Package 'bmm'

June 5, 2026

Title Easy and Accessible Bayesian Measurement Models Using 'brms'

Version 1.3.1

Description Fit computational and measurement models using full Bayesian inference. The package provides a simple and accessible interface by translating complex domain-specific models into 'brms' syntax, a powerful and flexible framework for fitting Bayesian regression models using 'Stan'. The package is designed so that users can easily apply state-of-the-art models in various research fields, and so that researchers can use it as a new model development framework. References: Frischkorn and Popov (2025) <doi:10.3758/s13428-025-02643-0>.

License GPL-2

URL https://github.com/venpopov/bmm, https://venpopov.com/bmm/

BugReports https://github.com/venpopov/bmm/issues

Depends R (>= 4.1.0)

Imports bayesplot, brms (>= 2.21.0), crayon, fs, glue, matrixStats, methods, parallel, rtdists, rlang, stats, withr

Suggests bookdown, cmdstanr (>= 0.7.0), cowplot, dplyr, emmeans (>= 1.8.0), fansi, ggplot2, ggthemes, knitr, mixtur, remotes, rmarkdown, stringr, testthat (>= 3.0.0), tidybayes, tidyr, usethis, waldo, yaml

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

Config/testthat/edition 3

Config/Needs/website rmarkdown

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

NeedsCompilation no

Author Vencislav Popov [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-8073-4199>), Gidon T. Frischkorn [aut, cph] (ORCID:

<https://orcid.org/0000-0002-5055-9764>),
Chenyu Li [ctb],
Paul-Christian Bürkner [cph] (Creator of 'brms', code portions of which are used in 'bmm'.)

Maintainer Vencislav Popov <vencislav.popov@gmail.com>

Repository CRAN

Date/Publication 2026-06-05 07:40:02 UTC

Contents

`bmm-package` *Easy and Accesible Bayesian Measurement Models Using 'brms'*

Description

Fit computational and measurement models using full Bayesian inference. The package provides a simple and accessible interface by translating complex domain-specific models into 'brms' syntax, a powerful and flexible framework for fitting Bayesian regression models using 'Stan'. The package is designed so that users can easily apply state-of-the-art models in various research fields, and so that researchers can use it as a new model development framework. References: Frischkorn and Popov (2025) doi:10.3758/s13428025026430.

Author(s)

Maintainer: Vencislav Popov <vencislav.popov@gmail.com> (ORCID) [copyright holder]

Authors:

- Gidon T. Frischkorn <gidon.frischkorn@psychologie.uzh.ch> (ORCID) [copyright holder]

Other contributors:

- Chenyu Li [contributor]
- Paul-Christian Bürkner <paul.buerkner@gmail.com> (Creator of 'brms', code portions of which are used in 'bmm'.) [copyright holder]

See Also

Useful links:

- https://github.com/venpopov/bmm
- https://venpopov.com/bmm/
- Report bugs at https://github.com/venpopov/bmm/issues

`adjust_ezdm_accuracy` *Adjust Accuracy Counts for Contamination*

Description

Adjusts accuracy counts (`n_upper`, `n_trials`) by removing estimated contaminant trials using binomial sampling. Contaminant trials are assumed to produce correct responses at a fixed guess rate (e.g., 0.5 for 2AFC tasks).

Usage

```
adjust_ezdm_accuracy(n_upper, n_trials, contaminant_prop, guess_rate = 0.5)
```

Arguments

`n_upper`	Numeric. Count of upper boundary (correct) responses.
`n_trials`	Numeric. Total number of trials.
`contaminant_prop`	Numeric. Estimated proportion of contaminant trials (e.g., from the `contaminant_prop` column of `ezdm_summary_stats()`).
`guess_rate`	Numeric. Assumed accuracy rate for contaminant trials (random guessing). Default is 0.5 (appropriate for 2AFC tasks).

Details

Uses binomial sampling to estimate the number of contaminant trials and contaminant correct responses, then subtracts these from the raw counts. Because of the stochastic sampling, results will vary across calls unless a seed is set by the user.

Value

A 1-row `data.frame` with columns `n_upper_adj` and `n_trials_adj` (integers). When `contaminant_prop` is `NA` or <= 0, returns the original counts unchanged.

See Also

`ezdm_summary_stats()` for computing the summary statistics and contamination proportions

Examples

```
# Adjust accuracy for estimated 10% contamination
set.seed(42)
adjust_ezdm_accuracy(n_upper = 80, n_trials = 100, contaminant_prop = 0.1)

# In a pipeline with ezdm_summary_stats
# library(dplyr)
# mydata |>
#   group_by(subject) |>
```

```
#   reframe(ezdm_summary_stats(rt, response)) |>
#   mutate(adjust_ezdm_accuracy(n_upper, n_trials, contaminant_prop))
```

---

apply_links *Apply link functions for parameters in a formula or bmmformula*

Description

This function applies the specified link functions in the list of `links` to the `formula` or `bmmformula` that is passed to it. This function is mostly used internally for configuring `bmmodels`.

Usage

```
apply_links(formula, links = nlist())
```

Arguments

`formula`	A `formula` or `bmmformula` that the links should be applied to
`links`	A list of `links` that should be applied to the formula. Each element in this list should be named using the parameter labels the links should be applied for and contain a character variable specifying the link to be applied. Currently implemented links are: "log", "logit", "probit", and "identity".

Value

The `formula` or `bmmformula` the links have been applied to

Examples

```
# specify a bmmformula
form <- bmf(x ~ a + c, kappa ~ 1, a ~ 1, c ~ 1)
links <- list(a = "log", c = "logit")

apply_links(form, links)
```

---

bmm *Fit Bayesian Measurement Models*

Description

Fit a Bayesian measurement model using **brms** as a backend interface to Stan.

Usage

```
bmm(
  formula,
  data,
  model,
  prior = NULL,
  sort_data = getOption("bmm.sort_data", "check"),
  silent = getOption("bmm.silent", 1),
  backend = getOption("brms.backend", NULL),
  file = NULL,
  file_compress = TRUE,
  file_refit = getOption("bmm.file_refit", FALSE),
  ...
)

fit_model(
  formula,
  data,
  model,
  prior = NULL,
  sort_data = getOption("bmm.sort_data", "check"),
  silent = getOption("bmm.silent", 1),
  backend = getOption("brms.backend", NULL),
  ...
)
```

Arguments

formula	An object of class bmmformula. A symbolic description of the model to be fitted.
data	An object of class data.frame, containing data of all variables used in the model. The names of the variables must match the variable names passed to the bmmodel object for required argurments.
model	A description of the model to be fitted. This is a call to a bmmodel such as mixture3p() function. Every model function has a number of required arguments which need to be specified within the function call. Call supported_models() to see the list of supported models and their required arguments
prior	One or more brmsprior objects created by brms::set_prior() or related functions and combined using the c method or the + operator. See also default_prior() for more help. Not necessary for the default model fitting, but you can provide prior constraints to model parameters
sort_data	Logical. If TRUE, the data will be sorted by the predictor variables for faster sampling. If FALSE, the data will not be sorted, but sampling will be slower. If "check" (the default), bmm() will check if the data is sorted, and ask you via a console prompt if it should be sorted. You can set the default value for this option using global options(bmm.sort_data = TRUE/FALSE/"check")) or via bmm_options(sort_data)

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.
backend	Character. The backend to use for fitting the model. Can be "rstan" or "cmdstanr". If NULL (the default), "cmdstanr" will be used if the cmdstanr package is installed, otherwise "rstan" will be used. You can set the default backend using global options(brms.backend = "rstan"/"cmdstanr")
file	Either NULL or a character string. If a string, the fitted model object is saved via saveRDS in a file named after the string. The .rds extension is added automatically. If the file already exists, bmm will load and return the saved model object. 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 bmmfit object for later usage. If the directory of the file does not exist, it will be created.
file_compress	Logical or a character string, specifying one of the compression algorithms supported by saveRDS when saving the fitted model object.
file_refit	Logical or character string. Modifies when the fit stored via the file argument is re-used. Can be set globally for the current R session via the "bmm.file_refit" option (see options). If TRUE or "always", the model is fitted again. If FALSE or "never" (the default), the model saved under the name specified in file will be re-used. Note that unlike in brms, there is no "on_change" option
...	Further arguments passed to brms::brm() or Stan. See the description of brms::brm() for more details

Value

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

Supported Models

The following models are supported:

- cswald(rt, response, links, version)
- ddm(rt, response, links)
- ezdm(mean_rt, var_rt, n_upper, n_trials, links, version)
- imm(resp_error, nt_features, nt_distances, set_size, regex, version)
- m3(resp_cats, num_options, choice_rule, version)
- mixture2p(resp_error)
- mixture3p(resp_error, nt_features, set_size, regex)
- sdm(resp_error, version)

Type ?modelname to get information about a specific model, e.g. ?imm

bmmformula syntax

see this online article for a detailed description of the syntax and how it differs from the syntax for **brmsformula**

Default priors, Stan code and Stan data

For more information about the default priors in **bmm** and about who to extract the Stan code and data generated by bmm and #' brms, see the online article.

Miscellaneous

Type `help(package=bmm)` for a full list of available help topics.

fit_model() is a deprecated alias for **bmm()**.

References

Frischkorn, G. T., & Popov, V. (2023). A tutorial for estimating mixture models for visual working memory tasks in brms: Introducing the Bayesian Measurement Modeling (bmm) package for R. https://doi.org/10.31234/osf.io/umt57

See Also

`supported_models()`, `brms::brm()`, default_prior(), `bmmformula()`, stancode(), standata()

Examples

```
# generate artificial data from the Signal Discrimination Model
dat <- data.frame(y = rsdm(2000))

# define formula
ff <- bmmformula(c ~ 1, kappa ~ 1)

# fit the model
fit <- bmm(
  formula = ff,
  data = dat,
  model = sdm(resp_error = "y"),
  cores = 4,
  backend = "cmdstanr"
)
```

bmmformula *Create formula for predicting parameters of a* bmmodel

Description

This function is used to specify the formulas predicting the different parameters of a bmmodel.

Usage

```
bmmformula(...)

bmf(...)
```

Arguments

...	Formulas for predicting a bmmodel parameter. Each formula for a parameter should be specified as a separate argument, separated by commas

Value

A list of formulas for each parameters being predicted

General formula structure

The formula argument accepts formulas of the following syntax:

```
parameter ~ fixed_effects + (random_effects | grouping_variable)
```

bmm formulas are built on brms formulas and function in nearly the same way, so you can use most of the brms formula syntax. The main differences is that in bmm formulas, the response variable is not specified in the formula. Instead, each parameter of the model is explicitly specified as the left-hand side of the formula. In brms, the response variable is always specified as the left-hand side of the first formula, which implicitly means that any predictors in the first formula are predictors of the mu parameter of the model. In general, measurement models do not all have a mu parameter, therefore it is more straigthforward to explicitely predict each parameter of the model.

For example, in the following brms formula for the drift diffusion model, the first line corresponds to the drift rate parameter, but this is not explicitely stated.

```
brmsformula(rt | dec(response) ~ condition + (condition | id),
            bs ~ 1 + (1 | id),
            ndt ~ 1 + (1 | id),
            bias ~ 1 + (1 | id))
```

In bmm, the same formula would be written as:

```
bmmformula(drift ~ condition + (condition | id),
           bs ~ 1 + (1 | id),
           ndt ~ 1 + (1 | id),
           bias ~ 1 + (1 | id))
```

and the rt and response variables would be specified in the model argument of the `bmm()` function.

Aside from that, the `bmm` formula syntax is the same as the `brms` formula syntax. For more information on the `brms` formula syntax, see `brms::brmsformula()`.

You can also use the `bmf()` function as a shorthand for `bmmformula()`.

You can also set parameters to a constant value by using par = value syntax:

```
bmf(drift ~ condition + (condition | id),
    bs ~ 1 + (1 | id),
    ndt ~ 1 + (1 | id),
    bias = 0.5)
```

in which case the bias parameter will not be estimated but it will be fixed to 0.5

Examples

```
imm_formula <- bmmformula(
  c ~ 0 + set_size + (0 + set_size | id),
  a ~ 1,
  kappa ~ 0 + set_size + (0 + set_size | id)
)

# or use the shorter alias 'bmf'
imm_formula2 <- bmf(
  c ~ 0 + set_size + (0 + set_size | id),
  a ~ 1,
  kappa ~ 0 + set_size + (0 + set_size | id)
)
identical(imm_formula, imm_formula2)
```

`bmm_options` *View or change global bmm options*

Description

View or change global bmm options

Usage

```
bmm_options(
  sort_data,
  parallel,
  default_priors,
  silent,
  color_summary,
  file_refit,
  reset_options = FALSE
)
```

Arguments

sort_data	logical. If TRUE, the data will be sorted by the predictors. If FALSE, the data will not be sorted, but sampling will be slower. If "check" (the default), bmm() will check if the data is sorted, and ask you via a console prompt if it should be sorted. **Default: "check"**
parallel	logical. If TRUE, chains will be run in parallel. If FALSE, chains will be run sequentially. You can also set these value for each model separately via the argument parallel in bmm(). **Default: FALSE**
default_priors	logical. If TRUE (default), the default bmm priors will be used. If FALSE, only the basic brms priors will be used. **Default: TRUE**
silent	numeric. 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. **Default: 1**
color_summary	logical. If TRUE, the summary of the model will be printed in color. **Default: TRUE**
file_refit	logical. If TRUE, bmm() will refit the model even if the file argument is specified. **Default: FALSE**
reset_options	logical. If TRUE, the options will be reset to their default values **Default: FALSE**

Details

The bmm_options function is used to view or change the current bmm options. If no arguments are provided, the function will return the current options. If arguments are provided, the function will change the options and return the old options invisibly. If you provide only some of the arguments, the other options will not be changed. The options are stored in the global options list and will be used by bmm() and other functions in the bmm package. Each of these options can also be set manually using the built-in options() function, by setting the bmm.sort_data, bmm.default_priors, and bmm.silent options.

Value

A message with the current bmm options and their values, and invisibly returns the old options for use with on.exit() and friends.

Examples

```
# view the current options
bmm_options()

# change the options to always sort the data and to use parallel sampling
bmm_options(sort_data = TRUE, parallel = TRUE)

# restore the default options
bmm_options(reset_options = TRUE)

# you can change the options using the options() function as well
options(bmm.sort_data = TRUE, bmm.parallel = TRUE)
bmm_options()

# reset the options to their default values
bmm_options(reset_options = TRUE)

# bmm_options(sort_data = TRUE, parallel = TRUE) will also return the old options
# so you can use it with on.exit()
old_op <- bmm_options(sort_data = TRUE, parallel = TRUE)
on.exit(bmm_options(old_op))

bmm_options(reset_options = TRUE)
```

---

calc_error_relative_to_nontargets

Calculate response error relative to non-target values

Description

Given a vector of responses, and the values of non-targets, this function computes the error relative to each of the non-targets.

Usage

```
calc_error_relative_to_nontargets(data, response, nt_features)
```

Arguments

`data`	A `data.frame` object where each row is a single observation
`response`	Character. The name of the column in `data` which contains the response
`nt_features`	Character vector. The names of the columns in `data` which contain the values of the non-targets

Value

A `data.frame` with n*m rows, where n is the number of rows of `data` and m is the number of non-target variables. It preserves all other columns of `data`, except for the non-target locations, and adds a column `y_nt`, which contains the transformed response error relative to the non-targets

Examples

```
data <- oberauer_lin_2017
data <- calc_error_relative_to_nontargets(data, "dev_rad", paste0("col_nt", 1:7))
hist(data$y_nt, breaks = 100)
```

---

circle_transform *Convert degrees to radians or radians to degrees.*

Description

The helper functions `deg2rad` and `rad2deg` should add convenience in transforming data from degrees to radians and from radians to degrees.

Usage

```
deg2rad(deg)

rad2deg(rad)
```

Arguments

`deg`	A numeric vector of values in degrees.
`rad`	A numeric vector of values in radians.

Value

A numeric vector of the same length as `deg` or `rad`.

Examples

```
degrees <- runif(100, min = 0, max = 360)
radians <- deg2rad(degrees)
degrees_again <- rad2deg(radians)
```

---

conditional_effects.bmmfit *Conditional Effects for BMM Models*

Description

Compute conditional effects for parameters of a bmmfit object. This method provides a more intuitive interface than directly calling `brms::conditional_effects()` on bmmfit objects, by:

- Accepting model parameter names directly (e.g., `"kappa"`, `"thetat"`)
- Automatically determining whether parameters are distributional or non-linear
- Optionally applying inverse link transformations to show parameters on their natural scale

Usage

```
## S3 method for class 'bmmfit'
conditional_effects(x, par = NULL, scale = c("native", "sampling"), ...)
```

Arguments

x	A bmmfit object (created by bmm())
par	Character string. Name of the model parameter to compute effects for. This should be one of the parameter names from the original model specification (see names(xbmmmodel$parameters)). If NULL (the default), conditional effects are computed for all estimated (non-fixed) parameters.
scale	Character. Scale on which to show the parameter: "native" **(default)** Show on natural scale using inverse link transformation. For example, kappa with log link shown on exp scale, thetat with logit (mixture2p) or softmax (mixture3p) link shown on the probability scale. "sampling" Show on the sampling scale (as used during MCMC). For example, kappa with log link shown on log scale.
...	Additional arguments passed to brms::conditional_effects(). Common arguments include: • effects: Character vector specifying which predictor effects to plot • conditions: Named list for setting values of covariates • int_conditions: Conditions for interactions • prob: Probability mass to include in credible intervals (default 0.95) • spaghetti: Logical, whether to add spaghetti lines • method: Method for computing effects ("posterior_predict" or "posterior_epred")

Details

Parameter Types:

bmm models use two types of parameters internally:

- **Non-linear parameters** (nlpar): Core model parameters like kappa, c, a, thetat
- **Distributional parameters** (dpar): Derived parameters used in brms mixture distributions

Users should not need to know this distinction - conditional_effects.bmmfit() automatically routes to the correct parameter type.

Scale Transformations:

By default (scale = "native"), parameters are shown on their natural scale by applying inverse link transformations:

- log link → exp transformation
- logit link → inverse logit (probability scale)
- tan_half link → 2*atan transformation (radians)
- identity link → no transformation

Use scale = "sampling" to see parameters on the scale used during MCMC sampling.

Value

A `brms_conditional_effects` object (from brms), which can be:

- Plotted directly using `plot()`
- Converted to a data frame for custom plotting
- Combined with other conditional effects plots

See Also

`brms::conditional_effects()` for the underlying brms function

Examples

```
## Not run:
# Fit a mixture model with set size effect on kappa
fit <- bmm(
  formula = bmf(kappa ~ 0 + setsize, thetat ~ 1),
  data = zhang_luck_2008,
  model = mixture3p(
    resp_error = "response_error",
    nt_features = paste0("col_lure", 1:5),
    set_size = "setsize"
  )
)

# Get conditional effects for kappa on natural scale (exp of log)
ce_kappa <- conditional_effects(fit, par = "kappa")
plot(ce_kappa)

# Get conditional effects for kappa on log scale (sampling scale)
ce_kappa_log <- conditional_effects(fit, par = "kappa", scale = "sampling")
plot(ce_kappa_log)

# Get effects for thetat (memory probability)
ce_thetat <- conditional_effects(fit, par = "thetat")
plot(ce_thetat)

# Specify which effects to plot
ce_specific <- conditional_effects(fit, par = "kappa", effects = "setsize")

# Combine with other brms options
ce_detailed <- conditional_effects(
  fit,
  par = "kappa",
  effects = "setsize",
  spaghetti = TRUE,
  ndraws = 100
)

## End(Not run)
```

construct_m3_act_funs *Get Activation Functions for different M3 versions*

Description

This function generates the activation functions for different versions of the Memory Measurement Model (m3) implemented in the bmm package. If no bmmodel object is passed then it will print the available model versions.

Usage

```
construct_m3_act_funs(model = NULL, warnings = TRUE)
```

Arguments

model	A bmmodel object that specifies the M3 model for which the activation functions should be generated. If no model is passed the available M3 versions will be printed to the console.
warnings	Logical flag to indicate if information about the generated model formulas should be printed when the function is called.

Value

A bmmformula object with the activation functions for the m3 version specified in the model object. The activation functions use the names of the response categories specified in the model object.

Examples

```
model <- m3(
 resp_cats = c("correct","other", "npl"),
 num_options = c(1, 4, 5),
 version = "ss"
)

construct_m3_act_funs(model, warnings = FALSE)
```

cswald *Censored-Shifted Wald Model*

Description

Censored-Shifted Wald Model

Usage

```
cswald(rt, response, links = NULL, version = c("simple", "crisk"), ...)
```

Arguments

`rt`	The name of the variable in the dataset containing the response times. Response times should be coded in seconds (not milliseconds).
`response`	The name of the variable in the dataset containing the response/decision. Responses should be coded as 0 (lower boundary) or 1 (upper boundary). Alternatively, character values "lower" and "upper" or logical values (FALSE/TRUE) are accepted and will be converted automatically.
`links`	A named list of link functions for the model parameters. Available parameters depend on the version: "simple" has `drift`, `bound`, `ndt`, and `s`; "crisk" additionally has `zr`. Default links are "log" for most parameters and "logit" for `zr`.
`version`	A character string specifying which version of the cswald model to use. Options are: • `"simple"` (default): The standard censored shifted Wald model, which treats error responses as censored correct responses. Best suited for tasks with few errors (<20%). **Note:** The `bound` parameter in the simple version represents the distance from the starting point to the correct boundary, which is half the total boundary separation in the diffusion model (assuming an unbiased starting point). To convert to the full boundary separation (as in DDM or crisk), multiply by 2. • `"crisk"`: The competing risks version, which models both response types as arising from racing accumulators toward opposite boundaries. Better suited for tasks with substantial error rates. The `bound` parameter represents the total boundary separation, consistent with the diffusion model parameterization. For more details, see Miller et al. (2017).
`...`	Additional arguments passed internally (for testing purposes).

Details

- **Domain:** Decision Making / Response times
- **Task:** Choice Reaction Time tasks (with few errors)
- **Name:** Censored-Shifted Wald Model
- **Citation:**
  - Miller, R., Scherbaum, S., Heck, D. W., Goschke, T., & Enge, S. (2017). On the Relation Between the (Censored) Shifted Wald and the Wiener Distribution as Measurement Models for Choice Response Times. Applied Psychological Measurement, 42(2), 116-135. https://doi.org/10.1177/0146621617710465
- **Version:** simple
- **Requirements:**
  - Reaction times should be passed in seconds
- The response variable should be passed numerically: 0 = lower response, 1 = upper response
- **Parameters:**

  - drift: drift rate
  - bound: boundary (distance from starting point to correct boundary)
  - ndt: non-decision time
  - s: diffusion constant
- **Fixed parameters:**
  - mu = 0
  - s = 0
- **Default parameter links:**
  - drift = log; bound = log; ndt = log; s = log
- **Default priors:**
  - drift:
    - main: normal(0,1)
    - effects: normal(0,0.3)
  - bound:
    - main: normal(0,0.3)
    - effects: normal(0,0.3)
  - ndt:
    - main: normal(-2,0.3)
    - effects: normal(0,0.3)
  - s:
    - main: normal(0,0.3)
    - effects: normal(0,0.2)

Value

An object of class bmmodel

See Also

dcswald() and rcswald() for the density and random generation functions.

Examples

```
# generate simulated data from the diffusion model
dat <- rcswald(n = 500, drift = 2, bound = 1.5, ndt = 0.3, zr = 0.5, s = 1)

# specify the model
model <- cswald(rt = "rt", response = "response", version = "simple")

# specify the formula
formula <- bmf(
  drift ~ 1,
  bound ~ 1,
  ndt ~ 1
)
```

```
# fit the model
fit <- bmm(
  formula = formula,
  data = dat,
  model = model,
  cores = 4,
  backend = "cmdstanr"
)
```

---

cswald_dist *Distribution functions for the censored shifted Wald model (*cswald*)*

Description

These functions provide the density, distribution, quantile, and random generation functions for the censored shifted Wald model: cswald. The random generation (rcswald) and distribution functions (pcswald, qcswald) use rtdists::rdiffusion(), rtdists::pdiffusion(), and rtdists::qdiffusion() internally for the "crisk" version, which is theoretically consistent since the censored shifted Wald model is an approximation to the Wiener diffusion model for tasks with high accuracy (few errors).

Usage

```
dcswald(
  rt,
  response,
  drift,
  bound,
  ndt,
  zr = 0.5,
  s = 1,
  version = c("simple", "crisk"),
  log = TRUE
)

rcswald(n, drift, bound, ndt, zr = 0.5, s = 1)

pcswald(
  q,
  response,
  drift,
  bound,
  ndt,
  zr = 0.5,
  s = 1,
  version = "simple",
  lower.tail = TRUE,
  log.p = FALSE
```

```
)

qcswald(
  p,
  response,
  drift,
  bound,
  ndt,
  zr = 0.5,
  s = 1,
  version = "simple",
  lower.tail = TRUE,
  log.p = FALSE
)
```

Arguments

rt	A vector of response times in seconds for which the likelihood should be evaluated
response	A vector of responses coded numerically: 0 = lower response, 1 = upper response
drift	The drift rate
bound	The boundary separation
ndt	The non-decision time
zr	The relative starting point (proportion of boundary separation). Default is 0.5 (unbiased). Values must be between 0 and 1.
s	The diffusion constant - the standard deviation of the noise in the evidence accumulation process. Default is s = 1
version	A character string specifying the version of the cswald for which the likelihood should be returned. Available versions are "simple" and "crisk", the default is "simple."
log	A single logical value indicating if log-likelihoods should be returned, the default is TRUE
n	The number of random samples that should be generated
q	A vector of quantiles (response times) at which to evaluate the CDF
lower.tail	Logical; if TRUE (default), probabilities are P(RT <= q), otherwise P(RT > q)
log.p	Logical; if TRUE, probabilities are returned on the log scale. Default is FALSE
p	A vector of probabilities for which to compute quantiles

Details

Cumulative Distribution Function (pcswald)

For the "simple" version, the CDF is only defined for response = 1 (correct responses), as errors are treated as censored observations. The CDF returns the probability that a correct response occurs by time q. For response = 0, NA is returned with a warning.

For the "crisk" version (competing risks), the CDF computes the defective cumulative distribution P(RT <= q, response = r), which is the probability of responding with the specified response by time q. This uses

rtdists::pdiffusion() internally for accurate computation.

Quantile Function (qcswald)

The quantile function returns the response time q such that P(RT <= q) = p. Similar to the CDF, for the "simple" version this is only defined for response = 1. For the "crisk" version, this uses rtdists::qdiffusion() internally.

Value

- dcswald() returns a numeric vector of (log-)likelihoods.
- rcswald() returns a data.frame with columns rt (response times) and response (1 = upper, 0 = lower).
- pcswald() returns a numeric vector of (log-)probabilities.
- qcswald() returns a numeric vector of quantiles (response times).

See Also

rtdists::rdiffusion(), rtdists::pdiffusion(), rtdists::qdiffusion() for the underlying functions

Examples

```
dat <- rcswald(n = 1000, drift = 2, bound = 1, ndt = 0.3)
head(dat)
hist(dat$rt)
```

---

c_parametrizations *Convert between parametrizations of the c parameter of the SDM distribution*

Description

Convert between parametrizations of the c parameter of the SDM distribution

Usage

```
c_sqrtexp2bessel(c, kappa)

c_bessel2sqrtexp(c, kappa)
```

Arguments

c	Vector of memory strength values
kappa	Vector of precision values

Details

`c_bessel2sqrtexp` converts the memory strength parameter (c) from the bessel parametrization to the sqrtexp parametrization, `c_sqrtexp2bessel` converts from the sqrtexp parametrization to the bessel parametrization.

See the online article for details on the parameterization. The sqrtexp parametrization is the default in the `bmm` package.

Value

A numeric vector of the same length as `c` and `kappa`.

Examples

```
c_bessel <- c_sqrtexp2bessel(c = 4, kappa = 3)
c_sqrtexp <- c_bessel2sqrtexp(c = c_bessel, kappa = 3)
```

---

data_color_judgement_task

Example Data from a Color Judgement Task

Description

Raw data from 50 subjects that completed a color judgement task. Participants had to detect if there were more blue or red dots in a cloud of colored dots. The task included a difficulty manipulation coded by the `condition` variable. In the easy condition 60 percent of the dots were either blue or red, and in the difficult condition 52 percent were either blue or red. In addition the fixation cross preceding the onset of the dot matrix had a variable presentation time coded in the `fix_duration` variable

Usage

```
data_color_judgement_task
```

Format

`data_color_judgement_task`**:**

A data frame with 9941 rows and 10 columns:

ID Character String uniquely identifying each subject

trial The trial number of the experimental trials

condition A character variable coding the difficulty condition

fix_duration The duration in ms the fixation cross was shown before onset of the dot matrix.

target_response The color / response that was more frequent in the dot matrix

response The response coded in red / blue given by the participant

response_correct Boolean variable coding if the response was correct or not.

rt The reaction time in seconds.

pressed_key The actual key pressed by the participants

correct_key The correct key that should have been pressed.

`ddm` *Diffusion Decision Model*

Description

Diffusion Decision Model

Usage

```
ddm(rt, response, links = NULL, ...)
```

Arguments

`rt`	Name of the reaction time variable coding reaction time in seconds in the data.
`response`	Name of the response variable coding the response numerically (0 = lower response / incorrect, 1 = upper response / correct)
`links`	A list of links for the parameters.
`...`	used internally for testing, ignore it

Details

- **Domain:** Decision Making / Response times
- **Task:** Two-Alternative Force Choice RT
- **Name:** Diffusion Decision Model
- **Citation:**
  - Ratcliff, R. (1978). A theory of memory retrieval. Psychological Review, 85(2), 59-108. https://doi.org/10/fjwm2f;
- **Requirements:**
  - The response time should be in seconds and represent the time between onset of the target stimulus until the response execution
- The response should be coded numerically: 0 = lower response, 1 = upper response
- **Parameters:**
  - `drift`: Drift rate = Average rate of evidence accumulation of the decision processes
  - `bound`: Boundary separation = Distance between the decision boundaries that need to be reached
  - `ndt`: Non-decision time = Additional time required beyond the evidence accumulation process

  - zr: Relative starting point = Starting point between the decision thresholds relative to the upper bound.
- **Fixed parameters:**
  - zr = 0
  - mu = 0
- **Default parameter links:**
  - drift = identity; bound = log; ndt = log; zr = logit
- **Default priors:**
  - drift:
    - main: cauchy(0,1)
    - effects: normal(0,0.5)
  - bound:
    - main: normal(0,0.5)
    - effects: normal(0,0.5)
  - ndt:
    - main: normal(-1.5,0.5)
    - effects: normal(0,0.3)
  - zr:
    - main: normal(0,0.5)
    - effects: normal(0,0.3)

Value

An object of class bmmodel

Default behavior

By default, zr is fixed at 0. If you want to estimate zr, add a formula for zr in your bmf() call.

Examples

```
# Simulate data for one subject using rddm
set.seed(123)
n_trials <- 500

# Simulate DDM data with fixed parameters
sim_data <- rddm(
  n = n_trials,
  drift = 1.5,    # drift rate
  bound = 1.2,    # boundary separation
  ndt = 0.3,      # non-decision time
  zr = 0.5        # relative starting point
)

# Prepare data frame
dat <- data.frame(
  rt = sim_data$rt,
```

```
  response = sim_data$response
)

# Define formula (intercept-only model)
ff <- bmmformula(
  drift ~ 1,
  bound ~ 1,
  ndt ~ 1
)

# Specify the DDM model
model <- ddm(rt = "rt", response = "response")

# Fit the model
fit <- bmm(
  formula = ff,
  data = dat,
  model = model,
  cores = 4,
  iter = 1000,
  backend = "cmdstanr"
)

# Check parameter recovery
summary(fit)
```

---

ddm_dist *Distribution function for the Diffusion Decision Model (*ddm*)*

Description

Density and random generation function for the Diffusion Decision Model.

Usage

```
dddm(rt, response, drift, bound, ndt, zr = 0.5, log = TRUE)

rddm(n, drift, bound, ndt, zr = 0.5)
```

Arguments

rt	Vector of response times for which the density should be returned
response	Vector of responses for which the density should be returned
drift	Drift rates of the ddm
bound	Boundary separation of the ddm
ndt	Non-decision time of the ddm
zr	relative starting point of the ddm

log	Logical, indicating if log-densities should be returned (default = TRUE)
n	Number of random samples to generate

`default_prior.bmmformula` *Get Default priors for Measurement Models specified in BMM*

Description

Obtain the default priors for a Bayesian multilevel measurement model, as well as information for which parameters priors can be specified. Given the `model`, the `data` and the `formula` for the model, this function will return the default priors that would be used to estimate the model. Additionally, it will return all model parameters that have no prior specified (flat priors). This can help to get an idea about which priors need to be specified and also know which priors were passed to the `bmm()` function.

The default priors in `bmm` tend to be more informative than the default priors in `brms`, as we use domain knowledge to specify the priors.

Usage

```
## S3 method for class 'bmmformula'
default_prior(object, data, model, formula = object, ...)
```

Arguments

object	A `bmmformula` object
data	An object of class data.frame, containing data of all variables used in the model. The names of the variables must match the variable names passed to the `bmmodel` object for required argurments.
model	A description of the model to be fitted. This is a call to a `bmmodel` such as `mixture3p()` function. Every model function has a number of required arguments which need to be specified within the function call. Call `supported_models()` to see the list of supported models and their required arguments
formula	An object of class `bmmformula`. A symbolic description of the model to be fitted.
...	Further arguments passed to `brms::default_prior()`

Value

A data.frame with columns specifying the `prior`, the `class`, the `coef` and `group` for each of the priors specified. Separate rows contain the information on the parameters (or parameter classes) for which priors can be specified.

See Also

`supported_models()`, `brms::default_prior()`

Examples

```
default_prior(bmf(c ~ 1, kappa ~ 1),
  data = oberauer_lin_2017,
  model = sdm(resp_error = "dev_rad")
)
```

`emmeans-bmmfit` *emmeans support for bmmfit objects*

Description

These S3 methods allow `emmeans::emmeans()` to work seamlessly with bmmfit objects by automatically resolving whether a parameter is a distributional (`dpar`) or non-linear (`nlpar`) parameter in brms. Users can always use `dpar = "parameter_name"` regardless of internal classification.

Usage

```
## S3 method for class 'bmmfit'
recover_data(object, ..., dpar = NULL, nlpar = NULL)

## S3 method for class 'bmmfit'
emm_basis(object, trms, xlev, grid, ..., dpar = NULL, nlpar = NULL)
```

Arguments

`object`	A bmmfit object (created by `bmm()`)
`...`	Additional arguments passed to the brmsfit methods.
`dpar`	Character string. Name of the model parameter (e.g., `"kappa"`, `"c"`, `"thetat"`). Automatically resolved to the correct brms parameter type.
`nlpar`	Character string. Explicit non-linear parameter name. If provided, takes precedence over `dpar`.
`trms`, `xlev`, `grid`	Arguments passed by emmeans internally.

Details

bmm models use two types of parameters internally in brms:

- **Distributional parameters** (`dpar`): Used in models with `custom_family(dpars = ...)` (e.g., SDM, EZDM)
- **Non-linear parameters** (`nlpar`): Used in models with `bmf2bf() + nlf()` (e.g., mixture2p, mixture3p, IMM, M3)

Users should not need to know this distinction. These methods intercept the `dpar` argument and, if the parameter is actually an nlpar, silently re-route it to `nlpar`.

Value

See `emmeans::recover_data()` and `emmeans::emm_basis()` for return values.

See Also

`emmeans::emmeans()`

Examples

```
## Not run:
# Works for any bmm model — no need to know dpar vs nlpar
em <- emmeans(fit, ~ condition, dpar = "kappa")
pairs(em)
confint(em)

## End(Not run)
```

`extract_parameter_dimensions` *Extract dimension from parameters in STAN parameter block*

Description

This functions extracts the names, dimensions, and types from a compiled STAN parameters blocks generated by bmm or brms. This function is used to specify initial values for bmm models.

Usage

```
extract_parameter_dimensions(parameters_block)
```

Arguments

`parameters_block` The parameters block extracted via `extract_stan_blocks`

Value

A list of all parameters, their types, and dimensions as as specified in the STAN data generated by bmm and brms

Examples

```
# generate simple stan code from brms
stan_code <- stancode(brms::bf(x ~ 1 + cond + (1 + cond | ID)),
  data = data.frame(
    x = rnorm(100),
    ID = rep(1:50, each = 2),
    cond = rep(1:2, times = 50)
```

```
  )
)

extracted_program_blocks <- extract_stan_blocks(stan_code)

par_dims <- extract_parameter_dimensions(extracted_program_blocks$parameters) #'
```

`extract_stan_blocks` *Extract code from different STAN program blocks*

Description

This function extracts the code from the different program blocks of a STAN program. This can be used in combination with the `stancode` function to access information about the STAN code generated by `brms` and `bmm`.

Usage

```
extract_stan_blocks(
  stan_code,
  blocks = c("functions", "data", "transformed data", "parameters",
    "transformed parameters", "model", "generated quantities")
)
```

Arguments

`stan_code`	The STAN code for which the elements should be extracted
`blocks`	A character vector specifying for which program blocks the code should be extracted. The default extracks all standard blocks: "functions", "data", "transformed data", "parameters", "transformed parameters", "model", and "generated quantities"

Value

A named list with each element containing the code of one of the STAN program blocks. If a block

Examples

```
# generate simple stan code from brms
stan_code <- stancode(brms::bf(x ~ 1), data = data.frame(x = rnorm(100)))

extracted_program_blocks <- extract_stan_blocks(stan_code)
```

ezdm *EZ-Diffusion Model*

Description

EZ-Diffusion Model

Usage

```
ezdm(mean_rt, var_rt, n_upper, n_trials, links = NULL, version = "3par", ...)
```

Arguments

mean_rt	The names of the variable or variables (for 4par version) coding the mean reaction time in seconds in the data.
var_rt	The names of the variable or variables (for 4par version) coding the variance of the reaction time in seconds in the data
n_upper	The name of the variable coding the number of responses that hit the upper response threshold (typically the number of correct responses) in the data.
n_trials	The name of the variable coding the number of trials that was used to calculated the aggregated statistics.
links	A list of links for the parameters.
version	A character label for the version of the model. There is a three-parameter version (version = "3par") of the ezdm that fixes the relative starting point zr to 0.5, and a four parameter version (version = "4par"), that allows to freely estimate the starting point.
...	used internally for testing, ignore it

Details

- **Domain:** Decision Making / Response times
- **Task:** Choice Reaction Time tasks
- **Name:** EZ-Diffusion Model
- **Citation:**
  - Wagenmakers, E.-J., Van Der Maas, H. L. J., & Grasman, R. P. P. P. (2007). An EZ-diffusion model for response time and accuracy. Psychonomic Bulletin & Review, 14(1), 3-22. https://doi.org/10/fk447c
- Chávez De la Peña, A. F., & Vandekerckhove, J. (2025). An EZ Bayesian hierarchical drift diffusion model for response time and accuracy. Psychonomic Bulletin & Review. https://doi.org/10.3758/s13423-025-02729-y
- **Version:** 4par

- **Requirements:**
  Provide aggregated statistics for each subject and condition that model parameters should vary over:

- Mean reaction times (mean_rt) in seconds
- Variance of reaction times (var_rt) in seconds
- Number of responses to the upper decision threshold (n_upper)
- Total number of trials used to calculate aggregated statistics (n_trials)

- **Parameters:**
  - `drift`: Drift rate = Average rate of evidence accumulation of the decision processes
  - `bound`: Boundary separation = Distance between the decision boundaries that need to be reached
  - `ndt`: Non-decision time = Additional time required beyond the evidence accumulation process
  - `zr`: Relative starting point = Starting point between the decision thresholds relative to the upper bound.
  - `s`: The diffusion constant, that is the standard deviation of the Gaussian noise during sampling
- **Fixed parameters:**
  - `s` = 0
  - `mu` = 0
- **Default parameter links:**
  - drift = identity; bound = log; ndt = log; zr = logit; s = log
- **Default priors:**
  - `drift`:
    - `main`: cauchy(0,1)
    - `effects`: normal(0,0.5)
  - `bound`:
    - `main`: normal(0,0.5)
    - `effects`: normal(0,0.5)
  - `ndt`:
    - `main`: normal(-1.5,0.5)
    - `effects`: normal(0,0.3)
  - `zr`:
    - `main`: normal(0,0.5)
    - `effects`: normal(0,0.3)
  - `s`:
    - `main`: normal(0,1)
    - `effects`: normal(0,0.3)

Value

An object of class `bmmodel`

Examples

```
## Not run:
# Minimal parameter recovery example with 3-parameter EZDM

# Simulate data from known parameters
set.seed(123)
sim_data <- rezdm(
  n = 10,
  n_trials = 100,
  drift = 2,
  bound = 1.5,
  ndt = 0.3,
  version = "3par"
)

# Add subject ID
sim_data$id <- 1:10

# Specify model
model <- ezdm(
  mean_rt = "mean_rt",
  var_rt = "var_rt",
  n_upper = "n_upper",
  n_trials = "n_trials",
  version = "3par"
)

# Specify formula with random effects
formula <- bmf(
  drift ~ 1 + (1 | id),
  bound ~ 1 + (1 | id),
  ndt ~ 1
)

# Fit model (using cmdstanr backend)
fit <- bmm(
  formula = formula,
  data = sim_data,
  model = model,
  backend = "cmdstanr",
  cores = 4,
  chains = 4,
  iter = 2000,
  warmup = 1000
)

# Check parameter recovery
summary(fit)

# Extract population-level effects
# True values: drift = 2, bound = 1.5, ndt = 0.3 (on log scale for drift/bound)
exp(brms::fixef(fit))
```

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

`ezdm_dist` *Distribution functions for the EZ-Diffusion Model (ezdm)*

Description

Density and random generation functions for the EZ-Diffusion Model. The model operates on aggregated data: mean reaction time, variance of reaction time, and number of responses to the upper boundary.

Usage

```
dezdm(
  mean_rt,
  var_rt,
  n_upper,
  n_trials,
  drift,
  bound,
  ndt,
  zr = 0.5,
  s = 1,
  version = c("3par", "4par"),
  log = TRUE
)

rezdm(
  n,
  n_trials,
  drift,
  bound,
  ndt,
  zr = 0.5,
  s = 1,
  version = c("3par", "4par")
)
```

Arguments

`mean_rt`	Observed mean reaction time(s) in seconds. For version "3par", a numeric vector or single value. For version "4par", either a vector of length 2 (c(mean_rt_upper, mean_rt_lower)) for single observation, or a matrix with 2 columns for multiple observations.

var_rt	Observed variance of reaction times in seconds^2. For version "3par", a numeric vector or single value. For version "4par", either a vector of length 2 (c(var_rt_upper, var_rt_lower)) for single observation, or a matrix with 2 columns for multiple observations.
n_upper	Number of responses to the upper boundary
n_trials	Total number of trials
drift	Drift rate (evidence accumulation rate; can be positive or negative for below-chance performance).
bound	Boundary separation (distance between decision thresholds).
ndt	Non-decision time (seconds).
zr	Relative starting point (0 to 1). Only used for version "4par".
s	Diffusion constant (standard deviation of noise), default = 1.
version	Character; either "3par" (default) or "4par"
log	Logical; if TRUE, values are returned on the log scale.
n	Number of samples to generate

Value

dezdm gives the log-density of the observed summary statistics under the EZDM, and rezdm generates random summary statistics from the implied sampling distributions.

References

Wagenmakers, E.-J., Van Der Maas, H. L. J., & Grasman, R. P. P. P. (2007). An EZ-diffusion model for response time and accuracy. Psychonomic Bulletin & Review, 14(1), 3-22.

Chávez De la Peña, A. F., & Vandekerckhove, J. (2025). An EZ Bayesian hierarchical drift diffusion model for response time and accuracy. Psychonomic Bulletin & Review.

Examples

```
# 3-parameter version (single observation)
dezdm(
  mean_rt = 0.5, var_rt = 0.02, n_upper = 80, n_trials = 100,
  drift = 2, bound = 1.5, ndt = 0.3
)

# 3-parameter version (vectorized)
dezdm(
  mean_rt = c(0.5, 0.55), var_rt = c(0.02, 0.025),
  n_upper = c(80, 75), n_trials = c(100, 100),
  drift = 2, bound = 1.5, ndt = 0.3
)

# 4-parameter version (single observation)
dezdm(
  mean_rt = c(0.45, 0.55), var_rt = c(0.018, 0.025),
  n_upper = 80, n_trials = 100,
```

```
  drift = 2, bound = 1.5, ndt = 0.3, zr = 0.55, version = "4par"
)

# generate random summary statistics
rezdm(n = 100, n_trials = 100, drift = 2, bound = 1.5, ndt = 0.3)
rezdm(
  n = 100, n_trials = 100, drift = 2, bound = 1.5, ndt = 0.3,
  zr = 0.55, version = "4par"
)
```

`ezdm_summary_stats` *Compute Robust Summary Statistics for EZ-Diffusion Model*

Description

Computes robust summary statistics for the EZ-Diffusion Model by fitting mixture models to raw trial-level RT data, separating contaminant responses from true responses.

Usage

```
ezdm_summary_stats(
  rt,
  response,
  version = c("3par", "4par"),
  distribution = c("exgaussian", "lognormal", "invgaussian"),
  method = c("mixture", "simple", "robust"),
  robust_scale = c("iqr", "mad"),
  contaminant_bound = c("min", "max"),
  min_trials = 10,
  init_contaminant = 0.05,
  max_contaminant = 0.5,
  maxit = 100,
  tol = 1e-06
)
```

Arguments

`rt`	Numeric vector of reaction times in seconds.
`response`	Vector of response indicators. Accepts multiple formats: • Numeric: 1 = upper/correct, 0 = lower/error • Logical: TRUE = upper/correct, FALSE = lower/error • Character/Factor: "upper"/"lower", "correct"/"error", "acc"/"err", "hit"/"miss", "yes"/"no" (case-insensitive)
`version`	Character. Either "3par" (default) for pooled RTs or "4par" for separate upper/lower boundary RTs. Controls the output columns.

`distribution` Character. The parametric distribution for the RT component. One of "exgaussian" (default), "lognormal", or "invgaussian"

`method` Character. One of "mixture" (default) for robust estimation via mixture modeling, "robust" for non-parametric robust estimation using median and IQR/MAD-based variance, or "simple" for standard moment calculation. The "robust" method is faster and requires no distributional assumptions, but note that the EZ equations were derived for mean and variance, so using median may introduce some bias for skewed distributions.

`robust_scale` Character. Scale estimator for robust method. Either "iqr" (default) for IQR-based variance estimation (variance = (IQR/1.349)^2) or "mad" for MAD-based estimation (variance = MAD^2, where MAD is scaled to be consistent with SD for normal data). Only used when method = "robust".

`contaminant_bound`
Vector of length 2 specifying the bounds (in seconds) for the uniform contaminant distribution. Can be numeric values or the special strings "min" and "max" to use data-driven bounds (default):

- "min": Use the minimum RT in each group, minus a 50\
- "max": Use the maximum RT in each group, plus a 50\
- Numeric: Fixed bounds, e.g., c(0.1, 3.0)

The buffer extends data-driven bounds to ensure conservative estimates. Examples: c(0.1, 3.0), c("min", "max"), c(0.1, "max"), c("min", 3.0)

`min_trials` Integer. Minimum number of trials required for fitting. Returns NA if fewer trials are available. Default is 10

`init_contaminant`
Numeric. Initial proportion of contaminants for EM algorithm. Default is 0.05

`max_contaminant`
Numeric. Maximum allowed contaminant proportion (0 < max <= 1). Estimates are clipped to this value to prevent inflated contaminant proportions. Default is 0.5

`maxit` Integer. Maximum number of EM iterations. Default is 100

`tol` Numeric. Convergence tolerance for EM algorithm. Default is 1e-6

Details

RT outliers and contaminant responses (fast guesses, lapses of attention) can distort the mean and variance estimates used as input to the EZ-Diffusion equations. This function addresses this by fitting a mixture model with two components: a uniform distribution for contaminants and a parametric RT distribution for true responses. Robust moments are then extracted from the fitted parametric component.

This function is designed to work with `dplyr::group_by()` and `dplyr::reframe()` for grouped operations. Use `adjust_ezdm_accuracy()` as a separate step if you need to adjust accuracy counts for contamination.

Value

A 1-row data.frame. For version = "3par": mean_rt, var_rt, n_upper, n_trials, contaminant_prop. For version = "4par": mean_rt_upper, mean_rt_lower, var_rt_upper, var_rt_lower, n_upper, n_trials, contaminant_prop_upper, contaminant_prop_lower.

See Also

adjust_ezdm_accuracy() for adjusting accuracy counts, flag_contaminant_rts() for trial-level contamination probabilities, ezdm() for fitting the EZ-Diffusion Model

Examples

```
# Generate example data
set.seed(123)
rt <- rgamma(100, shape = 5, rate = 10) + 0.3
response <- rbinom(100, 1, 0.8)

# 3par summary stats
ezdm_summary_stats(rt, response)

# With dplyr for grouped operations
# library(dplyr)
# mydata |>
#   group_by(subject) |>
#   reframe(ezdm_summary_stats(rt, response))

# 4par version with separate upper/lower moments
ezdm_summary_stats(rt, response, version = "4par")
```

---

fit_info *Extract information from a brmsfit object*

Description

Extract information from a brmsfit object

Usage

```
fit_info(fit, what)
```

Arguments

fit	A brmsfit object, or a list of brmsfit objects
what	String. What to return: • "time" for the sampling time per chain • "time_mean" for the mean sampling time

Value

Depends on `what` and the class of `fit`. For `brmsfit` objects, information about the single fit is returned. For `brmsfit_list` objects, a list or data.frame with the information for each fit is returned.

- "time": A data.frame with the sampling time per chain
- "time_mean": A named numeric vector with the mean sampling time

Examples

```
fit <- bmm(
  formula = bmmformula(c ~ 1, kappa ~ 1),
  data = data.frame(y = rsdm(1000)),
  model = sdm(resp_error = "y")
)

fit_info(fit, "time")
```

---

flag_contaminant_rts *Flag contaminant reaction times using mixture modeling*

Description

Identifies contaminant RTs (fast guesses, attention lapses) at the trial level using mixture modeling. For each trial, it computes the posterior probability of being a contaminant given a mixture of a uniform distribution (contaminants) and an RT distribution.

The function takes a numeric vector of RTs and returns a numeric vector of contamination probabilities, making it compatible with `dplyr::mutate()` and `dplyr::group_by()` workflows.

Usage

```
flag_contaminant_rts(
  rt,
  distribution = c("exgaussian", "lognormal", "invgaussian"),
  contaminant_bound = c("min", "max"),
  init_contaminant = 0.05,
  max_contaminant = 0.5,
  maxit = 100,
  tol = 1e-06
)
```

Arguments

`rt`	Numeric vector. Reaction times in seconds. Must be positive.
`distribution`	Character. RT distribution for the mixture model: "exgaussian" (default), "lognormal", or "invgaussian".

contaminant_bound
: Vector of length 2. Bounds [lower, upper] for the uniform contaminant distribution. Can be numeric values or "min"/"max" for data-driven bounds. Default c("min", "max").

init_contaminant
: Numeric. Initial contaminant proportion for EM algorithm. Must be in (0, 1). Default 0.05.

max_contaminant
: Numeric. Maximum allowed contaminant proportion. Values exceeding this are clipped with a warning. Must be in (0, 1]. Default 0.5.

maxit
: Integer. Maximum EM iterations. Default 100.

tol
: Numeric. Convergence tolerance for log-likelihood. Default 1e-6.

Details

Mixture Model:

The function fits: f(RT) = pi_c * Uniform(a,b) + (1-pi_c) * f_RT(RT|theta)

where pi_c is the contaminant proportion, Uniform(a,b) is the contaminant distribution over contaminant_bound, and f_RT is the specified RT distribution with parameters theta.

Grouping:

To fit separate mixtures by condition or response boundary, use dplyr::group_by() before calling this function inside dplyr::mutate().

Diagnostics:

Mixture fit diagnostics (parameters, convergence, log-likelihood) are attached as the "diagnostics" attribute of the returned vector. Access them with attr(result, "diagnostics").

Value

Numeric vector of posterior contamination probabilities P(contaminant | RT), with a "diagnostics" attribute containing a one-row data.frame with columns: mixture_params (list), contaminant_prop, converged, iterations, loglik, n_trials, distribution, method.

See Also

ezdm_summary_stats() for aggregated RT statistics with contamination handling, validate_fast_guesses() for testing whether flagged contaminants show random guessing behavior

Examples

```
## Not run:
# Simulate data with contaminants
library(bmm)
set.seed(123)
rt_clean <- rgamma(150, shape = 5, rate = 10)
rt_contam <- runif(50, 0.1, 0.2)

data <- data.frame(
```

```
  rt = c(rt_clean, rt_contam),
  subject = 1,
  response = sample(c("upper", "lower"), 200, replace = TRUE)
)

# Basic usage with mutate
library(dplyr)
data <- data |>
  mutate(contam_prob = flag_contaminant_rts(rt))

# Hard threshold: remove trials with P(contaminant) > 0.5
data_clean <- data |> filter(contam_prob <= 0.5)

# Separate fits by response boundary
data <- data |>
  group_by(subject, response) |>
  mutate(contam_prob = flag_contaminant_rts(rt))

# Access diagnostics
probs <- flag_contaminant_rts(data$rt)
attr(probs, "diagnostics")

## End(Not run)
```

`imm` *Interference measurement model by Oberauer and Lin (2017).*

Description

Three versions of the Interference measurement model by Oberauer and Lin (2017). - the full, bsc, and abc. `IMMfull()`, `IMMbsc()`, and `IMMabc()` are deprecated and will be removed in the future. Please use `imm(version = 'full')`, `imm(version = 'bsc')`, or `imm(version = 'abc')` instead.

Usage

```
imm(
  resp_error,
  nt_features,
  nt_distances,
  set_size,
  regex = FALSE,
  version = "full",
  ...
)

IMMfull(resp_error, nt_features, nt_distances, set_size, regex = FALSE, ...)

IMMbsc(resp_error, nt_features, nt_distances, set_size, regex = FALSE, ...)

IMMabc(resp_error, nt_features, set_size, regex = FALSE, ...)
```

Arguments

resp_error	The name of the variable in the provided dataset containing the response error. The response Error should code the response relative to the to-be-recalled target in radians. You can transform the response error in degrees to radian using the deg2rad function.
nt_features	A character vector with the names of the non-target variables. The non_target variables should be in radians and be centered relative to the target. Alternatively, if regex=TRUE, a regular expression can be used to match the non-target feature columns in the dataset.
nt_distances	A vector of names of the columns containing the distances of non-target items to the target item. Alternatively, if regex=TRUE, a regular expression can be used to match the non-target distances columns in the dataset. Only necessary for the bsc and full versions.
set_size	Name of the column containing the set size variable (if set_size varies) or a numeric value for the set_size, if the set_size is fixed.
regex	Logical. If TRUE, the nt_features and nt_distances arguments are interpreted as a regular expression to match the non-target feature columns in the dataset.
version	Character. The version of the IMM model to use. Can be one of full, bsc, or abc. The default is full.
...	used internally for testing, ignore it

Details

- **Domain:** Visual working memory
- **Task:** Continuous reproduction
- **Name:** Interference measurement model by Oberauer and Lin (2017).
- **Citation:**
  - Oberauer, K., & Lin, H.Y. (2017). An interference model of visual working memory. Psychological Review, 124(1), 21-59

Version: full**:**

- **Requirements:**
  - The response vairable should be in radians and represent the angular error relative to the target
- The non-target features should be in radians and be centered relative to the target
- **Parameters:**
  - mu1: Location parameter of the von Mises distribution for memory responses (in radians). Fixed internally to 0 by default.
  - kappa: Concentration parameter of the von Mises distribution
  - a: General activation of memory items
  - c: Context activation
  - s: Spatial similarity gradient

- **Fixed parameters:**
  - `mu1` = 0
  - `mu2` = 0
  - `kappa2` = -100
- **Default parameter links:**
  - mu1 = tan_half; kappa = log; a = log; c = log; s = log
- **Default priors:**
  - `mu1`:
    - `main`: student_t(1, 0, 1)
  - `kappa`:
    - `main`: normal(2, 1)
    - `effects`: normal(0, 1)
  - `a`:
    - `main`: normal(0, 1)
    - `effects`: normal(0, 1)
  - `c`:
    - `main`: normal(0, 1)
    - `effects`: normal(0, 1)
  - `s`:
    - `main`: normal(0, 1)
    - `effects`: normal(0, 1)

Version: `bsc`**:**

- **Requirements:**
  - The response vairable should be in radians and represent the angular error relative to the target
- The non-target features should be in radians and be centered relative to the target
- **Parameters:**
  - `mu1`: Location parameter of the von Mises distribution for memory responses (in radians). Fixed internally to 0 by default.
  - `kappa`: Concentration parameter of the von Mises distribution
  - `c`: Context activation
  - `s`: Spatial similarity gradient
- **Fixed parameters:**
  - `mu1` = 0
  - `mu2` = 0
  - `kappa2` = -100
- **Default parameter links:**
  - mu1 = tan_half; kappa = log; c = log; s = log
- **Default priors:**
  - `mu1`:
    - `main`: student_t(1, 0, 1)

  - kappa:
    * main: normal(2, 1)
    * effects: normal(0, 1)
  - c:
    * main: normal(0, 1)
    * effects: normal(0, 1)
  - s:
    * main: normal(0, 1)
    * effects: normal(0, 1)

Version: abc**:**

- **Requirements:**
  - The response vairable should be in radians and represent the angular error relative to the target
- The non-target features should be in radians and be centered relative to the target
- **Parameters:**
  - mu1: Location parameter of the von Mises distribution for memory responses (in radians). Fixed internally to 0 by default.
  - kappa: Concentration parameter of the von Mises distribution
  - a: General activation of memory items
  - c: Context activation
- **Fixed parameters:**
  - mu1 = 0
  - mu2 = 0
  - kappa2 = -100
- **Default parameter links:**
  - mu1 = tan_half; kappa = log; a = log; c = log
- **Default priors:**
  - mu1:
    * main: student_t(1, 0, 1)
  - kappa:
    * main: normal(2, 1)
    * effects: normal(0, 1)
  - a:
    * main: normal(0, 1)
    * effects: normal(0, 1)
  - c:
    * main: normal(0, 1)
    * effects: normal(0, 1)

Additionally, all imm models have an internal parameter that is fixed to 0 to allow the model to be identifiable. This parameter is not estimated and is not included in the model formula. The parameter is:

- b = "Background activation (internally fixed to 0)"

Value

An object of class bmmodel

Examples

```
# load data
data <- oberauer_lin_2017

# define formula
ff <- bmmformula(
  kappa ~ 0 + set_size,
  c ~ 0 + set_size,
  a ~ 0 + set_size,
  s ~ 0 + set_size
)

# specify the full IMM model with explicit column names for non-target features and distances
# by default this fits the full version of the model
model1 <- imm(
  resp_error = "dev_rad",
  nt_features = paste0("col_nt", 1:7),
  nt_distances = paste0("dist_nt", 1:7),
  set_size = "set_size"
)

# fit the model
fit <- bmm(
  formula = ff,
  data = data,
  model = model1,
  cores = 4,
  backend = "cmdstanr"
)

# alternatively specify the IMM model with a regular expression to match non-target features
# this is equivalent to the previous call, but more concise
model2 <- imm(
  resp_error = "dev_rad",
  nt_features = "col_nt",
  nt_distances = "dist_nt",
  set_size = "set_size",
  regex = TRUE
)

# fit the model
fit <- bmm(
  formula = ff,
  data = data,
  model = model2,
  cores = 4,
  backend = "cmdstanr"
)
```

```
# you can also specify the `bsc` or `abc` versions of the model to fit a reduced version
model3 <- imm(
  resp_error = "dev_rad",
  nt_features = "col_nt",
  set_size = "set_size",
  regex = TRUE,
  version = "abc"
)
fit <- bmm(
  formula = ff,
  data = data,
  model = model3,
  cores = 4,
  backend = "cmdstanr"
)
```

IMMdist *Distribution functions for the Interference Measurement Model (IMM)*

Description

Density, distribution, and random generation functions for the interference measurement model with the location of mu, strength of cue- dependent activation c, strength of cue-independent activation a, the generalization gradient s, and the precision of memory representations kappa.

Usage

```
dimm(
  x,
  mu = c(0, 2, -1.5),
  dist = c(0, 0.5, 2),
  c = 5,
  a = 2,
  b = 1,
  s = 2,
  kappa = 5,
  log = FALSE
)

pimm(
  q,
  mu = c(0, 2, -1.5),
  dist = c(0, 0.5, 2),
  c = 1,
  a = 0.2,
  b = 0,
```

```
  s = 2,
  kappa = 5
)

qimm(
  p,
  mu = c(0, 2, -1.5),
  dist = c(0, 0.5, 2),
  c = 1,
  a = 0.2,
  b = 0,
  s = 2,
  kappa = 5
)

rimm(
  n,
  mu = c(0, 2, -1.5),
  dist = c(0, 0.5, 2),
  c = 1,
  a = 0.2,
  b = 1,
  s = 2,
  kappa = 5
)
```

Arguments

x	Vector of observed responses
mu	Vector of locations
dist	Vector of distances of the item locations to the cued location
c	Vector of strengths for cue-dependent activation
a	Vector of strengths for cue-independent activation
b	Vector of baseline activation
s	Vector of generalization gradients
kappa	Vector of precision values
log	Logical; if TRUE, values are returned on the log scale.
q	Vector of quantiles
p	Vector of probability
n	Number of observations to generate data for

Value

dimm gives the density of the interference measurement model, pimm gives the cumulative distribution function of the interference measurement model, qimm gives the quantile function of the interference measurement model, and rimm gives the random generation function for the interference measurement model.

References

Oberauer, K., Stoneking, C., Wabersich, D., & Lin, H.-Y. (2017). Hierarchical Bayesian measurement models for continuous reproduction of visual features from working memory. Journal of Vision, 17(5), 11.

Examples

```
# generate random samples from the imm and overlay the density
r <- rimm(10000,
  mu = c(0, 2, -1.5), dist = c(0, 0.5, 2),
  c = 5, a = 2, s = 2, b = 1, kappa = 4
)
x <- seq(-pi, pi, length.out = 10000)
d <- dimm(x,
  mu = c(0, 2, -1.5), dist = c(0, 0.5, 2),
  c = 5, a = 2, s = 2, b = 1, kappa = 4
)
hist(r, breaks = 60, freq = FALSE)
lines(x, d, type = "l", col = "red")
```

---

k2sd *Transform kappa of the von Mises distribution to the circular standard deviation*

Description

This function transforms the precision parameter kappa of the von Mises distribution to the circular standard deviation. Adapted from Matlab code by Paul Bays (https://www.paulbays.com/code.php)

Usage

```
k2sd(K)
```

Arguments

K	numeric. A vector of kappa values.

Value

A vector of sd values.

Examples

```
kappas <- runif(1000, 0.01, 100)

# calcualte SD (in radians)
SDs <- k2sd(kappas)
```

```
# transform SDs from radians to degrees
SDs_degress <- SDs * 180 / pi

# plot the relationship between kappa and circular SD
plot(kappas, SDs)
plot(kappas, SDs_degress)
```

---

m3 *The Multinomial / Memory Measurement Model*

Description

The Multinomial / Memory Measurement Model (M3) is a measurement model that was originally introduced for working memory tasks with categorical responses. It assumes that each candidate in each response category is activated by a combination of sources of activation. The probability of choosing a response category is determined by the activation of the candidates. The model can be used for any n-AFC categorical decision task.

Usage

```
m3(resp_cats, num_options, choice_rule = "softmax", version = "custom", ...)
```

Arguments

`resp_cats`	The variable names that contain the number of responses for each of the response categories used for the M3.
`num_options`	Either an integer vector of the same length as `resp_cats` if the number of candidates in the respective response categories are constant across all conditions in the experiment. Or a vector specifying the variable names that contain the number of candidates in each response category. The order of these variables should be in the same order as the names of the response categories passed to `resp_cats`
`choice_rule`	The choice rule that should be used for the M3. The options are "softmax" or "simple". The "softmax" option implements the softmax normalization of activation into probabilities for choosing the different response categories. The "simple" option implements a simple normalization of the absolute activations over the sum of all activations. For details on the differences of these choice rules please see the appendix of Oberauer & Lewandowsky (2019) "Simple measurement models for complex working memory tasks" published in Psychological Review.
`version`	Character. The version of the M3 model to use. Can be one of `ss`, `cs`, or `custom`. The default is `custom`.
`...`	used internally for testing, ignore it

Details

- **Domain:** Working Memory (categorical), Categorical Decision Making
- **Task:** n-AFC retrieval
- **Name:** The Multinomial / Memory Measurement Model
- **Citation:**
  - Oberauer, K., & Lewandowsky, S. (2019). Simple measurement models for complex working-memory tasks. Psychological Review, 126.

Version: `ss`**:**

- **Requirements:**
  - Provide names for variables specifying the number of responses in a set of response categories.
  - Specify activation sources for each response categories
  - Include at least an activation source "b" for all response categories
  - Predict the specified activation at least by a fixed intercept and any additional predictors from your data
- **Parameters:**
  - `b`: Background activation. Added to each response category. Fixed for scaling, necessary in all models.
  - `c`: Context activation. Added to the item cued to be recalled, that is the correct item.
  - `a`: General activation. Added to all items that were presented during the current trial.
- **Fixed parameters:**
  - `b` = 0
- **Default parameter links:**
  - c = identity; a = identity
- **Default priors:**
  - `a`:
    - `main`: normal(2,1)
    - `effects`: normal(0,0.5)
  - `c`:
    - `main`: normal(3,1)
    - `effects`: normal(0,2)

Version: `cs`**:**

- **Requirements:**
  - Provide names for variables specifying the number of responses in a set of response categories.
  - Specify activation sources for each response categories
  - Include at least an activation source "b" for all response categories
  - Predict the specified activation at least by a fixed intercept and any additional predictors from your data
- **Parameters:**

  - b: Background activation. Added to each response category. Fixed for scaling, necessary in all models.
  - c: Context activation. Added to the item cued to be recalled, that is the correct item.
  - a: General activation. Added to all items that were presented during the current trial.
  - f: Filtering. This parameter captures the extent to which distractors remained in working memory.
- **Fixed parameters:**
  - b = 0
- **Default parameter links:**
  - c = identity; a = identity; f = logit
- **Default priors:**
  - a:
    - main: normal(3,1)
    - effects: normal(0,0.5)
  - c:
    - main: normal(3,1)
    - effects: normal(0,2)
  - f:
    - main: logistic(0,1)
    - effects: normal(0,1)

Version: custom**:**

- **Requirements:**
  - Provide names for variables specifying the number of responses in a set of response categories.
  - Specify activation sources for each response categories
  - Include at least an activation source "b" for all response categories
  - Predict the specified activation at least by a fixed intercept and any additional predictors from your data
- **Parameters:**
  - b: Background activation. Added to each response category. Fixed for scaling, necessary in all models.
- **Fixed parameters:**
  - b = 0
- **Default parameter links:**
  - =
- **Default priors:**

Value

An object of class bmmodel

Examples

```
data <- oberauer_lewandowsky_2019_e1

# initiate the model object
m3_model <- m3(
  resp_cats = c("corr", "other", "dist", "npl"),
  num_options = c("n_corr", "n_other", "n_dist", "n_npl"),
  choice_rule = "simple"
)

# specify the model formula including the activation formulas for each response category
m3_formula <- bmf(
  corr ~ b + a + c,
  other ~ b + a,
  dist ~ b + d,
  npl ~ b,
  c ~ 1 + cond + (1 + cond | ID),
  a ~ 1 + cond + (1 + cond | ID),
  d ~ 1 + (1 | ID)
)

# specify links for the model parameters
m3_model$links <- list(
  c = "log",
  a = "log",
  d = "log"
)

# check if the default priors are applied correctly
default_prior(m3_formula, data = data, model = m3_model)

# fit the model
m3_fit <- bmm(
  formula = m3_formula,
  data = data,
  model = m3_model,
  cores = 4
)

# print summary of the model
summary(m3_fit)
```

`m3dist` *Distribution functions for the Memory Measurement Model (M3)*

Description

Density and random generation functions for the memory measurement model. Please note that these functions are currently not vectorized.

Usage

```
dm3(x, pars, m3_model, act_funs = NULL, log = TRUE, ...)

rm3(n, size, pars, m3_model, act_funs = NULL, unpack = FALSE, ...)
```

Arguments

x	Integer vector of length K where K is the number of response categories and each value is the number of observed responses per category
pars	A named vector of parameters of the memory measurement model. Note: The fixed parameter b does not need to be provided - it will be automatically added from the model specification if missing.
m3_model	A bmmodel object specifying the m3 model that densities or random samples should be generated for
act_funs	A bmmformula object specifying the activation functions for the different response categories. This can be either: (1) Just the activation formulas (one for each response category), or (2) A full bmmformula including both activation formulas and other parameters. If a full formula is provided, only the formulas matching response categories will be extracted. The default will attempt to construct the standard activation functions for the "ss" and "cs" model version. For a custom m3 model you need to specify the act_funs argument manually.
log	Logical; if TRUE (default), values are returned on the log scale.
...	can be used to pass additional variables that are used in the activation functions, but not parameters of the model
n	Integer. Number of observations to generate data for
size	The total number of observations in all categories
unpack	Logical; if TRUE and n = 1, returns a named vector instead of a matrix. This allows automatic unpacking of response categories into separate columns when used with dplyr::reframe(). Default is FALSE for backward compatibility.

Value

dm3 gives the density of the memory measurement model, and rm3 gives the random generation function for the memory measurement model.

References

Oberauer, K., & Lewandowsky, S. (2019). Simple measurement models for complex working-memory tasks. Psychological Review, 126(6), 880–932. https://doi.org/10.1037/rev0000159

Examples

```
# Basic usage - b parameter is added automatically
model <- m3(
  resp_cats = c("corr", "other", "npl"),
  num_options = c(1, 4, 5),
  choice_rule = "simple",
```

```
  version = "ss"
)

# No need to provide b parameter
dm3(x = c(20, 10, 10), pars = c(a = 1, c = 2), m3_model = model)
rm3(n = 10, size = 100, pars = c(a = 1, c = 2), m3_model = model)

# Can also use full formula (activation formulas are extracted automatically)
full_formula <- bmf(
  corr ~ b + a + c,
  other ~ b + a,
  npl ~ b,
  a ~ 1,
  c ~ 1
)
rm3(
  n = 10, size = 100, pars = c(a = 1, c = 2),
  m3_model = model, act_funs = full_formula
)

## Not run:
# Use with dplyr::reframe() for automatic unpacking into columns
library(dplyr)
library(tibble)
param_grid <- expand.grid(a = c(0.5, 1, 1.5), c = c(1, 2, 3))

simulated_data <- param_grid |>
  rowwise() |>
  reframe(
    a = a,
    c = c,
    # unpack=TRUE returns named vector; wrap in as_tibble_row for auto-unpacking
    as_tibble_row(rm3(
      n = 1, size = 100, pars = c(a = a, c = c),
      m3_model = model, unpack = TRUE
    ))
  )
# Result has columns: a, c, corr, other, npl

## End(Not run)
```

`mixture2p` *Two-parameter mixture model by Zhang and Luck (2008).*

Description

Two-parameter mixture model by Zhang and Luck (2008).

Usage

```
mixture2p(resp_error, ...)
```

Arguments

resp_error	The name of the variable in the provided dataset containing the response error. The response Error should code the response relative to the to-be-recalled target in radians. You can transform the response error in degrees to radian using the deg2rad function.
...	used internally for testing, ignore it

Details

- **Domain:** Visual working memory
- **Task:** Continuous reproduction
- **Name:** Two-parameter mixture model by Zhang and Luck (2008).
- **Citation:**
  - Zhang, W., & Luck, S. J. (2008). Discrete fixed-resolution representations in visual working memory. Nature, 453(7192), 233-235
- **Requirements:**
  - The response vairable should be in radians and represent the angular error relative to the target
- **Parameters:**
  - mu1: Location parameter of the von Mises distribution for memory responses (in radians). Fixed internally to 0 by default.
  - kappa: Concentration parameter of the von Mises distribution
  - thetat: Mixture weight for target responses
- **Fixed parameters:**
  - mu1 = 0
  - mu2 = 0
  - kappa2 = -100
- **Default parameter links:**
  - mu1 = tan_half; kappa = log; thetat = logit
- **Default priors:**
  - mu1:
    - main: student_t(1, 0, 1)
  - kappa:
    - main: normal(2, 1)
    - effects: normal(0, 1)
  - thetat:
    - main: logistic(0, 1)

Value

An object of class bmmodel

Examples

```
# generate artificial data
dat <- data.frame(y = rmixture2p(n = 2000))

# define formula
ff <- bmmformula(kappa ~ 1, thetat ~ 1)

model <- mixture2p(resp_error = "y")

# fit the model
fit <- bmm(
  formula = ff,
  data = dat,
  model = model,
  cores = 4,
  iter = 500,
  backend = "cmdstanr"
)
```

`mixture2p_dist` *Distribution functions for the two-parameter mixture model (mixture2p)*

Description

Density, distribution, and random generation functions for the two-parameter mixture model with the location of `mu`, precision of memory representations `kappa` and probability of recalling items from memory `p_mem`.

Usage

```
dmixture2p(x, mu = 0, kappa = 5, p_mem = 0.6, log = FALSE)

pmixture2p(q, mu = 0, kappa = 7, p_mem = 0.8)

qmixture2p(p, mu = 0, kappa = 5, p_mem = 0.6)

rmixture2p(n, mu = 0, kappa = 5, p_mem = 0.6)
```

Arguments

`x`	Vector of observed responses
`mu`	Vector of locations
`kappa`	Vector of precision values
`p_mem`	Vector of probabilities for memory recall
`log`	Logical; if `TRUE`, values are returned on the log scale.

q	Vector of quantiles
p	Vector of probability
n	Number of observations to generate data for

Value

dmixture2p gives the density of the two-parameter mixture model, pmixture2p gives the cumulative distribution function of the two-parameter mixture model, qmixture2p gives the quantile function of the two-parameter mixture model, and rmixture2p gives the random generation function for the two-parameter mixture model.

References

Zhang, W., & Luck, S. J. (2008). Discrete fixed-resolution representations in visual working memory. Nature, 453.

Examples

```
# generate random samples from the mixture2p model and overlay the density
r <- rmixture2p(10000, mu = 0, kappa = 4, p_mem = 0.8)
x <- seq(-pi, pi, length.out = 10000)
d <- dmixture2p(x, mu = 0, kappa = 4, p_mem = 0.8)
hist(r, breaks = 60, freq = FALSE)
lines(x, d, type = "l", col = "red")
```

---

mixture3p *Three-parameter mixture model by Bays et al (2009).*

Description

Three-parameter mixture model by Bays et al (2009).

Usage

```
mixture3p(resp_error, nt_features, set_size, regex = FALSE, ...)
```

Arguments

resp_error	The name of the variable in the dataset containing the response error. The response error should code the response relative to the to-be-recalled target in radians. You can transform the response error in degrees to radians using the deg2rad function.
nt_features	A character vector with the names of the non-target feature values. The non_target feature values should be in radians and centered relative to the target. Alternatively, if regex=TRUE, a regular expression can be used to match the non-target feature columns in the dataset.

set_size	Name of the column containing the set size variable (if set_size varies) or a numeric value for the set_size, if the set_size is fixed.
regex	Logical. If TRUE, the nt_features argument is interpreted as a regular expression to match the non-target feature columns in the dataset.
...	used internally for testing, ignore it

Details

- **Domain:** Visual working memory
- **Task:** Continuous reproduction
- **Name:** Three-parameter mixture model by Bays et al (2009).
- **Citation:**
  - Bays, P. M., Catalao, R. F. G., & Husain, M. (2009). The precision of visual working memory is set by allocation of a shared resource. Journal of Vision, 9(10), 1-11
- **Requirements:**
  - The response vairable should be in radians and represent the angular error relative to the target
- The non-target features should be in radians and be centered relative to the target
- **Parameters:**
  - mu1: Location parameter of the von Mises distribution for memory responses (in radians). Fixed internally to 0 by default.
  - kappa: Concentration parameter of the von Mises distribution
  - thetat: Mixture weight for target responses
  - thetant: Mixture weight for non-target responses
- **Fixed parameters:**
  - mu1 = 0
  - mu2 = 0
  - kappa2 = -100
- **Default parameter links:**
  - mu1 = tan_half; kappa = log; thetat = softmax; thetant = softmax
- **Default priors:**
  - mu1:
    - main: student_t(1, 0, 1)
  - kappa:
    - main: normal(2, 1)
    - effects: normal(0, 1)
  - thetat:
    - main: logistic(0, 1)
  - thetant:
    - main: logistic(0, 1)

Value

An object of class bmmodel

Examples

```
# generate artificial data from the Bays et al (2009) 3-parameter mixture model
dat <- data.frame(
  y = rmixture3p(n = 2000, mu = c(0, 1, -1.5, 2)),
  nt1_loc = 1,
  nt2_loc = -1.5,
  nt3_loc = 2
)

# define formula
ff <- bmmformula(
  kappa ~ 1,
  thetat ~ 1,
  thetant ~ 1
)

# specify the 3-parameter model with explicit column names for non-target features
model1 <- mixture3p(resp_error = "y", nt_features = paste0("nt", 1:3, "_loc"), set_size = 4)

# fit the model
fit <- bmm(
  formula = ff,
  data = dat,
  model = model1,
  cores = 4,
  iter = 500,
  backend = "cmdstanr"
)

# alternatively specify the 3-parameter model with a regular expression to match non-target features
# this is equivalent to the previous call, but more concise
model2 <- mixture3p(resp_error = "y", nt_features = "nt.*_loc", set_size = 4, regex = TRUE)

# fit the model
fit <- bmm(
  formula = ff,
  data = dat,
  model = model2,
  cores = 4,
  iter = 500,
  backend = "cmdstanr"
)
```

`mixture3p_dist` *Distribution functions for the three-parameter mixture model (mixture3p)*

Description

Density, distribution, and random generation functions for the three-parameter mixture model with the location of `mu`, precision of memory representations `kappa`, probability of recalling items from memory `p_mem`, and probability of recalling non-targets `p_nt`.

Usage

```
dmixture3p(
  x,
  mu = c(0, 2, -1.5),
  kappa = 5,
  p_mem = 0.6,
  p_nt = 0.2,
  log = FALSE
)

pmixture3p(q, mu = c(0, 2, -1.5), kappa = 5, p_mem = 0.6, p_nt = 0.2)

qmixture3p(p, mu = c(0, 2, -1.5), kappa = 5, p_mem = 0.6, p_nt = 0.2)

rmixture3p(n, mu = c(0, 2, -1.5), kappa = 5, p_mem = 0.6, p_nt = 0.2)
```

Arguments

`x`	Vector of observed responses
`mu`	Vector of locations. First value represents the location of the target item and any additional values indicate the location of non-target items.
`kappa`	Vector of precision values
`p_mem`	Vector of probabilities for memory recall
`p_nt`	Vector of probabilities for swap errors
`log`	Logical; if `TRUE`, values are returned on the log scale.
`q`	Vector of quantiles
`p`	Vector of probability
`n`	Number of observations to generate data for

Value

`dmixture3p` gives the density of the three-parameter mixture model, `pmixture3p` gives the cumulative distribution function of the two-parameter mixture model, `qmixture3p` gives the quantile function of the two-parameter mixture model, and `rmixture3p` gives the random generation function for the two-parameter mixture model.

References

Bays, P. M., Catalao, R. F. G., & Husain, M. (2009). The precision of visual working memory is set by allocation of a shared resource. Journal of Vision, 9(10), 7.

Examples

```
# generate random samples from the mixture3p model and overlay the density
r <- rmixture3p(10000, mu = c(0, 2, -1.5), kappa = 4, p_mem = 0.6, p_nt = 0.2)
x <- seq(-pi, pi, length.out = 10000)
d <- dmixture3p(x, mu = c(0, 2, -1.5), kappa = 4, p_mem = 0.6, p_nt = 0.2)
hist(r, breaks = 60, freq = FALSE)
lines(x, d, type = "l", col = "red")
```

---

oberauer_lewandowsky_2019_e1

Data from Experiment 1 reported by Oberauer & Lewandowsky (2019)

Description

Raw data of 40 subjects that completed a verbal memory recall task in three different conditions using different types of distractor words.

Usage

```
oberauer_lewandowsky_2019_e1
```

Format

`oberauer_lewandowsky_2019_e1`**:**

A data frame with 120 rows and 10 columns:

ID Integer uniquely identifying each subject

cond Factor sperating the three experimental conditions: `new distractors` refers to new words being used as distractors, `old reordered` refers to the to be remembered words being the distractors, but reordered relative to the serial position, `old same` refers to the to be remebered words being the distractors, and appearing in the same order as the to be remembered words.

corr The frequency a subject recalled the correct item

other The frequency a subject recalled one of the other to be remebered words

dist The frequency a subject recalled one of the distractors

npl The frequency a subject recalled a not-presented lure (NPL), that is a word that was not presented during a trial

n_corr, n_other, n_dist, n_npl The number of candidataes in each of the response categories

oberauer_lin_2017 *Data from Experiment 1 reported by Oberauer & Lin (2017)*

Description

Raw data of 19 subjects that completed a continuous reproduction task with set size 1 to 8 reported by Oberauer & Lin (2017).

Usage

```
oberauer_lin_2017
```

Format

`oberauer_lin_2017`**:**

A data frame with 15,200 rows and 19 columns:

ID Integer uniquely identifying different subjects

session Session number

trial Trial number within each session

set_size The set_size of the data in this row

dev_rad The response error, that is the difference between the response given and the target color in radians.

col_nt1, col_nt2, col_nt3, col_nt4, col_nt5, col_nt6, col_nt7 The non-target items' color value relative to the target.

dist_nt1, dist_nt2, dist_nt3, dist_nt4, dist_nt5, dist_nt6, dist_nt7, dist_nt8 The spatial distance between all non-target items and the target item in radians.

Source

https://osf.io/m4shu

parameters *Get parameter information for a bmm model*

Description

Returns a data frame with information about the model parameters, including their descriptions, whether they are fixed, their link functions, and optionally their default priors.

Usage

```
parameters(x, ...)

## S3 method for class 'bmmodel'
parameters(x, formula = NULL, ...)

## S3 method for class 'bmmfit'
parameters(x, ...)
```

Arguments

x	A bmmodel object (e.g., sdm(resp_error = "y")) or a bmmfit object (a fitted model returned by bmm).
...	Additional arguments (currently unused).
formula	An optional bmmformula object. Only relevant for M3 custom models, where additional parameters are discovered from the formula. Ignored for all other models.

Value

A data frame of class bmm_parameters with one row per parameter and columns: parameter, description, fixed, value, and link.

Examples

```
# For an unfitted model
parameters(sdm(resp_error = "y"))

# For an M3 model
parameters(m3(
  resp_cats = c("corr", "other", "npl"),
  num_options = c(1, 4, 5),
  version = "ss"
))
```

---

pp_check.bmmfit *Posterior predictive check for bmmfit objects*

Description

For models where brms provides pp_check support, this method delegates to brms::pp_check(). For models with multinomial families (e.g., the m3 model), brms's pp_check is unavailable; this method dispatches to a model-specific visualisation instead.

Usage

```
## S3 method for class 'bmmfit'
pp_check(object, type = "dens_overlay", ndraws = NULL, group = NULL, ...)
```

Arguments

object	A bmmfit object returned by bmm().
type	Character. Type of pp_check (default "dens_overlay"). For non-multinomial models, passed to brms::pp_check(). When group is specified, the grouped variant (e.g., "dens_overlay_grouped") is auto-selected if available. Multinomial models produce a response proportion profile regardless of the value supplied.
ndraws	Integer. Number of posterior draws. Defaults to 100 for multinomial models; otherwise passed to brms::pp_check().
group	Character. Optional grouping variable for faceting. For non-multinomial models, passed to brms::pp_check(); when specified, the grouped variant of type (e.g., "dens_overlay_grouped") is auto-selected if available. For multinomial models, facets by the named predictor.
...	Additional arguments forwarded to brms::pp_check() (non-multinomial) or to brms::posterior_predict() (multinomial). For multinomial models, probs (numeric vector of length 2, default c(0.025, 0.975)) controls the credible interval. Both model types accept re_formula (e.g., re_formula = NA to predict at the population level, excluding random effects).

Details

For **multinomial models**, the plot mirrors the bayesplot ppc_bars style: observed proportions are shown as bars and posterior predictive medians with credible intervals are shown as point-ranges, using the bayesplot default colour scheme and theme.

Value

For multinomial models, a ggplot2 object. For other models, the result of brms::pp_check().

See Also

brms::pp_check()

---

rejection_sampling *Rejection Sampling*

Description

Performs rejection sampling to generate samples from a target distribution.

Usage

```
rejection_sampling(n, f, max_f, proposal_fun, ...)
```

Arguments

n	Integer. The number of samples to generate.
f	Function. The target density function from which to sample.
max_f	Numeric. The maximum value of the target density function f.
proposal_fun	Function. A function that generates samples from the proposal distribution.
...	Additional arguments to be passed to the target density function f.

Value

A numeric vector of length n containing samples from the target distribution.

Examples

```
target_density <- function(x) brms::dvon_mises(x, mu = 0, kappa = 10)
proposal <- function(n) runif(n, min = -pi, max = pi)
samples <- rejection_sampling(10000, target_density, max_f = target_density(0), proposal)
hist(samples, freq = FALSE)
curve(target_density, col = "red", add = TRUE)
```

---

restructure.bmmfit *Restructure Old* bmmfit *Objects*

Description

Restructure old bmmfit objects to work with the latest **bmm** version. This function is called internally when applying post-processing methods.

Usage

```
## S3 method for class 'bmmfit'
restructure(x, ...)
```

Arguments

x	An object of class bmmfit.
...	Currently ignored.

Value

A bmmfit object compatible with the latest version of **bmm** and **brms**.

Examples

```
# Load an old bmmfit object
old_fit <- readRDS("bmmfit_old.rds")
new_fit <- restructure(old_fit)
```

`sdm` *Signal Discrimination Model (SDM) by Oberauer (2023)*

Description

Signal Discrimination Model (SDM) by Oberauer (2023)

Usage

```
sdm(resp_error, version = "simple", ...)

sdmSimple(resp_error, version = "simple", ...)
```

Arguments

`resp_error`	The name of the variable in the dataset containing the response error. The response error should code the response relative to the to-be-recalled target in radians. You can transform the response error in degrees to radians using the `deg2rad` function.
`version`	Character. The version of the model to use. Currently only "simple" is supported.
`...`	used internally for testing, ignore it

Details

see the online article for a detailed description of the model and how to use it. * **Domain:** Visual working memory

- **Task:** Continuous reproduction
- **Name:** Signal Discrimination Model (SDM) by Oberauer (2023)
- **Citation:**
  - Oberauer, K. (2023). Measurement models for visual working memory - A factorial model comparison. Psychological Review, 130(3), 841-852
- **Version:** simple
- **Requirements:**
  - The response variable should be in radians and represent the angular error relative to the target
- **Parameters:**
  - `mu`: Location parameter of the SDM distribution (in radians; by default fixed internally to 0)
  - `c`: Memory strength parameter of the SDM distribution
  - `kappa`: Precision parameter of the SDM distribution
- **Fixed parameters:**

  - mu = 0
- **Default parameter links:**
  - mu = tan_half; c = log; kappa = log
- **Default priors:**
  - mu:
    * main: student_t(1, 0, 1)
  - kappa:
    * main: student_t(5, 1.75, 0.75)
    * effects: normal(0, 1)
  - c:
    * main: student_t(5, 2, 0.75)
    * effects: normal(0, 1)

Value

An object of class bmmodel

Examples

```
# simulate data from the model
dat <- data.frame(y = rsdm(n = 1000, c = 4, kappa = 3))

# specify formula
ff <- bmf(
  c ~ 1,
  kappa ~ 1
)

# specify the model
fit <- bmm(
  formula = ff,
  data = dat,
  model = sdm(resp_error = "y"),
  cores = 4,
  backend = "cmdstanr"
)
```

SDMdist *Distribution functions for the Signal Discrimination Model (SDM)*

Description

Density, distribution function, and random generation for the Signal Discrimination Model (SDM) Distribution with location mu, memory strength c, and precision kappa. Currently only a single activation source is supported.

Usage

```
dsdm(x, mu = 0, c = 3, kappa = 3.5, log = FALSE, parametrization = "sqrtexp")

psdm(
  q,
  mu = 0,
  c = 3,
  kappa = 3.5,
  lower.tail = TRUE,
  log.p = FALSE,
  lower.bound = -pi,
  parametrization = "sqrtexp"
)

qsdm(p, mu = 0, c = 3, kappa = 3.5, parametrization = "sqrtexp")

rsdm(n, mu = 0, c = 3, kappa = 3.5, parametrization = "sqrtexp")
```

Arguments

x	Vector of quantiles
mu	Vector of location values in radians
c	Vector of memory strength values
kappa	Vector of precision values
log	Logical; if TRUE, values are returned on the log scale.
parametrization	Character; either "bessel" or "sqrtexp" (default). See the online article for details on the parameterization.
q	Vector of quantiles
lower.tail	Logical; If TRUE (default), return P(X <= x). Else, return P(X > x)
log.p	Logical; if TRUE, probabilities are returned on the log scale.
lower.bound	Numeric; Lower bound of integration for the cumulative distribution
p	Vector of probabilities
n	Number of observations to sample

Details

Parametrization

See the online article for details on the parameterization. Oberauer (2023) introduced the SDM with the bessel parametrization. The sqrtexp parametrization is the default in the bmm package for numerical stability and efficiency. The two parametrizations are related by the functions `c_bessel2sqrtexp()` and `c_sqrtexp2bessel()`.

The cumulative distribution function

Since responses are on the circle, the cumulative distribution function requires you to choose a lower bound of integration. The default is $-\pi$, as for the brms::pvon_mises() function but you can choose

any value in the argument lower_bound of psdm. Another useful choice is the mean of the response distribution minus π, e.g. lower_bound = mu-pi. This is the default in circular::pvonmises(), and it ensures that 50% of the cumulative probability mass is below the mean of the response distribution.

Value

dsdm gives the density, psdm gives the distribution function, qsdm gives the quantile function, rsdm generates random deviates, and .dsdm_integrate is a helper function for calculating the density of the SDM distribution.

References

Oberauer, K. (2023). Measurement models for visual working memory - A factorial model comparison. Psychological Review, 130(3), 841–852

Examples

```
# plot the density of the SDM distribution
x <- seq(-pi, pi, length.out = 10000)
plot(x, dsdm(x, 0, 2, 3),
  type = "l", xlim = c(-pi, pi), ylim = c(0, 1),
  xlab = "Angle error (radians)",
  ylab = "density",
  main = "SDM density"
)
lines(x, dsdm(x, 0, 9, 1), col = "red")
lines(x, dsdm(x, 0, 2, 8), col = "green")
legend("topright", c(
  "c=2, kappa=3.0, mu=0",
  "c=9, kappa=1.0, mu=0",
  "c=2, kappa=8, mu=1"
),
col = c("black", "red", "green"), lty = 1, cex = 0.8
)

# plot the cumulative distribution function of the SDM distribution
p <- psdm(x, mu = 0, c = 3.1, kappa = 5)
plot(x, p, type = "l")

# generate random deviates from the SDM distribution and overlay the density
r <- rsdm(10000, mu = 0, c = 3.1, kappa = 5)
d <- dsdm(x, mu = 0, c = 3.1, kappa = 5)
hist(r, breaks = 60, freq = FALSE)
lines(x, d, type = "l", col = "red")
```

softmax *Softmax function and its inverse*

Description

softmax returns the value of the softmax function softmaxinv returns the value of the inverse-softmax function

Usage

```
softmax(eta, lambda = 1)

softmaxinv(p, lambda = 1, ref_position = length(p), ref_value = 0)
```

Arguments

eta	A numeric vector input
lambda	Tuning parameter (a single positive value)
p	A probability vector (i.e., numeric vector of non-negative values that sum to one)
ref_position	The reference position that should be used to calculate the inverse softmax function. The default is the last position.
ref_value	The value the reference position will be set to. The default is 0.

Details

The softmax function is a bijective function that maps a real vector with length m to a probability vector with length m with all non-zero probabilities. The present functions define the softmax function and its inverse, both with a tuning parameter.

The current functions define the softmax as:

$$P(\eta_i) = \frac{e^{\lambda \eta_i}}{\sum_{j=1}^{m} e^{\lambda \eta_j}}$$

Value

Value of the softmax function or its inverse

Examples

```
softmax(5:7)
softmaxinv(softmax(5:7), ref_position = 1, ref_value = 5)
```

stancode.bmmformula *Generate Stan code for bmm models*

Description

Given the model, the data and the formula for the model, this function will return the combined stan code generated by bmm and brms

Usage

```
## S3 method for class 'bmmformula'
stancode(object, data, model, prior = NULL, ...)
```

Arguments

object	A bmmformula object
data	An object of class data.frame, containing data of all variables used in the model. The names of the variables must match the variable names passed to the bmmodel object for required argurments.
model	A description of the model to be fitted. This is a call to a bmmodel such as mixture3p() function. Every model function has a number of required arguments which need to be specified within the function call. Call supported_models() to see the list of supported models and their required arguments
prior	One or more brmsprior objects created by brms::set_prior() or related functions and combined using the c method or the + operator. See also default_prior() for more help. Not necessary for the default model fitting, but you can provide prior constraints to model parameters
...	Further arguments passed to brms::stancode(). See the description of brms::stancode() for more details

Value

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

See Also

supported_models(), brms::stancode()

Examples

```
scode1 <- stancode(bmf(c ~ 1, kappa ~ 1),
  data = oberauer_lin_2017,
  model = sdm(resp_error = "dev_rad")
)
cat(scode1)
```

standata.bmmformula *Stan data for* bmm *models*

Description

Given the model, the data and the formula for the model, this function will return the combined stan data generated by bmm and brms

Usage

```
## S3 method for class 'bmmformula'
standata(object, data, model, ...)
```

Arguments

object	A bmmformula object
data	An object of class data.frame, containing data of all variables used in the model. The names of the variables must match the variable names passed to the bmmodel object for required argurments.
model	A description of the model to be fitted. This is a call to a bmmodel such as mixture3p() function. Every model function has a number of required arguments which need to be specified within the function call. Call supported_models() to see the list of supported models and their required arguments
...	Further arguments passed to brms::standata(). See the description of brms::standata() for more details

Value

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

See Also

supported_models(), brms::standata()

Examples

```
sdata1 <- standata(bmf(c ~ 1, kappa ~ 1),
  data = oberauer_lin_2017,
  model = sdm(resp_error = "dev_rad")
)
str(sdata1)
```

summary.bmmfit *Create a summary of a fitted model represented by a* bmmfit *object*

Description

Create a summary of a fitted model represented by a bmmfit object

Usage

```
## S3 method for class 'bmmfit'
summary(
  object,
  priors = FALSE,
  prob = 0.95,
  robust = FALSE,
  mc_se = FALSE,
  ...,
  backend = "bmm"
)
```

Arguments

object	An object of class brmsfit.
priors	Logical; Indicating if priors should be included in the summary. Default is FALSE.
prob	A value between 0 and 1 indicating the desired probability to be covered by the uncertainty intervals. The default is 0.95.
robust	If FALSE (the default) the mean is used as the measure of central tendency and the standard deviation as the measure of variability. If TRUE, the median and the median absolute deviation (MAD) are applied instead.
mc_se	Logical; Indicating if the uncertainty in Estimate caused by the MCMC sampling should be shown in the summary. Defaults to FALSE.
...	Other potential arguments
backend	Choose whether to display the *bmm* summary method (default), or to display the *brms* summary method.

Value

A list of class bmmsummary containing the summary of the model's parameters, the model formula, the model, and the data used to fit the model.

Note

You can turn off the color output by setting the option options(bmm.color_summary = FALSE) or bmm_options(color_summary = FALSE)

See Also

summary.brmsfit

Examples

```
# generate artificial data from the Signal Discrimination Model
dat <- data.frame(y = rsdm(2000))

# define formula
ff <- bmmformula(c ~ 1, kappa ~ 1)

# fit the model
fit <- bmm(
  formula = ff,
  data = dat,
  model = sdm(resp_error = "y"),
  cores = 4,
  backend = "cmdstanr"
)

# summary of the model
summary(fit)
```

---

supported_models *Measurement models available in* bmm

Description

Measurement models available in bmm

Usage

```
supported_models(print_call = TRUE)
```

Arguments

print_call	Logical; If TRUE (default), the function will print information about how each model function should be called and its required arguments. If FALSE, the function will return a character vector with the names of the available models

Value

A character vector of measurement models available in bmm

Examples

```
supported_models()
```

`update.bmmfit` *Update a bmm model*

Description

Update an existing bmm mode. This function calls `brms::update.brmsfit()`, but it applies the necessary bmm postprocessing to the model object before and after the update.

Usage

```
## S3 method for class 'bmmfit'
update(object, formula., newdata = NULL, recompile = NULL, ...)
```

Arguments

`object`	An object of class `bmmfit`
`formula.`	A `bmmformula()`. If missing, the original formula is used. Currently you have to specify a full `bmmformula`
`newdata`	An optional data frame containing the variables in the model
`recompile`	Logical, indicating whether the Stan model should be recompiled. If NULL (the default), update tries to figure out internally, if recompilation is necessary. Setting it to FALSE will cause all Stan code changing arguments to be ignored.
`...`	Further arguments passed to `brms::update.brmsfit()`

Details

When updating a brmsfit created with the cmdstanr backend in a different R session, a recompilation will be triggered because by default, cmdstanr writes the model executable to a temporary directory. To avoid that, set option "cmdstanr_write_stan_file_dir" to a nontemporary path of your choice before creating the original bmmfit.

For more information and examples, see `brms::update.brmsfit()`

Value

An updated `bmmfit` object refit to the new data and/or formula

Examples

```
# generate artificial data from the Signal Discrimination Model
# generate artificial data from the Signal Discrimination Model
dat <- data.frame(y = rsdm(2000))

# define formula
ff <- bmf(c ~ 1, kappa ~ 1)

# fit the model
fit <- bmm(
```

```
  formula = ff,
  data = dat,
  model = sdm(resp_error = "y"),
  cores = 4,
  backend = "cmdstanr"
)

# update the model
fit <- update(fit, newdata = data.frame(y = rsdm(2000, kappa = 5)))
```

`validate_fast_guesses` *Test if fast contaminants show random guessing behavior*

Description

Uses Bayesian Beta-Binomial conjugate analysis to test whether fast flagged contaminants show random guessing (~50% accuracy for 2AFC). The test computes the posterior distribution for the proportion of "upper" responses and uses a Savage-Dickey Bayes Factor to quantify evidence for or against the guessing hypothesis.

Usage

```
validate_fast_guesses(
  contam_flag,
  rt_data,
  response,
  threshold_type = c("quantile", "absolute"),
  rt_threshold = 0.25,
  prior_alpha = 1,
  prior_beta = 1,
  guess_prob = 0.5,
  credible_mass = 0.95
)
```

Arguments

`contam_flag`	Logical vector indicating which trials were flagged as contaminants
`rt_data`	Numeric vector of reaction times (in seconds)
`response`	Response data in any format accepted by `.convert_response_to_upper()` (numeric 0/1, logical, character, factor)
`threshold_type`	Character. How to interpret `rt_threshold`: • `"quantile"` (default): Use rt_threshold as quantile (0-1) • `"absolute"`: Use rt_threshold as absolute RT in seconds
`rt_threshold`	Numeric. Threshold for defining "fast" trials. Interpretation depends on `threshold_type`:

- If `threshold_type = "quantile"` (default): Quantile of RT distribution (e.g., 0.25 = 25th percentile). Default 0.25.
- If `threshold_type = "absolute"`: Absolute RT value in seconds (e.g., 0.25 = 250ms).

`prior_alpha, prior_beta`
: Numeric. Parameters for Beta prior distribution. Default 1,1 gives uniform prior. Values > 1 express prior belief about response proportions.

`guess_prob`
: Numeric. Null hypothesis value for guessing probability. Default 0.5 (equal probability of upper/lower responses).

`credible_mass`
: Numeric. Probability mass for Highest Density Interval. Default 0.95 for 95% HDI. Common alternatives: 0.90, 0.99.

Details

The function performs a Bayesian test using the Beta-Binomial conjugate prior-posterior relationship. With a Beta(alpha, beta) prior and observing n_upper "upper" responses out of n_tested trials, the posterior is:

Beta(alpha + n_upper, beta + n_lower)

The Savage-Dickey Bayes Factor compares the posterior and prior densities at the null hypothesis value (default 0.5):

BF_01 = posterior_density(guess_prob) / prior_density(guess_prob)

Evidence categories follow Jeffreys (1961) scale:

- BF > 10: Strong evidence for guessing
- BF > 3: Moderate evidence for guessing
- BF > 1: Anecdotal evidence for guessing
- BF < 1/3: Moderate evidence against guessing
- BF < 1/10: Strong evidence against guessing

Note: This function can be used as a standalone validation step after obtaining contamination probabilities from `flag_contaminant_rts()`.

Value

List with components:

- `method`: "bayesian"
- `prop_upper`: Observed proportion of upper responses
- `hdi_lower`, `hdi_upper`: 95% Highest Density Interval bounds
- `bf_01`: Bayes Factor for H0 (guessing) vs H1 (non-random)
- `guess_in_hdi`: Logical, whether `guess_prob` is in HDI
- `bf_evidence`: Character, evidence category on Jeffreys scale
- `posterior_alpha`, `posterior_beta`: Posterior Beta parameters
- `n_tested`: Number of fast flagged trials tested

- `rt_threshold`: Actual RT threshold value used (in seconds)
- `threshold_type`: Type of threshold used ("quantile" or "absolute")
- `credible_mass`: Credible mass used for HDI computation
- `mean_rt_tested`: Mean RT of tested trials

References

Jeffreys, H. (1961). Theory of Probability (3rd ed.). Oxford University Press.

See Also

`flag_contaminant_rts()` for obtaining contamination probabilities

Examples

```
## Not run:
# Simulate data with random guessing on fast trials
set.seed(123)
n <- 100
rt <- c(runif(20, 0.15, 0.30), rgamma(80, 5, 10))
response <- c(rbinom(20, 1, 0.5), rbinom(80, 1, 0.7))
contam_flag <- rt < 0.35

# Test using quantile threshold (default, adaptive)
result1 <- validate_fast_guesses(
  contam_flag = contam_flag,
  rt_data = rt,
  response = response,
  threshold_type = "quantile",
  rt_threshold = 0.30 # 30th percentile
)

# Test using absolute threshold (fixed RT)
result2 <- validate_fast_guesses(
  contam_flag = contam_flag,
  rt_data = rt,
  response = response,
  threshold_type = "absolute",
  rt_threshold = 0.30 # 300ms
)

print(result1$bf_01) # Bayes Factor
print(result1$bf_evidence) # Evidence category
print(result1$guess_in_hdi) # Is 0.5 in 95% HDI?
print(result1$threshold_type) # "quantile"

## End(Not run)
```

wrap *Wrap angles that extend beyond (-pi;pi)*

Description

On the circular space, angles can be only in the range (-pi;pi or -180;180). When subtracting angles, this can result in values outside of this range. For example, when calculating the difference between a value of 10 degrees minus 340 degrees, this results in a difference of 330 degrees. However, the true difference between these two values is -30 degrees. This function wraps such values, so that they occur in the circle

Usage

```
wrap(x, radians = TRUE)
```

Arguments

x	A numeric vector, matrix or data.frame of angles to be wrapped. In radians (default) or degrees.
radians	Logical. Is x in radians (default=TRUE) or degrees (FALSE)

Value

An object of the same type as x

Examples

```
x <- runif(1000, -pi, pi)
y <- runif(1000, -pi, pi)
diff <- x - y
hist(diff)
wrapped_diff <- wrap(x - y)
hist(wrapped_diff)
```

zhang_luck_2008 *Data from Experiment 2 reported by Zhang & Luck (2008)*

Description

Raw data of 8 subjects for the response error in a continuous reproduction task with set size 1, 2, 3, and 6 reported by Zhang & Luck (2008).

Usage

```
zhang_luck_2008
```

Format

`zhang_luck_2008`**:**

A data frame with 4,000 rows and 9 columns:

subID Integer uniquely identifying different subjects

trial Trial identifyier

setsize The set_size of the data in this row

response_error The response error, that is the difference between the response given and the target color in radians.

col_lure1, col_Lure2, col_Lure3, col_Lure4, col_Lure5 Color value of the lure items coded relative to the target color.

Source

https://www.nature.com/articles/nature06860

Index