98
plot_conditional_effects.bgmfit, 98
plot_curves (plot_curves.bgmfit), 104
plot_curves.bgmfit, 104
plot_ppc (plot_ppc.bgmfit), 114
plot_ppc.bgmfit, 114
PPC, *115*
predict_draws (predict_draws.bgmfit), 118
predict_draws(), *52*, *56*, *108*, *113*, *123*
predict_draws.bgmfit, 118

reloo, *70*
rstan::expose_stan_functions(), *41*, *42*
rstan::rstan(), *35*
rstan::stan_model, *34*

saveRDS, *34*, *35*, *96*
sitar::getPeak(), *46*, *54*, *56*, *63*, *78*, *111*, *122*
sitar::getTakeoff(), *56*, *63*, *78*
sitar::getTrough(), *56*, *63*, *78*
sitar::plot.sitar, *109*
sitar::sitar(), *15*, *17*, *36*
sitar::velout(), *32*
sitar::zapvelout(), *32*
splines::ns(), *36*
stats::density(), *116*
stats::model.matrix(), *15*

update, *34*
update_model, *97*
update_model (update_model.bgmfit), 124
update_model.bgmfit, 124

yaxis_text(), *116*