

Package ‘bmm’

July 24, 2025

Title Easy and Accessible Bayesian Measurement Models Using 'brms'

Version 1.2.0

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 (2023) <[doi:10.31234/osf.io/umt57](https://doi.org/10.31234/osf.io/umt57)>.

License GPL-2

URL <https://github.com/venpopov/bmm>, <https://venpopov.github.io/bmm/>

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

Depends R (>= 3.6.0)

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

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

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

Config/testthat/edition 3

Config/Needs/website rmarkdown

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

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 2025-07-24 15:20:02 UTC

Contents

bmm-package	3
apply_links	3
bmm	4
bmmformula	7
bmm_options	9
calc_error_relative_to_nontargets	10
circle_transform	11
construct_m3_act_funs	12
c_parametrizations	12
default_prior.bmmformula	13
fit_info	14
imm	15
IMMdist	20
k2sd	22
m3	23
m3dist	26
mixture2p	28
mixture2p_dist	29
mixture3p	31
mixture3p_dist	33
oberauer_lewandowsky_2019_e1	34
oberauer_lin_2017	35
rejection_sampling	36
restructure.bmmfit	36
sdm	37
SDMdist	39
softmax	41
stancode.bmmformula	42
standata.bmmformula	43
summary.bmmfit	44
supported_models	45
update.bmmfit	46
wrap	47
zhang_luck_2008	48

Index	49
--------------	-----------

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 (2023) [doi:10.31234/osf.io/umt57](https://doi.org/10.31234/osf.io/umt57).

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

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 bmmmodels.

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 <code>bmmformula</code> . A symbolic description of the model to be fitted.
data	An object of class <code>data.frame</code> , containing data of all variables used in the model. The names of the variables must match the variable names passed to the <code>bmm</code> object for required arguments.
model	A description of the model to be fitted. This is a call to a <code>bmm</code> model such as <code>mixture3p()</code> 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 <code>brmsprior</code> objects created by <code>brms::set_prior()</code> or related functions and combined using the <code>c</code> method or the <code>+</code> 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 <code>TRUE</code> , the data will be sorted by the predictor variables for faster sampling. If <code>FALSE</code> , the data will not be sorted, but sampling will be slower. If "check" (the default), <code>bmm()</code> 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 <code>global_options(bmm.sort_data = TRUE/FALSE/"check")</code> or via <code>bmm_options(sort_data)</code>
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 <code>refresh = 0</code> to turn this off as well. If using <code>backend = "rstan"</code> you can also set <code>open_progress = FALSE</code> to prevent opening additional progress bars.
backend	Character. The backend to use for fitting the model. Can be "rstan" or "cmdstanr". If <code>NULL</code> (the default), "cmdstanr" will be used if the <code>cmdstanr</code> package is installed, otherwise "rstan" will be used. You can set the default backend using <code>global_options(brms.backend = "rstan"/"cmdstanr")</code>
file	Either <code>NULL</code> or a character string. If a string, the fitted model object is saved via saveRDS in a file named after the string. The <code>.rds</code> extension is added automatically. If the file already exists, <code>bmm</code> will load and return the saved model object. Unless you specify the <code>file_refit</code> 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 <code>bmmfit</code> 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 <code>file</code> argument is re-used. Can be set globally for the current R session via the <code>"bmm.file_refit"</code>

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:

- `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 how 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 bmmmodel***Description**

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

Usage

```
bmmformula(...)
```

```
bmf(...)
```

Arguments

... Formulas for predicting a bmmmodel 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 difference 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 μ parameter of the model. In general, measurement models do not all have a μ parameter, therefore it is more straightforward to explicitly 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 explicitly 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 $n \times m$ 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	<i>Convert degrees to radians or radians to degrees.</i>
------------------	--

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)
```

`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 `bmmmodel` object is passed then it will print the available model versions.

Usage

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

Arguments

<code>model</code>	A <code>bmmmodel</code> 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.
<code>warnings</code>	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)
```

`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_sqrtextp2bessel(c, kappa)
```

```
c_bessel2sqrtextp(c, kappa)
```

Arguments

c	Vector of memory strength values
kappa	Vector of precision values

Details

c_bessel2sqrtextp converts the memory strength parameter (c) from the `bessel` parametrization to the `sqrtextp` parametrization, c_sqrtextp2bessel converts from the `sqrtextp` parametrization to the `bessel` parametrization.

See [the online article](#) for details on the parameterization. The `sqrtextp` parametrization is the default in the `bmm` package.

Value

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

Examples

```
c_bessel <- c_sqrtextp2bessel(c = 4, kappa = 3)
c_sqrtextp <- c_bessel2sqrtextp(c = c_bessel, kappa = 3)
```

```
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 used if no user-specified 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 bmmmodel object for required arguments.
model	A description of the model to be fitted. This is a call to a bmmmodel 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")
)
```

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")
```

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:**
 - $\mu_1 = 0$
 - $\mu_2 = 0$
 - $\kappa_2 = -100$
- **Default parameter links:**
 - $\mu_1 = \tan_half$; $\kappa = \log$; $a = \log$; $c = \log$; $s = \log$
- **Default priors:**
 - μ_1 :
 - * main: `student_t(1, 0, 1)`
 - κ :
 - * 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 variable 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:**
 - μ_1 : Location parameter of the von Mises distribution for memory responses (in radians). Fixed internally to 0 by default.
 - κ : Concentration parameter of the von Mises distribution
 - c : Context activation
 - s : Spatial similarity gradient
- **Fixed parameters:**
 - $\mu_1 = 0$
 - $\mu_2 = 0$
 - $\kappa_2 = -100$
- **Default parameter links:**
 - $\mu_1 = \tan_half$; $\kappa = \log$; $c = \log$; $s = \log$
- **Default priors:**
 - μ_1 :
 - * 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 variable 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 `bmmmodel`

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 μ , strength of cue- dependent activation c , strength of cue-independent activation a , the generalization gradient s , and the precision of memory representations κ .

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)

# calculate 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:**
 - $c = \text{identity}$; $a = \text{identity}$
- **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:**
 - $c = \text{identity}$; $a = \text{identity}$; $f = \text{logit}$
- **Default priors:**
 - a:
 - * main: $\text{normal}(3,1)$
 - * effects: $\text{normal}(0,0.5)$
 - c:
 - * main: $\text{normal}(3,1)$
 - * effects: $\text{normal}(0,2)$
 - f:
 - * main: $\text{logistic}(0,1)$
 - * effects: $\text{normal}(0,1)$

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 = construct_m3_act_funs(m3_model, warnings = FALSE),
  log = TRUE,
  ...
)

rm3(
  n,
  size,
  pars,
  m3_model,
  act_funs = construct_m3_act_funs(m3_model, warnings = FALSE),
  ...
)
```

Arguments

<code>x</code>	Integer vector of length K where K is the number of response categories and each value is the number of observed responses per category
<code>pars</code>	A named vector of parameters of the memory measurement model
<code>m3_model</code>	A <code>bmmmodel</code> object specifying the m3 model that densities or random samples should be generated for
<code>act_funs</code>	A <code>bmmformula</code> object specifying the activation functions for the different response categories for the "custom" version of the M3. 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 <code>act_funs</code> argument manually
<code>log</code>	Logical; if TRUE (default), values are returned on the log scale.
<code>...</code>	can be used to pass additional variables that are used in the activation functions, but not parameters of the model
<code>n</code>	Integer. Number of observations to generate data for
<code>size</code>	The total number of observations in all categories

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

```
model <- m3(
  resp_cats = c("corr", "other", "npl"),
  num_options = c(1, 4, 5),
  choice_rule = "simple",
  version = "ss"
)
dm3(x = c(20, 10, 10), pars = c(a = 1, b = 1, c = 2), m3_model = model)
```

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 variable 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:**
 - $\mu_1 = 0$
 - $\mu_2 = 0$
 - $\kappa_2 = -100$
- **Default parameter links:**
 - $\mu_1 = \tan_half$; $\kappa = \log$; $\theta = \text{identity}$
- **Default priors:**
 - μ_1 :
 - * main: $\text{student_t}(1, 0, 1)$
 - κ :
 - * main: $\text{normal}(2, 1)$
 - * effects: $\text{normal}(0, 1)$
 - θ :
 - * main: $\text{logistic}(0, 1)$

Value

An object of class `bmmmodel`

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"
)
```

Description

Density, distribution, and random generation functions for the two-parameter mixture model with the location of μ , precision of memory representations κ 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

<code>x</code>	Vector of observed responses
<code>mu</code>	Vector of locations
<code>kappa</code>	Vector of precision values
<code>p_mem</code>	Vector of probabilities for memory recall
<code>log</code>	Logical; if TRUE, values are returned on the log scale.
<code>q</code>	Vector of quantiles
<code>p</code>	Vector of probability
<code>n</code>	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	<i>Three-parameter mixture model by Bays et al (2009).</i>
-----------	--

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 <code>deg2rad</code> function.
nt_features	A character vector with the names of the non-target feature values. The <code>non_target</code> feature values should be in radians and centered relative to the target. Alternatively, if <code>regex=TRUE</code> , 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 <code>set_size</code> varies) or a numeric value for the <code>set_size</code> , if the <code>set_size</code> is fixed.
regex	Logical. If <code>TRUE</code> , the <code>nt_features</code> 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 variable 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 = identity; thetant = identity
- **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 `bmmmodel`

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	<i>Distribution functions for the three-parameter mixture model (mixture3p)</i>
----------------	---

Description

Density, distribution, and random generation functions for the three-parameter mixture model with the location of μ , precision of memory representations κ , 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 separating 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 remembered 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 remembered 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 candidates in each of the response categories

oberauer_lin_2017	<i>Data from Experiment 1 reported by Oberauer & Lin (2017)</i>
-------------------	---

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>

rejection_sampling	<i>Rejection Sampling</i>
--------------------	---------------------------

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	<i>Restructure Old bmmfit Objects</i>
--------------------	---------------------------------------

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	<i>Signal Discrimination Model (SDM) by Oberauer (2023)</i>
-----	---

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 `bmmmodel`

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 μ , memory strength c , and precision κ . 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

<code>x</code>	Vector of quantiles
<code>mu</code>	Vector of location values in radians
<code>c</code>	Vector of memory strength values
<code>kappa</code>	Vector of precision values
<code>log</code>	Logical; if TRUE, values are returned on the log scale.
<code>parametrization</code>	Character; either "bessel" or "sqrtexp" (default). See the online article for details on the parameterization.
<code>q</code>	Vector of quantiles
<code>lower.tail</code>	Logical; If TRUE (default), return $P(X \leq x)$. Else, return $P(X > x)$

<code>log.p</code>	Logical; if TRUE, probabilities are returned on the log scale.
<code>lower.bound</code>	Numeric; Lower bound of integration for the cumulative distribution
<code>p</code>	Vector of probabilities
<code>n</code>	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)

```

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)

Details

The softmax function is a bijective function that maps a real vector with length $m-1$ 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:

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

Code adapted from the [utilities](#) package

Value

Value of the softmax function or its inverse

Examples

```
softmax(5:7)
softmaxinv(softmax(5:7))
```

stancode.bmmformula	<i>Generate Stan code for bmm models</i>
---------------------	--

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 bmmmodel object for required arguments.
model	A description of the model to be fitted. This is a call to a bmmmodel 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 bmmmodel object for required arguments.
model	A description of the model to be fitted. This is a call to a bmmmodel 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 <i>bmm</i> summary method (default), or to display the <i>brms</i> 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	<i>Measurement models available in bmm</i>
------------------	--

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	<i>Update a bmm model</i>
---------------	---------------------------

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

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

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)))

```

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 identifier

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

- * **bmmmodel**
 - imm, [15](#)
 - m3, [23](#)
 - mixture2p, [28](#)
 - mixture3p, [31](#)
 - sdm, [37](#)
- * **dataset**
 - oberauer_lewandowsky_2019_e1, [34](#)
 - oberauer_lin_2017, [35](#)
 - zhang_luck_2008, [48](#)
- * **deprecated**
 - bmm, [4](#)
 - imm, [15](#)
 - sdm, [37](#)
- * **developer**
 - apply_links, [3](#)
- * **distribution**
 - IMMdist, [20](#)
 - m3dist, [26](#)
 - mixture2p_dist, [29](#)
 - mixture3p_dist, [33](#)
 - rejection_sampling, [36](#)
 - SDMdist, [39](#)
- * **extract_info**
 - default_prior.bmmformula, [13](#)
 - fit_info, [14](#)
 - stancode.bmmformula, [42](#)
 - standata.bmmformula, [43](#)
- * **transform**
 - c_parametrizations, [12](#)
 - calc_error_relative_to_nontargets, [10](#)
 - circle_transform, [11](#)
 - construct_m3_act_funs, [12](#)
 - k2sd, [22](#)
 - restructure.bmmfit, [36](#)
 - softmax, [41](#)
 - wrap, [47](#)
- apply_links, [3](#)
- bmf (bmmformula), [7](#)
- bmm, [4](#)
- bmm(), [5](#), [13](#)
- bmm-package, [3](#)
- bmm_options, [9](#)
- bmmformula, [7](#)
- bmmformula(), [6](#), [46](#)
- brms::brm(), [6](#)
- brms::brmsformula(), [8](#)
- brms::default_prior(), [14](#)
- brms::set_prior(), [5](#), [42](#)
- brms::stancode(), [42](#)
- brms::standata(), [43](#)
- brms::update.brmsfit(), [46](#)
- c_bessel2sqrtepx (c_parametrizations), [12](#)
- c_parametrizations, [12](#)
- c_sqrtepx2bessel (c_parametrizations), [12](#)
- calc_error_relative_to_nontargets, [10](#)
- circle_transform, [11](#)
- construct_m3_act_funs, [12](#)
- default_prior
 - (default_prior.bmmformula), [13](#)
- default_prior(), [5](#), [6](#), [42](#)
- default_prior.bmmformula, [13](#)
- deg2rad (circle_transform), [11](#)
- dimm (IMMdist), [20](#)
- dm3 (m3dist), [26](#)
- dmixture2p (mixture2p_dist), [29](#)
- dmixture3p (mixture3p_dist), [33](#)
- dsdm (SDMdist), [39](#)
- fit_info, [14](#)
- fit_model (bmm), [4](#)
- imm, [15](#)
- IMMabc (imm), [15](#)

IMMbsc (imm), 15
IMMdist, 20
IMMfull (imm), 15

k2sd, 22

m3, 23
m3dist, 26
mixture2p, 28
mixture2p_dist, 29
mixture3p, 31
mixture3p_dist, 33

oberauer_lewandowsky_2019_e1, 34
oberauer_lin_2017, 35
options, 6

pimm (IMMdist), 20
pmixture2p (mixture2p_dist), 29
pmixture3p (mixture3p_dist), 33
psdm (SDMdist), 39

qimm (IMMdist), 20
qmixture2p (mixture2p_dist), 29
qmixture3p (mixture3p_dist), 33
qsdm (SDMdist), 39

rad2deg (circle_transform), 11
rejection_sampling, 36
restructure (restructure.bmmfit), 36
restructure.bmmfit, 36
rimm (IMMdist), 20
rm3 (m3dist), 26
rmixture2p (mixture2p_dist), 29
rmixture3p (mixture3p_dist), 33
rsdm (SDMdist), 39

saveRDS, 5
sdm, 37
SDMdist, 39
sdmSimple (sdm), 37
softmax, 41
softmaxinv (softmax), 41
stancode (stancode.bmmformula), 42
stancode(), 6
stancode.bmmformula, 42
standata (standata.bmmformula), 43
standata(), 6
standata.bmmformula, 43
summary.bmmfit, 44

summary.brmsfit, 45
supported_models, 45
supported_models(), 5, 6, 14, 42, 43

update.bmmfit, 46

wrap, 47

zhang_luck_2008, 48