Package: bmm (via r-universe)

October 16, 2026

Title Easy and Accessible Bayesian Measurement Models Using 'brms'

Version 1.4.1

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/popov-lab/bmm, https://venpopov.com/bmm/

BugReports https://github.com/popov-lab/bmm/issues

Depends R (>= 4.1.0)

Imports bayesplot, brms (>= 2.21.0), crayon, fs, glue, graphics, matrixStats, methods, mvtnorm, parallel, posterior (>= 1.6.0), rtdists, rlang, stats, withr

Suggests bookdown, cmdstanr (>= 0.7.0), cowplot, dplyr, emmeans (>= 1.8.0), fansi, ggplot2, ggthemes, knitr, loo (>= 2.10.0), mixtur, MPTinR, pkgbuild, remotes, rmarkdown, roxygen2, rstan, stringr, testthat (>= 3.3.2), tidybayes, tidyr, usethis, vctrs, waldo, yaml

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

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first conditional_effects, update, model_sdt_rating, model_sdt_cdp, default-priors, model_m3, model_cswald, helpers-init, model_ezdm, pp_check

Config/Needs/website rmarkdown

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake make libgsl0-dev libicu-dev libuv1-dev

Repository https://popov-lab.r-universe.dev

Date/Publication 2026-10-16 06:19:10 UTC

RemoteUrl https://github.com/venpopov/bmm

RemoteRef v1.4.1

RemoteSha 2e6761be6357f8ced3904efc684dc05b399e9930

Contents

bmm-package . . . 4
adjust_ezdm_accuracy . . . 5
aggregate_sdt_cdp_data . . . 6
apply_links . . . 7
auc_sdt . . . 8
bmm . . . 9
bmm_data_check . . . 13
bmm_models . . . 15
bmm_options . . . 16
bmm_setup . . . 18
bmmformula . . . 19
broeder_schuetz_2009_e3 . . . 21
c_parametrizations . . . 22
calc_error_relative_to_nontargets . . . 23
circle_transform . . . 23
conditional_effects.bmmfit . . . 24
construct_m3_act_funs . . . 27
cswald . . . 28
cswald_dist . . . 30
data_color_judgement_task . . . 33
ddm . . . 33
ddm_dist . . . 36
default_prior.bmmformula . . . 37
emmeans-bmmfit . . . 38
extract_parameter_dimensions . . . 39
extract_stan_blocks . . . 40
ezdm . . . 41
ezdm_dist . . . 44
ezdm_summary_stats . . . 47
fit_info . . . 50
flag_contaminant_rts . . . 51
imm . . . 53

IMMdist . . . 58
k2sd . . . 61
latent_sdt . . . 61
m3 . . . 64
m3dist . . . 68
meyer_grant_jakob_2025 . . . 70
mixture2p . . . 71
mixture2p_dist . . . 73
mixture3p . . . 74
mixture3p_dist . . . 77
model_citation . . . 78
mpt . . . 79
mpt_data_reshape . . . 84
mpt_from_eqn . . . 87
mpt_from_string . . . 90
mpt_tree . . . 94
mptdist . . . 96
mratio . . . 98
native_parameters . . . 99
native_transform . . . 102
oberauer_lewandowsky_2019_e1 . . . 104
oberauer_lin_2017 . . . 105
parameter_info . . . 105
plot.bmm_sdt_auc . . . 106
plot.bmm_sdt_latent . . . 107
plot.bmm_sdt_roc . . . 108
plot.mpt_tree . . . 109
posterior_epred.bmmfit . . . 111
pp_check.bmmfit . . . 112
pp_check_vars . . . 114
ppp . . . 115
prior_info . . . 118
rejection_sampling . . . 119
restructure.bmmfit . . . 120
roc_observed . . . 121
roc_sdt . . . 122
sdm . . . 124
SDMdist . . . 126
sdt_cdp . . . 128
sdt_cdp_dist . . . 133
sdt_dpsdt_dist . . . 135
sdt_mafc . . . 137
sdt_mafc_dist . . . 140
sdt_metad_dist . . . 141
sdt_ranking . . . 143
sdt_ranking_dist . . . 146
sdt_rating . . . 148
sdt_rating_dist . . . 156

sdt_sensitivity 158
sdt_thresholds 160
sdt_yn 162
sdt_yn_dist 169
SDTdist 171
softmax 173
stancode.bmmformula 174
standata.bmmformula 175
summary.bmmfit 176
update.bmmfit 177
validate_fast_guesses 179
wrap 181
zhang_luck_2008 182

Index **184**

`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: Gidon T. Frischkorn <gfrischkorn@icloud.com> (ORCID) [copyright holder]

Authors:

- Gidon T. Frischkorn <gfrischkorn@icloud.com> (ORCID) [copyright holder]
- Vencislav Popov <vencislav.popov@gmail.com> (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/popov-lab/bmm
- https://venpopov.com/bmm/
- Report bugs at https://github.com/popov-lab/bmm/issues

`adjust_ezdm_accuracy` *Adjust Accuracy Counts for Contamination (deprecated)*

Description

Deprecated. `ezdm_summary_stats()` now returns contaminant-free `n_upper` and `n_trials`, so this second step is no longer needed; applying it corrects the same cell twice. Set the guess rate with the `guess_rate` argument of `ezdm_summary_stats()` instead.

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, a single value.
`n_trials`	Numeric. Total number of trials, a single value.
`contaminant_prop`	Numeric. Estimated proportion of contaminant trials (e.g., from the `contaminant_prop` column of `ezdm_summary_stats()`), a single value.
`guess_rate`	Numeric. Assumed accuracy rate for contaminant trials (random guessing). Default is 0.5 (appropriate for 2AFC tasks).

Details

The function adjusts one cell at a time. Vectors of length greater than one are an error; to correct several cells, use `ezdm_summary_stats()` on each group.

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

```
# Deprecated. ezdm_summary_stats() returns contaminant-free counts already:
set.seed(42)
rt <- c(rnorm(80, 0.55, 0.05), runif(20, 0.1, 4))
response <- c(rbinom(80, 1, 0.85), rbinom(20, 1, 0.5))
ezdm_summary_stats(rt, response, contaminant_bound = c(0.1, 4))
```

`aggregate_sdt_cdp_data` *Aggregate long-format Remember/Know data for* `sdt_cdp()`

Description

Reshapes long-format Remember/Know recognition data – one row per trial, or one row per response category with a count column – into the aggregated wide format `sdt_cdp()` is fit to: one row per cell and one count column per response category, named `new1 ... new<n_new>`, `know<k>` / `remember<k>` (and `guess<k>` when the data contain "guess" judgments), with `k` on the unified confidence scale `n_new + 1 ... n_new + n_old`.

Usage

```
aggregate_sdt_cdp_data(data, judgment, confidence, count = NULL, response = "")
```

Arguments

`data`	A data frame in long format. Cells are defined by all columns other than `judgment`, `confidence`, and `count` (e.g. participant, stimulus class, and any condition variables), which are carried over to the output unchanged.
`judgment`	The name of the column coding the memory judgment, with values `"new"`, `"remember"`, `"know"`, and optionally `"guess"`.
`confidence`	The name of the column coding confidence on the unified old/new scale (integer, 1 = most confident "new" to K = most confident "old"; "new" judgments occupy the low levels, "old" judgments the high levels).
`count`	Optional name of a column of response counts. If `NULL` (default), each row is treated as a single trial. Non-integer counts are rounded with a warning; `NA` counts and `NA` confidences are refused.
`response`	Optional common prefix for the generated count columns (default `""`). Pass the same value to the `response` argument of `sdt_cdp()`.

Value

A data frame with one row per cell: the cell-defining columns followed by the response count columns, ready to pass to `bmm()` with an `sdt_cdp()` model. The numbers of "new" and "old" confidence levels are inferred from the data.

Examples

```
dat <- data.frame(
  stimulus = c(1L, 1L, 1L, 0L, 0L, 0L),
  judgment = c("remember", "know", "new", "new", "know", "new"),
  confidence = c(3L, 2L, 1L, 1L, 3L, 1L),
  count = c(40L, 25L, 15L, 55L, 10L, 20L)
)
aggregate_sdt_cdp_data(dat, "judgment", "confidence", "count")
```

`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)
```

auc_sdt *Area under the ROC curve from a fitted SDT model*

Description

Computes the posterior area under the ROC curve (AUC). For Gaussian and Gumbel (min or max) equal-variance binary SDT the AUC is available in closed form from the d draws; otherwise it is obtained by trapezoidal integration of the model-implied curve (for rating fits, the curve swept from the posterior of d and sdratio, lifted by Ro/Rn for version = "dpsdt", so the AUC is the area under the predicted mixture ROC and not $\Phi(d/\sqrt{2})$; for version = "metad" it is the type-1 curve). The returned AUC is always the area under the full curve, not the trapezoid of the discrete operating points or the K-1 rating thresholds; for binary multi-criteria fits it is one value per curve.

Usage

```
auc_sdt(
  fit,
  conditions = NULL,
  probs = c(0.025, 0.975),
  criterion_points = NULL,
  ...
)
```

Arguments

fit	A bmmfit object returned by bmm() from an SDT model.
conditions	Optional data frame of predictor values at which to evaluate the model. Column names must match predictor variables used in the formula. If NULL (default), unique predictor combinations are derived from the data.
probs	Numeric vector of length 2. Quantiles for the credible interval (default c(0.025, 0.975)).
criterion_points	Optional. Control of the binary multi-criteria behaviour. NULL (default) auto-detects predictors that vary the criterion only (not d/sdratio) and treats their levels as operating points on one curve. Pass a character vector of column names to force that classification, or FALSE to disable it (one separate curve per predictor combination). Ignored for rating models.
...	Additional arguments passed to brms::posterior_linpred(), such as draw_ids to use a subset of the posterior draws. ndraws is refused: each parameter is drawn by its own call, so random subsets would not match across parameters.

Details

The closed form is used only when sdratio is fixed at 0, where d (which is d_a) equals d'; every fit with sdratio estimated or fixed away from 0 takes the numerical route, so the AUC is invariant to the sensitivity parameterization.

Analytical formulas (equal variance, where $d_a = d'$): normal EV-SDT $AUC = \Phi(d'/\sqrt{2})$; Gumbel-min and Gumbel-max EV-SDT $AUC = \text{logistic}(g')$.

Value

A data frame of class `"bmm_sdt_auc"` with columns `AUC`, `.draw`, and any condition columns, plus a `summary` attribute (`AUC_mean`, `AUC_lower`, `AUC_upper`), `model_class`, `dist` and `conditions` attributes.

See Also

`roc_sdt()`, `plot.bmm_sdt_auc()`

Examples

```
## Not run:
# Three base-rate conditions shift the criterion, which identifies sdratio
dat <- expand.grid(id = 1:20, stimulus = c(0L, 1L),
                   condition = c("liberal", "neutral", "strict"))
dat$n_trials <- 100L
# simulated as bias, measured from the midpoint between the means (0 = boundary at the midpoint)
criteria <- c(liberal = -0.5, neutral = 0, strict = 0.5)
dat$n_old <- rsdt_yn(nrow(dat), dat$n_trials, dat$stimulus, d = 1.5,
                     bias = criteria[as.character(dat$condition)],
                     sdratio = 1.3)

fit <- bmm(
  formula = bmf(d ~ 1, criterion ~ 0 + condition, sdratio ~ 1),
  data = dat,
  model = sdt_yn(response = "n_old", stimulus = "stimulus",
                 n_trials = "n_trials"),
  cores = 4,
  backend = "cmdstanr"
)

auc <- auc_sdt(fit)
auc
plot(auc)

## End(Not run)
```

---

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 bmm_models() to see the list of supported models, and ?modelname (for example ?imm) for 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. If "on_change", the saved model is re-used only if the Stan code, the Stan data, the factor levels of the model variables and the algorithm are unchanged; otherwise the model is fitted again. Because that comparison needs the Stan code and data of the current call, "on_change" runs the full bmm configuration pipeline, standata() and stancode() even when the cached fit is returned: about 0.4 s rather than 0.02 s for an **sdm** model of `oberauer_lin_2017`, much cheaper than compiling and sampling, but not free. Only the four things listed above are compared, so sampler settings do **not** force a refit – in particular `control = list(adapt_delta = )`, `iter`, `warmup`, `chains`, `seed`, `init` and `save_pars`. Raising `adapt_delta` after divergent transitions, or rerunning with `save_pars(all = TRUE)` for `loo()`, therefore returns the cached fit unchanged; delete the file or pass `file_refit = "always"` for those. Because the row order of the data reaches the Stan data, `"on_change"` is most predictable with `sort_data` fixed to `TRUE` or `FALSE` (globally via `options(bmm.sort_data = )`): under the default `"check"`, answering the interactive prompt differently than last time forces a refit.

`...` Further arguments passed to `brms::brm()` or Stan. See the description of `brms::brm()` for more details. Unless `control` names a `step_size` (`stepsize` for the rstan backend), bmm adds the starting step size set in `bmm_options()` to it; the other entries of `control` are kept.

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:

Continuous reproduction

- imm(): Interference measurement model by Oberauer and Lin (2017)
- mixture2p(): Two-parameter mixture model by Zhang and Luck (2008)
- mixture3p(): Three-parameter mixture model by Bays et al (2009)
- sdm(): Signal Discrimination Model (SDM) by Oberauer (2023)

Categorical recall and n-AFC decisions

- m3(): The Multinomial / Memory Measurement Model

Processing-tree models

- mpt(): Multinomial Processing Tree (MPT) models

Detection, recognition and confidence judgments

- sdt_cdp(): Continuous Dual-Process Signal Detection Theory (CDP)
- sdt_mafc(): Signal Detection Theory (m-AFC)
- sdt_ranking(): Signal Detection Theory (Ranking)
- sdt_rating(): Signal Detection Theory (Confidence Rating)
- sdt_yn(): Signal Detection Theory (Yes/No)

Choices and response times

- cswald(): Censored-Shifted Wald Model
- ddm(): Diffusion Decision Model
- ezdm(): EZ-Diffusion Model

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

bmm_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"
)
```

---

bmm_data_check *Human-readable pre-fit data report for bmm models*

Description

Inspect whether your data is coded the way a bmmodel expects *before* committing to Stan compilation and sampling. The function runs the same validation pipeline that bmm() runs internally (check_data() and check_formula()), but captures all errors, warnings and messages instead of throwing them, and combines them with a readable summary of:

- the response variables, their observed values and the coding the model expects
- the data columns each parameter formula uses, and how they are coded (factor levels, character vectors, numeric variables with few unique values)
- the number of observations per design cell (crossing the grouping variables from random effects terms with the categorical predictors), flagging sparse or empty cells
- model-specific diagnostics for common data mistakes (e.g. responses in degrees rather than radians for circular models, or misplaced NA values in the non-target features of set size varying designs). These are provided by data_check_findings() methods, which new models can extend

Problems are reported as findings rather than errors, so a single call shows everything that needs fixing at once.

The report covers the data and the formula only. It stops after `check_formula()` and takes no `prior` argument, so a malformed custom prior or a problem with the initial values is out of its reach and will only surface when you call `bmm()`.

Usage

```
bmm_data_check(formula, data, model, min_trials = 10)
```

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 `bmm_models()` to see the list of supported models, and `?modelname` (for example `?imm`) for their required arguments.
`min_trials`	Numeric. Design cells with fewer observations are flagged in the report. Defaults to 10. Ignored for models fit to aggregated data (e.g. `m3`, `ezdm`), where one row summarizes many trials.

Value

An object of class `bmm_data_check` with a print method. The object is a list containing the summarized `response`, `predictors` and `cells` information (including the full cell count table in `$cells$counts`), all `findings`, and the captured results of the validation `pipeline`.

See Also

`bmm()`, `check_data()`, `data_check_findings()`

Examples

```
# three-parameter mixture model: radians check, NA padding of the
# non-target features across set sizes, trials per participant x set size
bmm_data_check(
  bmf(kappa ~ 0 + set_size + (0 + set_size | ID), thetat ~ 0 + set_size),
  data = oberauer_lin_2017,
  model = mixture3p(
    resp_error = "dev_rad",
    nt_features = paste0("col_nt", 1:7),
    set_size = "set_size"
  )
)
```

```
# interference measurement model: additionally checks the NA padding
# of the non-target distances
bmm_data_check(
  bmf(c ~ 1, a ~ 1, kappa ~ 1, s ~ 1),
  data = oberauer_lin_2017,
  model = imm(
    resp_error = "dev_rad",
    nt_features = paste0("col_nt", 1:7),
    nt_distances = paste0("dist_nt", 1:7),
    set_size = "set_size",
    version = "full"
  )
)

# signal discrimination model
bmm_data_check(
  bmf(c ~ 0 + setsize, kappa ~ 0 + setsize),
  data = zhang_luck_2008,
  model = sdm(resp_error = "response_error")
)

# memory measurement model: response category counts as response variables
bmm_data_check(
  bmf(c ~ 1 + (1 | ID), a ~ 1 + (1 | ID)),
  data = oberauer_lewandowsky_2019_e1,
  model = m3(
    resp_cats = c("corr", "other", "npl"),
    num_options = c("n_corr", "n_other", "n_npl"),
    choice_rule = "simple",
    version = "ss"
  )
)

# diffusion decision model: reaction time plausibility and response coding
bmm_data_check(
  bmf(drift ~ condition + (condition | ID)),
  data = data_color_judgement_task,
  model = ddm(rt = "rt", response = "response_correct")
)
```

---

bmm_models *Measurement models available in* bmm

Description

Measurement models available in bmm

Usage

```
bmm_models()
```

Details

Printed, the result lists the models grouped by the task they are meant for, one line per model with its constructor and full name. The groups are: continuous reproduction; categorical recall and n-AFC decisions; detection, recognition and confidence judgments; choices and response times; processing-tree models. Type `?modelname` (for example `?imm`) for the arguments of a model.

Value

A character vector of model names with class `bmm_models`, which prints as the grouped list. Use it as a character vector in base R or dplyr, e.g. `"imm" %in% bmm_models()`; `as.character()` gives a plain one where a function refuses the class.

Examples

```
bmm_models()
"imm" %in% bmm_models()
```

---

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,
  step_size,
  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 or character. Controls when bmm() re-uses a fit saved via its file argument. TRUE or "always" always refits, FALSE or "never" always re-uses the saved fit, and "on_change" re-uses it only while the Stan code and data are unchanged. See bmm() for details. **Default: FALSE**

step_size — numeric or FALSE. The step size at which bmm() and update() start Stan's step-size search, passed as control = list(step_size =) (stepsize for the rstan backend). Stan's own starting value of 1 is far above the step sizes the hierarchical models in bmm adapt to, and the oversized trial steps of the search print lkj_corr_cholesky_lpdf and von_mises_lpdf exceptions at the start of warmup. The adapted step size and the posterior do not depend on the starting value. FALSE leaves the starting step size to Stan; a step_size (or stepsize) in the control list of bmm() always wins. **Default: 0.01**

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)
```

`bmm_setup` *Check whether this machine can fit bmm models*

Description

Fitting a model with `bmm()` needs a C++ toolchain (a compiler and `make`) and a Stan backend, the software that turns the model into a program and samples from it. There are two backends: the R package `cmdstanr`, which runs CmdStan, a separate program installed with `cmdstanr::install_cmdstan()`; and the R package `rstan`. `rstan` must load with either backend, because `brms` uses it to store every fit. `bmm_setup()` checks each of these, prints whether it passed, and gives one fix for every check that failed. It does not install or change anything.

Two checks concern the toolchain. "C++ toolchain" lets R compile a small test file; "CmdStan toolchain" is `cmdstanr`'s own check, which only looks for `make` and a compiler on the `PATH`. The second can pass while the first fails, e.g. on a Mac whose command line tools are missing, and the first is the one that decides.

With `smoke_test = TRUE`, it finally compiles and samples a small two-parameter mixture model (`mixture2p()`) through `bmm()`, on the backend `bmm()` will use. This is the only check that shows the whole chain works. On an Apple Silicon Mac it takes about 10 seconds with `cmdstanr` and 45 seconds with `rstan`, and it is skipped when a check it needs has failed. Once every check passes, rerun the `bmm()` call that failed.

Usage

```
bmm_setup(smoke_test = TRUE, backend = getOption("brms.backend", NULL))
```

Arguments

`smoke_test`	Logical. Compile and sample a small test model? Defaults to `TRUE`.
`backend`	The backend to check, `"cmdstanr"` or `"rstan"`. The default checks the backend `bmm()` would choose: the `brms.backend` option if it is set, otherwise `cmdstanr` whenever the `cmdstanr` package is installed, even when CmdStan is not, and `rstan` otherwise.

Value

A data frame of class bmm_setup with one row per check and the columns check, status ("pass", "fail" or "skip"), detail and fix (NA unless the check failed). It prints as a report.

See Also

bmm(), bmm_data_check()

Examples

```
# checks without compiling anything
bmm_setup(smoke_test = FALSE)

# compiles and samples a small model as well
bmm_setup()
```

---

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)
```

`broeder_schuetz_2009_e3` *Recognition ROC data from Broeder & Schuetz (2009, Experiment 3)*

Description

Binary old/new recognition data from 40 subjects, aggregated to response counts. Each subject was tested under five base-rate conditions: the proportion of old items shifts the decision criterion from conservative (`br1`) to liberal (`br5`) while leaving sensitivity unchanged, tracing a five-point binary ROC per subject. That is the cleanest design for estimating the unequal-variance ratio (`sdratio`) of `sdt_yn()`, which needs more than one operating point: a single condition with no other varying predictor yields only one hit/false-alarm pair and cannot separate a wider signal distribution from a larger d'. Counts were digitised from the frequencies reported in the original article.

Usage

```
broeder_schuetz_2009_e3
```

Format

`broeder_schuetz_2009_e3`**:**

A data frame with 400 rows (40 subjects x 5 conditions x 2 stimulus types) and 5 columns:

id Integer uniquely identifying each subject

condition Factor with five base-rate conditions, ordered from the most conservative (`br1`) to the most liberal (`br5`) induced criterion

stimulus Integer stimulus type: 0 = new/lure, 1 = old/target

n_old Integer count of "old" responses in that cell: hits for old items (`stimulus == 1`) and false alarms for new items (`stimulus == 0`)

n_trials Integer number of items presented in that cell

Source

Broeder, A., & Schuetz, J. (2009). Recognition ROCs are curvilinear—or are they? On premature arguments against the two-high-threshold model of recognition. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 35(3), 587–606. doi:10.1037/a0015279

Examples

```
## Not run:
# Unequal-variance yes/no SDT: the criterion varies across base-rate
# conditions, while sensitivity (d) and the signal/noise SD ratio (sdratio)
# are held constant across conditions.
model <- sdt_yn(
  response = "n_old", stimulus = "stimulus", n_trials = "n_trials"
)
fit <- bmm(
  formula = bmf(
```

```
    d ~ 1 + (1 | id),
    criterion ~ 0 + condition + (1 | id),
    sdratio ~ 1
  ),
  data = broeder_schuetz_2009_e3,
  model = model,
  backend = "cmdstanr"
)

## End(Not run)
```

`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)
```

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"; "posterior_predict" is not available for models with a multinomial family, see Details)

Details

Parameter Types:

bmm models use two types of parameters internally:

- **Non-linear parameters** (nlpar): Model parameters that enter the likelihood through a non-linear formula, like kappa and thetat of mixture3p() or c and a of m3()
- **Distributional parameters** (dpar): Parameters of the response distribution itself, like kappa and c of sdm()

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.

estimate__, lower__, upper__ and the spaghetti lines follow scale. se__ does not: it is the spread of the draws on the scale brms or bmm summarises them on, so for a link other than identity it can sit on a different scale than estimate__. For example, se__ of kappa is on the sampling scale in mixture3p() and on the native scale in sdm(), at either scale. Use lower__ and upper__ to describe uncertainty on the requested scale. With robust = FALSE, estimate__ is the mean and se__ the standard deviation of the draws on that scale, and estimate__ is then mapped to the requested scale: at scale = "native" the estimate of a non-linear parameter with a log link is the exponential of the mean log value, not the mean of the exponentiated draws.

The default of robust is TRUE (median and MAD), as in brms::conditional_effects(), with one exception: thetat and thetant of mixture3p() at scale = "native" default to robust = FALSE (mean and standard deviation).

Models with a multinomial family:

In models fitted with a multinomial response (m3(), sdt_rating(), sdt_cdp(), sdt_ranking()), the parameters are latent quantities: they have no predictive distribution, no response categories and no response points. For these models:

- The effect is the posterior of the parameter itself, summarised by the median and MAD of the draws; use `robust = FALSE` for their mean and standard deviation.
- The grid of the plotted predictor and the values of all other predictors are set by the rules of `brms::conditional_effects()`.
- Without `effects`, only the effects `brms` lists for the parameter's own formula are returned. A parameter without predictors has no effects.
- `categorical`, `ordinal`, `select_points` and `transform` are errors, and `method` must be `"posterior_epred"` or `"posterior_linpred"` (or an alias `brms::conditional_effects()` reads as one of them, such as `"fitted"`), which agree for a parameter.

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 `bound`, `ndt` and `s`, and "logit" for `zr`. `drift` has a "log" link in "simple", where it is positive, and an "identity" link in "crisk", where it can be negative, as in `ddm()` and `ezdm()`. For positive parameters, "softplus" is available as an alternative to "log" that grows linearly for large values and avoids the numerical blow-up of `exp()`. A name that is not a parameter of the model is an error, and a link that allows values the default link excludes (e.g. "identity" for a positive parameter) is a warning.
`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. `zr` is fixed at 0 on its logit scale (an unbiased `zr = 0.5`) unless you give it a formula. For more details, see Miller et al. (2018).
`...`	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. (2018). 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, half the boundary separation bound of version "crisk", ddm() and ezdm())
  - `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)
    - `sd`: exponential(2)
  - `bound`:
    - `main`: normal(0,0.3)
    - `effects`: normal(0,0.3)
    - `sd`: exponential(2)
  - `ndt`:
    - `main`: normal(-2,0.3)
    - `effects`: normal(0,0.3)
    - `sd`: exponential(2)
  - `s`:
    - `main`: normal(0,0.3)
    - `effects`: normal(0,0.2)
    - `sd`: exponential(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)
```

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 named list of links for the parameters, e.g. `links = list(bound = "softplus")`. For positive parameters (e.g. `bound`, `ndt`), "softplus" is available as an alternative to the default "log" link that grows linearly for large values and avoids the numerical blow-up of `exp()`. A name that is not a parameter of the model is an error, and a link that allows values the default link excludes (e.g. "identity" for a positive parameter) is a warning.
`...`	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/10.1037/0033-295X.85.2.59
- **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 as a proportion of the boundary separation, measured from the lower boundary (0.5 = unbiased, > 0.5 = biased towards the upper response).
- **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)
    * sd: exponential(1)
  - bound:
    * main: normal(0,0.5)
    * effects: normal(0,0.5)
    * sd: exponential(2)
  - ndt:
    * main: normal(-1.5,0.5)
    * effects: normal(0,0.3)
    * sd: exponential(2)
  - zr:
    * main: normal(0,0.5)
    * effects: normal(0,0.3)
    * sd: exponential(2)

Value

An object of class bmmodel

Default behavior

By default, zr is fixed at 0 on its logit link scale, that is at an unbiased starting point of zr = 0.5. If you want to estimate zr, add a formula for zr in your bmf() call. Estimates of zr are on the logit scale too, so use plogis() to read them as proportions.

With accuracy-coded responses (1 = correct, 0 = error) the unbiased default is the usual assumption. With response-coded data (1 and 0 for the two response options, e.g. "word" and "nonword"), a bias towards one option moves the starting point; with zr fixed, that bias is absorbed by the other parameters instead. Estimate zr for response-coded data, e.g. zr ~ 1, and code drift so that its sign follows the stimulus.

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 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.

Each model parameter carries three default priors, listed in the documentation of the model: main for the intercept (or for all levels of a factor when the intercept is suppressed), effects for the remaining regression coefficients, and sd for the standard deviations of its random effects. The sd prior is applied as a blanket prior to every random-effects standard deviation of that parameter (all grouping factors, intercepts and slopes alike) and replaces the student_t(3, 0, 2.5) default of brms. To override it, address the parameter with dpar or nlpar in brms::set_prior(), e.g. set_prior("exponential(2)", class = "sd", nlpar = "kappa").

Correlations among random effects belong to a grouping factor rather than to one parameter, so they get a single default for the whole model: lkj(2) instead of the lkj(1) of brms, set whenever the model estimates a correlation matrix. To override it, use e.g. set_prior("lkj(1)", class = "cor").

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 bmm_models() to see the list of supported models, and ?modelname (for example ?imm) for 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

`bmm_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. A declaration whose type this function does not model is returned with its name and NA for its type and dimensions, so that only that parameter is left without initial values

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 = c("3par", "4par"),
  ...
)
```

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 named list of links for the parameters, e.g. `links = list(bound = "softplus")`. For positive parameters (e.g. `bound`, `ndt`), "softplus" is available as an alternative to the default "log" link that grows linearly for large values and avoids the numerical blow-up of `exp()`. A name that is not a parameter of the model is an error, and a link that allows values the default link excludes (e.g. "identity" for a positive parameter) is a warning.
`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/10.3758/BF03194023
  - Chávez De la Peña, A. F., & Vandekerckhove, J. (2025). An EZ Bayesian hierarchical drift diffusion model for response time and accuracy. Psychonomic Bulletin & Review, 32(6), 3067-3087. 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 as a proportion of the boundary separation, measured from the lower boundary (0.5 = unbiased, > 0.5 = biased towards the upper response).
  - `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)
    - `sd`: exponential(1)

- bound:
  - main: normal(0,0.5)
  - effects: normal(0,0.5)
  - sd: exponential(2)
- ndt:
  - main: normal(-1.5,0.5)
  - effects: normal(0,0.3)
  - sd: exponential(2)
- zr:
  - main: normal(0,0.5)
  - effects: normal(0,0.3)
  - sd: exponential(2)
- s:
  - main: normal(0,1)
  - effects: normal(0,0.3)
  - sd: exponential(2)

In version "4par", a boundary reached fewer than twice, or without RT summaries (NA), enters the model through the response counts only. bmm() adds two columns to the data, rt_used_upper and rt_used_lower, which it sets to 0 for such a boundary and to 1 otherwise, and replaces the summaries of such a boundary with a placeholder the likelihood never reads. A 0 already in these columns is kept, so that update() does not read the placeholders in fit$data as data.

newdata passed to log_lik(), predict() or posterior_predict() needs these columns too. Rows of fit$data have them. For raw data, set both to 1: a boundary reached fewer than twice or with NA summaries is then still left out of the likelihood. brms functions that compare the response with predictions, predictive_error() and residuals(method = "posterior_predict"), read the placeholders as data in cells whose rt_used_upper is 0, so leave those cells out first, for example with newdata = subset(fit$data, rt_used_upper == 1).

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
# drift has an identity link; bound and ndt have log links
fixed_effects <- brms::fixef(fit)
fixed_effects["drift_Intercept", "Estimate"]
exp(fixed_effects[c("bound_Intercept", "ndt_Intercept"), "Estimate"])

## 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

Details

The number of upper-boundary responses is binomial. The two RT summaries follow the joint sampling distribution of the mean and the variance of n independent decision times, matched to the exact first four cumulants of the first-passage time. Writing MDT and VRT for its mean and variance and κ_3, κ_4 for its third and fourth cumulants, and $W = \kappa_4/n + 2\,\mathrm{VRT}^2/(n-1)$ for the exact variance of the sample variance,

$$\mathrm{var_rt} \sim \mathrm{Gamma}(\mathrm{VRT}^2/W, \mathrm{VRT}/W),$$

$$\mathrm{mean_rt} \mid \mathrm{var_rt} \sim N\left(\mathrm{ndt} + \mathrm{MDT} + \frac{\kappa_3/n}{W}(\mathrm{var_rt} - \mathrm{VRT}),\ \sqrt{\mathrm{VRT}/n - (\kappa_3/n)^2/W}\right),$$

so that $\mathrm{Var}(\mathrm{mean_rt}) = \mathrm{VRT}/n$ and $\mathrm{Cov}(\mathrm{mean_rt}, \mathrm{var_rt}) = \kappa_3/n$ are exact. Decision times are right-skewed (with a symmetric start point their kurtosis is 8.8 at zero drift and falls towards 3 as drift grows), so the older form that assumes normal reaction times — independent normal and scaled chi-square $\mathrm{Gamma}((n-1)/2, (n-1)/(2\,\mathrm{VRT}))$ terms — understates the sampling variance of var_rt by a factor of $1 + (\mathrm{kurtosis} - 3)(n-1)/(2n)$: about 3.8 with 100 trials per cell and 3.5 to 3.6 with 10, less at accuracies above .95. With an asymmetric start point in version "4par" it is more at the boundary nearer the start point (up to 5.7 for zr between .3 and .7) and less at the other. It also ignores a correlation of about 0.7 between the two statistics, making posteriors too narrow. The terms above reduce to it when $\kappa_3 = \kappa_4 = 0$.

For version "3par" the start point is symmetric, so the decision time is independent of which boundary is hit and all n_trials responses inform one set of cumulants. For version "4par" the two boundaries have different decision-time distributions, so each is given its own summaries and its own response count. A boundary reached fewer than twice has no sample variance, and one whose summaries are NA has nothing to evaluate; dezdm() lets either contribute only through the binomial term. rezdm() and ezdm_summary_stats() code such summaries as NA, and bmm() keeps these cells: it replaces the summaries of such a boundary with a placeholder that the likelihood never reads, so the response counts still inform the fit. The per-boundary formulas condition on the realised counts, which are themselves random.

The two additional cumulants cost sampling time. In two simulated designs (30 subjects with 200 or 250 trials, 3 seeds each, one machine) a gradient of the whole model took 1.3 times as long as with the older form for version "3par" and 1.9 times for version "4par". The likelihood alone took 1.5 to 2.0 and 1.7 to 2.5 times as long, most where accuracy is near chance, so the ratio grows with the number of cells.

Simulated mean_rt is not truncated at ndt. When a summary rests on few responses (a handful of trials in version "3par", or a rarely reached boundary in version "4par", whatever n_trials is), rezdm() can return mean_rt below ndt or even mean_rt <= 0, which bmm() rejects; drop those rows before fitting.

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 = 500,
  tol = 1e-06,
  guess_rate = 0.5
)
```

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 its moments are not the mean and variance the EZ equations were derived for. For right-skewed RTs the median lies below the mean and (IQR/1.349)^2 below the variance, so the EZ estimates are offset systematically, not just noisier: a larger drift rate and a smaller boundary separation than the same data give with "mixture".
`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 the RT summaries, which are NA with fewer. Compared against the total number of trials for `version = "3par"` and against each boundary's own count for `version = "4par"`, so the two versions can disagree about whether the same cell is corrected. For `version = "4par"`, `bmm()` still fits a cell whose summaries at one or both boundaries are NA, through its response counts. 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 500. The EM converges slowly when the contaminant proportion is close to zero, and a cell that reaches `maxit` without converging falls back to robust moments with a warning.

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

`guess_rate` Numeric. Accuracy expected of a contaminant response, used to split the estimated contaminants of `version = "3par"` across the two boundaries. Default is 0.5, appropriate for a two-choice task; use 0.25 for a 4AFC task, or 0 if contaminants are never correct. Ignored for `version = "4par"`, which estimates contamination separately per boundary and needs no such assumption.

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.

The returned counts describe the same responses as the returned moments. For `version = "3par"` the estimated contaminants are shared between the boundaries at `guess_rate`; a cell whose observed accuracy lies outside `[guess_rate * p, 1 - p * (1 - guess_rate)]` for an estimated proportion `p` cannot have arisen that way, so the contaminants are removed proportionally from both boundaries instead, leaving the observed accuracy unchanged apart from rounding, and a warning is issued. For `version = "4par"` each boundary is corrected by its own estimate.

This function is designed to work with `dplyr::group_by()` and `dplyr::reframe()` for grouped operations.

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`.

`n_upper` and `n_trials` count the responses the reported moments rest on. Whenever a contaminant proportion was estimated, the expected number of contaminants is removed from both counts and the result is rounded; with `method = "simple"`, `method = "robust"`, or an EM that did not converge, they are the raw counts.

See Also

`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 = 500,
  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 500. A group whose EM reaches `maxit` without converging gets `NA` probabilities and a warning.

`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), `NA` for every trial of a group whose EM did not converge (a threshold such as `filter(contam_prob <= 0.5)` drops that whole group, so check for `NA` first), 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 = c("full", "bsc", "abc"),
  ...
)

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. https://doi.org/10.1037/rev0000044

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: normal(0, 0.5)
    - effects: normal(0, 0.25)
    - sd: exponential(4)
  - kappa:
    - main: normal(2, 1)
    - effects: normal(0, 1)
    - sd: exponential(1)
  - a:
    - main: normal(0, 1)
    - effects: normal(0, 1)
    - sd: exponential(1)
  - c:
    - main: normal(0, 1)
    - effects: normal(0, 1)
    - sd: exponential(1)
  - s:
    - main: normal(0, 1)
    - effects: normal(0, 1)
    - sd: exponential(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: normal(0, 0.5)
    * effects: normal(0, 0.25)
    * sd: exponential(4)
  - kappa:
    * main: normal(2, 1)
    * effects: normal(0, 1)
    * sd: exponential(1)
  - c:
    * main: normal(0, 1)
    * effects: normal(0, 1)
    * sd: exponential(1)
  - s:
    * main: normal(0, 1)
    * effects: normal(0, 1)
    * sd: exponential(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: normal(0, 0.5)
    - effects: normal(0, 0.25)
    - sd: exponential(4)
  - kappa:
    - main: normal(2, 1)
    - effects: normal(0, 1)
    - sd: exponential(1)
  - a:
    - main: normal(0, 1)
    - effects: normal(0, 1)
    - sd: exponential(1)
  - c:
    - main: normal(0, 1)
    - effects: normal(0, 1)
    - sd: exponential(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)
```

latent_sdt — *Model-implied latent decision-variable distributions*

Description

Computes the noise and signal latent decision-variable densities implied by a fitted `sdt_yn()` or `sdt_rating()` model, together with the response criterion (binary) or the K-1 confidence thresholds (rating). This is the canonical signal detection picture: two evidence distributions separated by the sensitivity and cut by one or more decision boundaries.

Usage

```
latent_sdt(
  fit,
  conditions = NULL,
  n_grid = 200,
  probs = c(0.025, 0.975),
  collapse = NULL,
  show_competitors = FALSE,
  ...
)
```

Arguments

fit	A bmmfit object returned by bmm() from an SDT model.
conditions	Optional data frame of predictor values at which to evaluate the model. Column names must match predictor variables used in the formula. If NULL (default), unique predictor combinations are derived from the data.
n_grid	Integer. Number of points on the evidence-axis grid at which each density is evaluated (default 200).
probs	Numeric vector of length 2. Lower and upper quantiles for the credible band on the criterion/threshold locations (default c(0.025, 0.975)).
collapse	Control of which predictor dimensions are collapsed into one panel. NULL (default) auto-detects: predictors of the criterion/thresholds only (and the set size) are collapsed; predictors of d/sdratio are faceted. Pass a character vector of column names to force those columns to collapse, or FALSE to facet every dimension (one panel per combination).
show_competitors	Logical (default FALSE). For sdt_mafc/sdt_ranking only, additionally overlay the density of the maximum of the m - 1 distractor samples (one curve per set size) – the "effective competitor" the target must beat – which shifts rightward as m grows and visualises why accuracy falls with set size. The overlay shows the set sizes in the fitted data, whatever conditions requests. Ignored for sdt_yn/sdt_rating.
...	Additional arguments passed to brms::posterior_linpred(), such as draw_ids to use a subset of the posterior draws. ndraws is refused: each parameter is drawn by its own call, so random subsets would not match across parameters.

Details

Densities are evaluated at the posterior-mean parameters of each condition, on the evidence axis the model's criterion is measured on: noise at 0 with unit SD and signal at sep with SD exp(sdratio), where sep = d * sqrt((1 + exp(sdratio)^2) / 2) is the separation in noise-SD units. A fit that estimated bias has its boundaries drawn on the same axis, at bias + sep / 2. d is d_a, measured in root-mean-square SD units, so sep equals d whenever sdratio is fixed at 0. The decision boundaries additionally carry a posterior credible band on their location. The area beyond a boundary equals the corresponding hit/false-alarm rate for every noise distribution: the likelihood evaluates the survivor function of the evidence distribution at the boundary, which is exactly that area.

Available for all four SDT models. sdt_yn() draws the criterion and sdt_rating() the K-1 confidence thresholds as boundary lines. sdt_mafc() and sdt_ranking() have no response criterion – the decision is a max/rank rule over the m alternatives – so only the densities are drawn (no boundary lines); there the noise density represents each of the m - 1 distractor alternatives and the signal density the target.

Because the densities depend only on d/sdratio, predictors that vary the criterion/thresholds *only* (e.g. a base-rate manipulation, as in broeder_schuetz_2009_e3) leave the densities unchanged: by default they are collapsed into a single panel with their several boundaries overlaid and colour-coded, rather than shown as repeated identical panels. Predictors that vary d/sdratio produce distinct density panels (faceted). For sdt_mafc/sdt_ranking the set size is likewise collapsed unless it predicts d. Use collapse to override the auto-detection.

Value

A data frame of class "bmm_sdt_latent" with columns x (the evidence axis), density, distribution ("noise" or "signal"), and any faceting (density) condition columns. A lines attribute holds the decision-boundary positions (columns position, lower, upper, marker, level, plus condition columns), or NULL for the criterion-free sdt_mafc/sdt_ranking models. When show_competitors = TRUE, a competitors attribute holds the max-of-distractors densities. An extra attribute carries the response-process parameters of the sdt_rating() versions as a posterior summary (parameter, mean, lower, upper, plus condition columns): for version = "dpsdt", Ro and Rn on the probability scale; for version = "metad", the M-ratio (mratio) and meta-d' (metad); NULL for version = "standard". The object also carries probs, model_class, dist, is_rating and conditions (the conditions of the density panels).

See Also

roc_sdt(), plot.bmm_sdt_latent()

Examples

```
## Not run:
# Three base-rate conditions shift the criterion, which identifies sdratio
dat <- expand.grid(id = 1:20, stimulus = c(0L, 1L),
                   condition = c("liberal", "neutral", "strict"))
dat$n_trials <- 100L
# simulated as bias, measured from the midpoint between the means (0 = boundary at the midpoint)
criteria <- c(liberal = -0.5, neutral = 0, strict = 0.5)
dat$n_old <- rsdt_yn(nrow(dat), dat$n_trials, dat$stimulus, d = 1.5,
                     bias = criteria[as.character(dat$condition)],
                     sdratio = 1.3)

fit <- bmm(
  formula = bmf(d ~ 1, criterion ~ 0 + condition, sdratio ~ 1),
  data = dat,
  model = sdt_yn(response = "n_old", stimulus = "stimulus",
                 n_trials = "n_trials"),
  cores = 4,
  backend = "cmdstanr"
)
```

```
latent <- latent_sdt(fit)
latent
plot(latent)

## End(Not run)
```

---

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 = c("custom", "ss", "cs"),
  ...
)
```

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`. Numbers named after the response categories, e.g. `c(corr = 1, other = 4)`, are matched to the categories by name. Numbers without names, or with other names, are taken in the order of `resp_cats`, and other names become the names of the columns bmm adds to the data. Column names given category names, e.g. `c(other = "n_other", corr = "n_corr")`, are matched by name as well, in any order. Custom activation formulas can use numbers by these column names; numbers without names or named after the categories are called `n_opt_<category>`.

choice_rule — The choice rule that should be used for the M3. The options are "softmax", "simple", or "gaussian". 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. The "gaussian" option adds independent standard normal noise to the activation of every candidate and chooses the candidate with the largest value (a Thurstonian rule: the choice is the maximum of noisy activations); "softmax" is the same with Gumbel noise. Activations under "gaussian" are on a smaller scale, and the difference is not one constant factor: `a` shrinks more than `c`, so a comparison of `c` with `a`, and effects of conditions that change the number of candidates (such as set size), can differ between the two rules. Sampling under "gaussian" costs about 40 times as much per iteration as under "softmax", because each probability is a 40-node numerical integral: a 40-subject, three-condition model with random intercepts took 15 seconds under "softmax" and about 4 minutes under "gaussian" end to end (4 chains, 1000 + 1000 iterations, Stan compilation included), 2.5 minutes with `threads = threading(2)`. In versions `ss` and `cs`, `c` and `a` have `normal(3, 1)` main priors under "softmax" and `normal(2, 1)` under "gaussian"; under "simple" `c` has `normal(3, 1)` and `a` has `normal(0, 1)`, on the log scale.

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(6), 880-932. https://doi.org/10.1037/rev0000159

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(3,1)
    - effects: normal(0,0.5)
    - sd: exponential(1)
  - c:
    - main: normal(3,1)
    - effects: normal(0,0.5)
    - sd: exponential(1)

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)
    - sd: exponential(1)
  - c:
    - main: normal(3,1)
    - effects: normal(0,0.5)

    * sd: exponential(1)
  - f:
    * main: logistic(0,1)
    * effects: normal(0,1)
    * sd: exponential(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:**

Missing values and reserved names:

A missing response count (NA) is counted as 0. If the category has options in that row, bmm() warns and says how many counts were replaced; if it has none (num_options is 0 in that row), the 0 is true and there is no warning, as for dist in oberauer_lewandowsky_2019_e1. A missing value in a column named in num_options is an error: enter 0 where there were no options. Data columns named Y, nTrials or Idx_<category> are refused because bmm() creates columns with these names; response categories may be called Y or nTrials.

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)
```

`meyer_grant_jakob_2025` *Ranking recognition data from Meyer-Grant & Jakob (2025)*

Description

Ranking signal-detection data from 60 subjects. On each trial, participants saw 3, 4, or 5 face images—one a studied target—and ranked them by perceived oldness; the rank assigned to the target is recorded. The set size varies across trials, so the data are aggregated to target rank-frequency counts per subject and set size, in the wide format `sdt_ranking()` consumes: one count column per rank position, with structural zeros where the rank exceeds the trial's set size. Fitting all set sizes jointly relies on the per-row set-size feature of `sdt_ranking()`: pass the set-size column to `m`.

Usage

```
meyer_grant_jakob_2025
```

Format

`meyer_grant_jakob_2025`**:**

A data frame with 180 rows (60 subjects x set sizes 3, 4, 5) and 7 columns:

id Factor with sequential codes `s01`–`s60` assigned by bmm; the original participant identifiers are not shipped

set_size Integer number of ranked items on the trial (3, 4, or 5)

rank1, rank2, rank3, rank4, rank5 Integer number of trials in which the target received that rank (rank1 = most likely target). Columns beyond `set_size` are structural zeros. The counts sum to 56 within each row.

Source

Meyer-Grant, C. G., & Jakob, M. (2025). Ranking tasks in recognition memory: A direct test of the two-high-threshold contrast model. *Journal of Experimental Psychology: General*, 154(5), 1445–1455. doi:10.1037/xge0001700. Data on OSF: `https://osf.io/gtzu7/`.

Examples

```
## Not run:
# Ranking SDT with set size varying per row: pass the set-size column to `m`
# so trials with 3, 4, and 5 alternatives are fit jointly.
model <- sdt_ranking(
  response = c("rank1", "rank2", "rank3", "rank4", "rank5"), m = "set_size"
)
fit <- bmm(
  formula = bmf(d ~ 1 + (1 | id)),
  data = meyer_grant_jakob_2025,
  model = model,
  backend = "cmdstanr"
)

## 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. https://doi.org/10.1038/nature06860
- **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: normal(0, 0.5)
    - effects: normal(0, 0.25)
    - sd: exponential(4)
  - kappa:
    - main: normal(2, 1)
    - effects: normal(0, 1)
    - sd: exponential(1)
  - thetat:
    - main: logistic(0, 1)
    - effects: normal(0, 2.5)
    - sd: exponential(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), Article 7. https://doi.org/10.1167/9.10.7
- **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: normal(0, 0.5)
    - effects: normal(0, 0.25)
    - sd: exponential(4)
  - kappa:

    * main: normal(2, 1)
    * effects: normal(0, 1)
    * sd: exponential(1)
  - thetat:
    * main: logistic(0, 1)
    * effects: normal(0, 2.5)
    * sd: exponential(1)
  - thetant:
    * main: logistic(0, 1)
    * effects: normal(0, 2.5)
    * sd: exponential(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")
```

---

model_citation *References for a measurement model*

Description

Returns the published source(s) of a model, for the reference list of a paper that uses it.

Usage

```
model_citation(x, ...)
```

Arguments

x	A model object (for example sdm(resp_error = "y")) or a bmmfit object returned by bmm().
...	Currently ignored.

Details

The references come from the model constructor of the installed version of bmm, so a fit made with an older version gets the current, corrected references. When the installed version no longer has a constructor for the model, or its references are empty, the copy stored in the fit or model object is returned instead.

Some versions of a model cite the papers that introduced them instead, for example `sdt_rating(version = "dpsdt")`.

A custom `m3()` model is cited with the paper that introduced the M3 framework. The model structure itself is defined by the user, so its source cannot be known to bmm. The returned vector then has the attribute `uncited`, naming the part of the model you have to cite yourself.

bmm itself is cited separately, see `citation("bmm")`.

Value

A character vector with one reference in APA style per element, or `character(0)` if no reference is available.

See Also

`citation()`

Examples

```
model_citation(sdm(resp_error = "y"))
model_citation(
  sdt_rating(response = paste0("r", 1:4), stimulus = "old", version = "dpsdt")
)
```

---

mpt *Multinomial Processing Tree (MPT) models*

Description

Multinomial Processing Tree (MPT) models measure the probabilities of discrete latent cognitive states from categorical responses. A model is specified as a set of trees created with `mpt_tree()`, one per distinct branch structure, such as one tree for targets and one for lures in a recognition test. Branch expressions that terminate in the same response category are summed within each tree, and parameters shared across trees are equated by giving them the same name. All latent probability parameters are estimated on an unconstrained latent scale (logit or probit); predictor formulas supplied via `bmmformula()` apply on that latent scale.

Usage

```
mpt(
  trees,
  tree_id = NULL,
  covariates = NULL,
  simplex = NULL,
  restrictions = NULL,
  links = "logit",
  ...
)
```

Arguments

trees	A single mpt_tree object or a list of mpt_tree objects. All trees must share the same set of response categories, counting those declared impossible.
tree_id	Character. Name of the data column whose values name the tree each observation belongs to; the values must match the tree names. Can be omitted for single-tree models. Effects of experimental conditions belong in the parameter formulas, not here; see Details.
covariates	Character vector. Names of data columns that appear in branch expressions but are not latent parameters, for example design-fixed guessing rates. Covariates pass into the model formulas unchanged and have no naming restrictions. bmm() stops when, with a row's covariate values, the branches of a tree do not sum to 1, leave (0, 1], are undefined on the rows of another tree, or when a covariate that a branch uses is missing. Every declared covariate needs a numeric column in the data, also one that no branch uses. The range is checked at the vertices of each branch's parameter box, which is exact when no term of the branch contains the same parameter twice (a * (1 - b) * G + (1 - a) * c). A parameter that does is also checked at 0.5, which is exact for a factor u * (1 - u) and approximate otherwise, e.g. for u * (1 - u)^2.
simplex	A character vector, or a list of character vectors, naming groups of parameters that are jointly constrained to sum to 1. Each group is reparameterized via stick-breaking: the last parameter of each group is derived as 1 minus the sum of the other members, and each other member p gets an unconstrained component praw that predictor formulas for p are applied to. The listing order sets the default prior and the meaning of the coefficients; see Details.
restrictions	Parameter restrictions in the string syntax of MPTinR and TreeBUGS, e.g. c("Dn = Do", "g = 0.5"), or as a named list, list(Dn = "Do", g = 0.5). A restriction either equates a parameter with another one (chains such as "G1 = G2 = G3" map all earlier names onto the last) or fixes it to a numeric constant ("g = 0.5"; bmm also reads fractions such as "g = 1/4", which MPTinR rejects). Constants must lie strictly between 0 and 1. The unnamed list of MPTinR and TreeBUGS, list("Dn = Do", "g = 0.5"), works as well. Restriction files are not read: in MPTinR and TreeBUGS a character vector names a restrictions file, in bmm it holds the restrictions. To use such a file, pass readLines(path); blank lines and lines starting with # are skipped, and a # after a restriction starts a comment, as in MPTinR. A restricted parameter is removed from the model; the Details

say how this differs from fixing a parameter in the formula. Order constraints (`"Do > Dn"`) are not supported here; see Details.

`links` The link function of the latent probability parameters: `"logit"` (default) or `"probit"`. A single string sets every parameter; a named list, `links = list(D = "probit")`, sets the named parameters and leaves the others at `"logit"`. Members of a `simplex` group and their `praw` components take no link of their own. A parameter switched to the other link, in the list or afterwards with `model$links$D <- "probit"`, takes that link's default priors (see Details).

`...` used internally for testing, ignore it

Details

- **Domain:** Categorical decision making, memory, and reasoning
- **Task:** Any task with categorical responses generated by a discrete processing-tree structure
- **Name:** Multinomial Processing Tree (MPT) models
- **Citation:**
  - Batchelder, W. H., & Riefer, D. M. (1999). Theoretical and empirical review of multinomial process tree modeling. Psychonomic Bulletin & Review, 6(1), 57-86. https://doi.org/10.3758/BF03210812

The data hold aggregated counts: one row per participant and tree (and per condition, if conditions are crossed with the trees), with one count column per response category. The count columns must be named exactly like the branch names given to `mpt_tree()` (`old` and `new` in the example below); `bmm()` finds them by these names, so no response variable is passed. The `tree_id` column names the tree of each row, and the counts of a row must come from trials of that tree. Wide tables with one column per category of an EQN file, as MPTinR and TreeBUGS take them, are reshaped with `mpt_data_reshape()`.

A separate tree is needed only when the *branch expressions* differ, that is, when the trial determines which latent processes apply or which response categories are reachable (targets versus lures, a response set with or without distractors, inclusion versus exclusion instructions). Experimental factors whose *effect on the parameters* you want to estimate belong in the parameter formulas, exactly as in every other `bmm` model: `bmf(D ~ 0 + condition)`. The two roles are independent, and one data column can serve both when a factor happens to change the structure as well. When several levels of a factor share a branch structure, add a column naming the tree and keep the factor for the formulas.

Equating and fixing parameters. Parameters are equated across trees by giving them the same name. Beyond that, there are two mechanisms, and they differ in when the constraint is decided. A *restriction* (`restrictions = c("Dn = Do", "g = 0.5")`) is applied when the model is built: it is substituted into the branch expressions before the parameters are identified, so the restricted parameter is not part of the model, gets no formula and no prior, cannot be a simplex member, and is listed by `print(model)` under `Restrictions:` but not by `parameter_info()`. A *constant in the formula* (`bmf(D ~ 1, g = 0.5)`) is applied at fit time: the parameter stays part of the model, is listed by `print()` under `Fixed:` and by `parameter_info()` with its value, and is freed again by giving it a formula. The value is mapped to the latent scale when the constant prior is built, so `default_prior()` shows `constant(0)` for a guessing rate of 0.5 under the logit link, and `summary()` lists it as `g_Intercept 0.00` under "Constant Parameters"; `prior_info()` and `parameter_info()` show the probability, 0.5. Members of a `simplex` group and their `praw` components cannot be fixed this way; fix

parameters outside the group instead. As a rule of thumb, a constant that defines the model goes into `restrictions`, a constant you compare against a free version goes into the formula; the section "Constraining parameters" of the MPT article walks through both.

Default priors. Under the logit link, intercepts and regression coefficients get a `logistic(0, 1)` prior and random-effect SDs get `exponential(1)`. Under the probit link, intercepts and coefficients get `normal(0, 1)` and SDs get `exponential(2)`. Group-level correlation matrices get `lkj(2)`. A parameter switched to the other link after construction (`model$links$D <- "probit"`) takes that link's priors unless its default prior was replaced. `default_prior()` lists the rows a given formula produces.

The default prior of a simplex group of K members is the uniform (symmetric Dirichlet(1)) distribution on the simplex: each member has prior mean 1/K, whatever the listing order. Under stick breaking this is a Beta(1, K - k) prior on stick k, which `default_prior()` shows as `logitbeta(1, K - k)` (or `probitbeta(1, K - k)`) on the intercept of the k-th `praw`; the last stick keeps `logistic(0, 1)` (or `normal(0, 1)`), which is the same distribution. The result is exact under both links, and under `0 + cond` it holds in every cell. Under `1 + cond` it holds in the reference cell only, because the effects keep their own prior. TreeBUGS instead puts a uniform prior on each branch parameter, under which the first member has prior mean 0.5 (0.50/0.25/0.25 for three members). To get it for a group `gA`, `gB`, `gC`, `gD`, pass `prior = brms::set_prior("logistic(0, 1)", nlpar = c("gAraw", "gBraw"))`, with `"normal(0, 1)"` under the probit link; without `coef`, it covers every cell of a `0 + cond` formula. A predictor on member `p` acts on the logit (or probit) of `p`'s share of the probability left over by the members listed before it. A predictor on the first member therefore scales all later members by the same factor, leaving their ratios unchanged. Read such effects on the probability scale with `native_parameters()`.

Identifiability. Printing the model ends with an identifiability check for intercept-only formulas. It compares the number of free parameters with the degrees of freedom (response categories minus 1, summed over trees) and computes the rank of the Jacobian of the category probabilities with respect to the free parameters, from exact derivatives at five interior test values (and five test values of each covariate); parameters fixed in the formula enter at their values. A rank below the number of free parameters means that a combination of the named parameters is not identified by the branch expressions, even when the count passes. The check is local: it holds at the test values, not at the boundaries of the parameter space. A test value with a non-finite derivative is left out, and the printout says how many; if `stats::D()` cannot differentiate a branch expression, the rank is not computed and the count decides. The rank sees the tree parameters only: after a non-linear formula, the rank line names that formula's parameter as not analysed.

`bmm()` warns when the rank is deficient and every formula is intercept-only. When a formula has a predictor or is non-linear, it only says that these may identify the parameters and that it did not check.

Example: separate detection parameters `Do` and `Dn` in a single old/new test give three free parameters and rank 2 of 3, so one combination of `Do`, `Dn` and `g` cannot be estimated; a second condition that varies the response bias identifies them.

Reading the estimates. `summary()` reports intercepts and regression coefficients on the latent (logit or probit) scale. `native_parameters()` returns the posterior draws of every parameter on the probability scale for each observed combination of the predictors, e.g. `native_parameters(fit, re_formula = NA, summary = TRUE)`; a difference between conditions is the difference of these draws. With `re_formula = NA` these are the probabilities of the median participant (the quantity TreeBUGS reports as `"mean"`), not the population mean of the participants' probabilities;

see the section on group-level effects in native_parameters().

Person effects written as (1 | id) in each parameter's formula are independent across parameters. A shared label, (1 |p| id) in every formula, estimates their correlations (prior lkj(2)), which corresponds to the latent-trait MPT model.

For posterior predictive checks, brms::posterior_predict() returns simulated counts (draws x rows x categories) and brms::posterior_epred() the expected counts; ppp() computes the T1 and T2 statistics of Klauer (2010) and their posterior predictive p-values from them. With newdata, brms::posterior_predict() needs the columns nTrials, Idx_<tree> and Poss_<category> that check_data() builds; fit$data contains them.

Non-linear formulas. Parameters that receive a non-linear predictor formula (a formula whose right-hand side references other formula parameters, e.g. D ~ Dmax * (1 - exp(-rate * ptime))) are not transformed by the link function: the user-supplied expression must keep the parameter within (0, 1). Sub-parameters of such formulas (e.g., Dmax and rate) are estimated on the identity scale with normal(0, 1) default priors on their intercepts and normal(0, 0.5) on their regression coefficients. Their random-effect SDs keep brms's default prior, student_t(3, 0, 2.5), because no single rate fits every scale such a formula can give them.

Order constraints between parameters (Do > Dn) are expressed by reparameterizing the larger parameter in the model formula, e.g. bmf(Do ~ Dn + (1 - Dn) * inv_logit(phi), Dn ~ 1, phi ~ 1); the section "Order constraints" of the MPT article walks through the recipe.

Names. Parameter and response category names must start with a letter and may contain only letters and digits, because brms does not allow underscores or dots in non-linear parameter names.

All latent parameters are non-linear parameters of the underlying brms model, and brms has no separate Intercept prior class for those: their intercept prior is stored as class = "b", coef = "Intercept". A prior given as class = "b", nlpar = "D" replaces the defaults of all coefficients of D, the intercept included; add coef = "Intercept" to set the intercept alone. Call default_prior() to see which rows a given formula produces.

Value

An object of class bmmodel

Examples

```
## Not run:
# two-high-threshold (2HTM) model of recognition memory
# one tree per probe type: targets (studied items) and lures (new items),
# each ending in the responses "old" and "new"
tree_target <- mpt_tree("target",
  old ~ D + (1 - D) * g,
  new ~ (1 - D) * (1 - g)
)
tree_lure <- mpt_tree("lure",
  old ~ (1 - D) * g,
  new ~ D + (1 - D) * (1 - g)
)

model <- mpt(
  trees = list(tree_target, tree_lure),
```

```
  tree_id = "item_type"
)

# simulate data for 20 participants with D = 0.7, g = 0.5:
# one row per participant x tree; count columns named after the branches
data <- data.frame(
  id = rep(1:20, each = 2),
  item_type = rep(c("target", "lure"), 20)
)
counts <- t(sapply(data$item_type, function(cond) {
  rmpt(
    n = 1, size = 50, pars = c(D = 0.7, g = 0.5),
    mpt_model = model, tree = cond, unpack = TRUE
  )
}))
data <- cbind(data, counts)

# predict both parameters by a fixed intercept and correlated random
# intercepts (the shared label |p| estimates the correlation)
formula <- bmf(
  D ~ 1 + (1 |p| id),
  g ~ 1 + (1 |p| id)
)

fit <- bmm(
  formula = formula,
  data = data,
  model = model,
  cores = 4
)

summary(fit)
# population-level D and g on the probability scale
native_parameters(fit, re_formula = NA, summary = TRUE)

## End(Not run)
```

`mpt_data_reshape` *Reshape wide MPT frequency tables to the long format of* `bmm()`

Description

MPTinR and TreeBUGS take one row per participant and one column per response category of the EQN file. `bmm()` takes one row per participant and tree, a column naming the tree, and one count column per *shared* response category (see `mpt()`). `mpt_data_reshape()` reshapes the first layout into the second for a given model.

Usage

```
mpt_data_reshape(data, model, id = NULL, columns = c("names", "mptinr"))
```

Arguments

data	A data frame or matrix with one row per participant. The count columns are read as described under columns; with columns = "names", all other columns except id are carried to the long data.
model	An mpt model from mpt(), mpt_from_eqn() or mpt_from_string(). For a model from mpt_from_eqn() the mpt_renaming attribute maps the labels of the EQN file onto the response categories of the model. For any other model, the columns of data are named <tree>.<category>, e.g. target.old, with the tree names and response categories of the model.
id	Character. The name of the participant column in data, or NULL (the default) to number the rows of data, as id = seq_len(nrow(data)) in a column named id. The participants must be unique.
columns	Character. How the count columns of data are identified: "names" (the default) by their names, as in TreeBUGS data; "mptinr" by their position, as in MPTinR data. See Details.

Details

Named columns (columns = "names"). For a model from mpt_from_eqn(), a column is matched to a category by the EQN label (hit), or, when trees share labels, by the tree name and label (target.hit, with the bmm name of the tree); the qualified name wins where both exist. A label used in several trees must be qualified. Labels that the categories argument of mpt_from_eqn() merged onto one response category are summed. All columns of data that are neither count columns nor id are carried unchanged to every long row, e.g. between-participant covariates.

Positional columns (columns = "mptinr"). MPTinR data for an EQN model have no column names. Every column of data except id is read as a count, in the order of check.mpt() in MPTinR (version 1.14.1), elements eqn.order.trees and eqn.order.categories: trees sorted by their label in the EQN file and, within each tree, the category labels of the file sorted. No other column can be carried; add covariates afterwards, e.g. with merge() on the participant column. The help page of check.mpt() states only that these elements give the order; the sorting rule is that of its internal .get.EQN.model.order(), which sorts the columns that read.table() returns for the file: numerically if all tree labels (all category labels) are numbers, else as character strings in the collation of the R session. mpt_data_reshape() applies the same rule, so run it in the session where MPTinR's order was determined if the labels mix upper and lower case. This mode needs a model from mpt_from_eqn(): MPTinR's own model format orders the columns by the lines of the model text, and mpt() reorders the branches of every tree to the categories of the first, so the line order cannot be recovered from the model. For other models, name the columns <tree>.<category> and use columns = "names".

Impossible categories. A response category that a tree cannot produce (impossible in mpt_tree(), mpt_from_string() and mpt_from_eqn()) has no column in data and gets 0 in the long rows of that tree, which bmm() accepts.

Counts. The count columns must be numeric, without missing values, and hold non-negative whole numbers; the long data hold them as integers.

Checks. The function cannot tell a correct reshape from one that swaps columns of equal meaning. Run bmm_data_check() on the result, and compare one participant's long rows with the wide row (see the Examples of mpt_from_eqn()).

Value

A data frame with one row per participant and tree, participants first. Its columns are the participant column (`id`), the tree column named by the `tree_id` of the model (absent in a model without it), the carried columns, and one integer count column per response category of the model.

See Also

`mpt_from_eqn()` and `mpt_from_string()` for importing the model itself; `mpt()` for the data layout `bmm()` expects; `bmm_data_check()` to check the result before fitting.

Examples

```
eqn_file <- tempfile(fileext = ".eqn")
writeLines(c(
  "6",
  "target  hit   D_o",
  "target  hit   (1-D_o)*g",
  "target  miss  (1-D_o)*(1-g)",
  "lure    fa    (1-D_n)*g",
  "lure    cr    D_n",
  "lure    cr    (1-D_n)*(1-g)"
), eqn_file)
model <- mpt_from_eqn(
  eqn_file,
  restrictions = "D_n = D_o",
  categories = c(hit = "yes", fa = "yes", miss = "no", cr = "no"),
  tree_id = "item_type"
)

# TreeBUGS data: one named column per category label of the EQN file.
# Columns that are no counts, here a between-participant factor, are
# carried to the long data
wide <- data.frame(
  subject = c("s1", "s2"), group = c("a", "b"),
  hit = c(30, 25), miss = c(10, 15), fa = c(8, 12), cr = c(32, 28)
)
long <- mpt_data_reshape(wide, model, id = "subject")
long
bmm_data_check(bmf(Do ~ 1, g ~ 1), long, model)

# MPTinR data: unnamed columns, sorted by tree, then by label (cr, fa for
# tree lure, then hit, miss for tree target)
wide_mptinr <- rbind(c(32, 8, 30, 10), c(28, 12, 25, 15))
identical(
  mpt_data_reshape(wide_mptinr, model, columns = "mptinr"),
  mpt_data_reshape(wide[c("cr", "fa", "hit", "miss")], model)
)

# trees with response categories that only one tree can produce
model_pc <- mpt(
  list(
    mpt_tree("pairs", list(E1 = "c * r", E2 = "1 - c * r"),
```

```
            impossible = "F1"),
    mpt_tree("singles", list(F1 = "u", E2 = "1 - u"),
            impossible = "E1")
  ),
  tree_id = "item_type"
)
wide_pc <- data.frame(
  pairs.E1 = c(20, 22), pairs.E2 = c(20, 18),
  singles.F1 = c(25, 24), singles.E2 = c(15, 16)
)
mpt_data_reshape(wide_pc, model_pc)
```

`mpt_from_eqn` *Create an MPT model from an EQN model file*

Description

Reads an EQN model file, the format used by multiTree, MPTinR, and TreeBUGS, and returns the corresponding `mpt()` model object; model files in MPTinR's own "easy" format are read by `mpt_from_string()`. Each line of an EQN file contains three whitespace-separated fields – tree, response category, branch probability expression – and branch lines that terminate in the same response category of a tree are summed.

Usage

```
mpt_from_eqn(
  file,
  restrictions = NULL,
  categories = NULL,
  tree_id = NULL,
  covariates = NULL,
  simplex = NULL,
  links = "logit",
  impossible = NULL,
  tree_names = NULL
)
```

Arguments

`file`	Character. Path to the EQN file. How its first line is read is described in Details.
`restrictions`	Parameter restrictions as in `mpt()`, written with the parameter names used in the EQN file, e.g. `c("D_n = D_o", "g = 0.5")` or `list(D_n = "D_o", g = 0.5)`. They mirror the `restrictions` argument of MPTinR and TreeBUGS and are renamed together with the parameters. Restriction files are not read: pass their lines (the Examples of `mpt_from_string()` read one with `readLines()`).

`categories` A named character vector mapping response category names used in the EQN file onto the shared response categories of the model (see Details), e.g. `c(hit = "yes", fa = "yes", miss = "no", cr = "no")`. Categories not listed keep their name. Mapped values and unlisted names alike are sanitized (underscores and dots removed, see Details), so `c(hit = "y_es")` gives `yes`. Labels that only carry their tree name as a prefix (`WC_Correct`, `LC_Correct`) need no mapping, because the prefix is removed. Labels that are not valid bmm names, such as numeric labels, must be mapped. Mapping two categories of the same tree onto one name merges them: their branch lines are summed, and a message says so.

`tree_id` Character. Name of the data column whose values name the tree each observation belongs to (see `mpt()`). Add this column to the data yourself: for each row it holds the name of the tree the counts come from, as printed by the model (the file's label, or its replacement from `tree_names`).

`covariates`, `simplex`, `links`
passed to `mpt()`. Covariate names are written as in the EQN file and are excluded from the renaming; simplex groups use the bmm names. `print(model)` lists the symbols that became parameters.

`impossible` A named list, `list(<tree> = c("<category>", ...))`, of the response categories that cannot occur in a tree (see `mpt_tree()`), written with the bmm names of the trees and categories, i.e. the `bmm_name` column of the renaming map; `print(model)` lists the shared categories under "Response". Trees not listed have no impossible categories. When the trees do not share their categories and `impossible` does not cover the difference, the error names the categories each tree lacks.

`tree_names` A named character vector mapping tree labels of the EQN file onto tree names in bmm, e.g. `c("1" = "target", "2" = "lure")`. bmm tree names must start with a letter and contain only letters, digits, or underscores, so files with numeric tree labels need this mapping. Labels not listed keep their name.

Details

File format. Blank lines and lines that start with # are skipped. If the first remaining line holds a single integer and nothing else, it is read as the equation count of classical EQN files and skipped, with a warning when it differs from the number of equation lines that follow. Any other first line is read as an equation. Note that TreeBUGS always ignores the first line of an EQN file, whatever it holds, and so does MPTinR with `model.type = "eqn"` (its default for `.eqn` files; with `"eqn2"` it reads the first line as the equation count). bmm therefore reads a file without a count line in full, while those programs drop its first equation; a title line produces an error that says to delete it or start it with #. A file that starts with the equation count, or with # lines followed by equations, is read the same way by all three. A line with fewer than three fields, or an expression that cannot be parsed, is an error that names the line; a tree whose branches do not sum to 1 at numeric test values is an error that names the tree.

Response categories. In bmm, every observation (row) belongs to one tree, and all trees share the same response categories. Classical EQN files often label the response categories per tree instead (e.g., `hit`/`miss` in the old-item tree and `fa`/`cr` in the new-item tree of a recognition model). Use the `categories` argument to map such tree-specific labels onto the shared response categories (e.g., the response options "yes" and "no"). If a category cannot occur in some trees (e.g., the pair and singleton trees of the pair-clustering model), declare it with the `impossible` argument instead.

Names. Because brms does not allow underscores or dots in non-linear parameter names, parameter and response category names are sanitized: a leading `<tree>_` prefix is removed from category names that are not mapped with `categories`, and all remaining underscores and dots are stripped. Two labels of the file, or two different values within one tree, that become the same name are an error; mapped values that become the same name in different trees are that one shared category. Names are case-sensitive, so `P_G1` and `P_g1` stay distinct (`PG1`, `Pg1`). Tree labels are kept, unless `tree_names` maps them. The formulas passed to `bmm()` use the bmm names (`bmf(Do ~ 1)` for a file parameter `D_o`; `bmf()` is short for `bmmformula()`); the `parameter` rows of the renaming map translate them, and `print(model)` lists them. Of the arguments that name parameters, only `restrictions` and `covariates` use the file's names.

Renaming map. The `mpt_renaming` attribute of the returned model is a data frame with one row per parameter, per tree, and per category of each tree, renamed or not, and the columns `kind` (`"parameter"`, `"tree"`, or `"category"`), `tree` (the bmm tree name for categories, `NA` otherwise), `file_name`, `bmm_name`, and `renamed` (`TRUE` where the two names differ). The names that changed are also reported in a message.

Data for an imported model. `bmm()` expects one row per participant and tree: a column named by `tree_id` holding the bmm tree names, and one count column per shared response category, named by `bmm_name` in the renaming map (see `mpt()`). A category that is `impossible` in a tree gets 0 (or `NA`) in the rows of that tree. TreeBUGS data are wide: one row per participant, one column per category label of the EQN file. MPTinR data for an EQN model are unnamed, and their columns do not follow the file: the trees and, within each tree, the category labels are sorted (alphabetically, or numerically when all labels in the file are numbers); `MPTinR::check.mpt("model.eqn")` returns the order in its elements `eqn.order.trees` and `eqn.order.categories`. (For MPTinR's own model format the columns follow the lines instead; see `mpt_from_string()`.) `mpt_data_reshape()` reshapes both layouts with the renaming map, sums the labels that `categories` merged onto one response category, and gives 0 to the categories a tree cannot produce (see Examples).

bmm cannot tell a reshape that swaps tree labels or category columns from a correct one: such data fit without an error. The row sums of the count columns are the trials per row; they cannot show swapped columns, and they show swapped tree labels only when the trees differ in their number of trials. Comparing the response proportions per tree with the design (targets must get more "yes" responses than lures) catches swapped tree labels and response columns swapped in every tree, but not always a column slipped or a map entry wrong in one tree, nor a swap between two trees of the same kind; so also compare one participant's long rows with the wide row. `bmm_data_check()` runs the remaining checks (columns, tree labels, counts in impossible cells) without a fit and prints `Hard checks: passed` when they hold.

Provenance. The returned model carries an `mpt_source` attribute, `list(file = , md5 = , lines = )`: the full path of the EQN file, its MD5 checksum, and the lines as read, so that a saved fit records which file it was built from.

Value

An object of class `bmmodel` (see `mpt()`), with the attributes `mpt_renaming` and `mpt_source` (see Details).

See Also

`mpt_from_string()` for model text in MPTinR's own format; `mpt()` and `mpt_tree()` for building the trees in R, including the data layout `bmm()` expects.

Examples

```
eqn_file <- tempfile(fileext = ".eqn")
writeLines(c(
  "6",
  "target  hit   D_o",
  "target  hit   (1-D_o)*g",
  "target  miss  (1-D_o)*(1-g)",
  "lure    fa    (1-D_n)*g",
  "lure    cr    D_n",
  "lure    cr    (1-D_n)*(1-g)"
), eqn_file)
# restrictions use the file's names; the formulas use the bmm names
model <- mpt_from_eqn(
  eqn_file,
  restrictions = "D_n = D_o",
  categories = c(hit = "yes", fa = "yes", miss = "no", cr = "no"),
  tree_id = "item_type"
)
attr(model, "mpt_renaming")

# TreeBUGS data: one row per participant, columns named by the category
# labels of the EQN file, no participant column. mpt_data_reshape() reshapes
# them to bmm's one row per participant and tree, through the map; the
# categories a tree cannot produce get 0
wide <- data.frame(
  hit = c(30, 25), miss = c(10, 15), fa = c(8, 12), cr = c(32, 28)
)
long <- mpt_data_reshape(wide, model)
long
# the row sums are the trials per row (40 here). The proportions catch
# swapped tree labels and swapped response columns (targets must get
# more "yes" than lures), but not always a column slipped in one
# tree: also compare one participant's long rows with the wide row
rowSums(long[c("yes", "no")])
tapply(long$yes / rowSums(long[c("yes", "no")]), long$item_type, mean)
long[long$id == 1, ]
wide[1, ]
bmm_data_check(bmf(Do ~ 1, g ~ 1), long, model)

# MPTinR data for an EQN model are unnamed and sorted by tree, then by
# label: MPTinR::check.mpt(eqn_file)$eqn.order.categories lists the
# labels (here cr, fa for tree new, then hit, miss for tree old)
wide_mptinr <- rbind(c(32, 8, 30, 10), c(28, 12, 25, 15))
identical(mpt_data_reshape(wide_mptinr, model, columns = "mptinr"), long)
```

`mpt_from_string` *Create an MPT model from an MPTinR-style model string*

Description

Reads a model definition in the "easy" format of MPTinR model files and returns the corresponding `mpt()` model object; EQN files (multiTree, MPTinR, TreeBUGS) are read by `mpt_from_eqn()`. Each line is one branch probability expression; blank lines separate trees. The response category of each line is given either by an inline comment (`expression # category`) or by the `categories` argument. Lines that terminate in the same response category are summed.

Usage

```
mpt_from_string(
  text,
  restrictions = NULL,
  tree_names,
  categories = NULL,
  tree_id = NULL,
  covariates = NULL,
  simplex = NULL,
  links = "logit",
  impossible = NULL
)
```

Arguments

`text`	Character. The model definition, either as a single string or as a vector of lines (e.g., from `readLines()`).
`restrictions`	Parameter restrictions as in `mpt()`, written with the parameter names of the model text, e.g. `"g = 0.5"` or `list(g = 0.5)`. They mirror the `restrictions` argument of MPTinR and TreeBUGS. Restriction files are not read: pass their lines (see Examples).
`tree_names`	Character vector with one name per tree block. The names are matched to the blocks by position: the first name goes to the first block, and so on. bmm cannot check this order, so names given in a different order than the blocks import a different model without an error. For multi-tree models the names must match the values of the tree identifier column in the data.
`categories`	The response categories of lines without an inline `# category` comment, matched to the lines of a tree by position: the i-th entry is the category of the i-th line. Either a list by tree, `list(<tree> = c("<category>", ...))`, with one vector per tree that has such lines, or one unnamed character vector shared by all trees. A tree's vector in the list must not be longer than the tree. A shared vector must not be longer than the longest tree; a shorter tree takes its leading entries, which suits trees that lack the last categories (see `impossible`). If the trees use different categories, as in the pair-clustering model, use the list. Either form needs an entry for every line without an inline comment; an inline `# category` comment takes precedence over the entry for its line.
`tree_id`	Character. Name of the data column whose values name the tree each observation belongs to (see `mpt()`). Add this column to the data yourself: for each row it holds the name, as given in `tree_names`, of the tree the counts come from.

`covariates`, `simplex`, `links`
: passed to `mpt()`. `print(model)` lists the symbols that became parameters.

`impossible`
: A named list with the response categories that cannot occur in a tree, one element per such tree: `list(<tree> = c("<category>", ...))` (see the `impossible` argument of `mpt_tree()`). Trees and categories are named as in the returned model, i.e. by `tree_names` and by the category labels of the lines. Trees that are not listed have no impossible categories.

Details

Comments. A line whose first non-blank character is # is a comment and is skipped. It does not separate trees; only blank lines do. On a line with an expression, the text after # is the response category of that line. This differs from MPTinR, which ignores everything after a # and treats a line that starts with # as a blank line, so that there comment lines can separate trees. An MPTinR model file whose trees are separated only by comment lines, or whose inline comments are not category labels, has to be edited before it is imported.

Trees with different response categories. In bmm all trees share the same response categories. If a category cannot occur in some trees (e.g., the pair and singleton trees of the pair-clustering model), declare it with the `impossible` argument. If the trees only label the same responses differently, give those lines the same category label.

Names. Parameter names are used as written in the model text and are case-sensitive, so the formulas passed to `bmm()` name them the same way (`bmf(D ~ 1)`; `bmf()` is short for `bmmformula()`). They must be valid bmm names (letters and digits, see `mpt()`); unlike `mpt_from_eqn()`, this function does not rename them. The import fails when the branches of a tree do not sum to 1 at numeric test values.

Data for an imported model. The Details of `mpt_from_eqn()` describe the layout `bmm()` expects (one row per participant and tree, a tree column, one count column per response category, 0 for impossible categories) and the checks to run before fitting; they apply to models from this function as well, with the tree names of `tree_names` and the category labels of the lines. The data MPTinR fits to a model file in this format are wide and positional: one row per participant (or one vector for aggregated counts) and one unnamed column per line of the model file, in file order, because each line is one response category. (For EQN files MPTinR sorts the columns instead; see `mpt_from_eqn()`.) Name the columns `<tree>.<category>`; `mpt_data_reshape()` then stacks the trees and gives 0 to the categories a tree cannot produce (see Examples).

Provenance. The returned model carries an `mpt_source` attribute, `list(text = )`, with the model definition as one string.

Value

An object of class `bmmodel` (see `mpt()`), with an `mpt_source` attribute holding the imported text.

See Also

`mpt_from_eqn()` for EQN files (multiTree, MPTinR, TreeBUGS); `mpt()` and `mpt_tree()` for building the trees in R, including the data layout `bmm()` expects.

Examples

```
model_2htm <- "
# two-high-threshold model
D + (1 - D) * g        # old
(1 - D) * (1 - g)      # new

(1 - D) * g            # old
D + (1 - D) * (1 - g)  # new
"
model <- mpt_from_string(
  model_2htm,
  tree_names = c("target", "lure"),
  tree_id = "item_type"
)
model

# fix the guessing rate as in MPTinR / TreeBUGS
mpt_from_string(
  model_2htm,
  tree_names = c("target", "lure"),
  tree_id = "item_type",
  restrictions = "g = 0.5"
)

# pair clustering, written as in MPTinR without category labels: pairs and
# singletons have different response categories
model_pc <- "
c * r
(1 - c) * u * u
2 * (1 - c) * u * (1 - u)
c * (1 - r) + (1 - c) * (1 - u) * (1 - u)

u
1 - u
"
mpt_from_string(
  model_pc,
  tree_names = c("pairs", "singles"),
  categories = list(
    pairs = c("E1", "E2", "E3", "E4"),
    singles = c("F1", "F2")
  ),
  tree_id = "item_type",
  impossible = list(pairs = c("F1", "F2"), singles = c("E1", "E2", "E3", "E4"))
)

# an MPTinR model file read with readLines() keeps the blank lines that
# separate the trees. A restrictions file is not read by bmm: drop its
# comments and blank lines and pass the remaining lines.
model_file <- tempfile(fileext = ".model")
writeLines(model_2htm, model_file)
restr_file <- tempfile(fileext = ".restr")
```

```
writeLines(c("# fix the guessing rate", "g = 0.5"), restr_file)
restr <- trimws(sub("#.*", "", readLines(restr_file)))
model <- mpt_from_string(
  readLines(model_file),
  tree_names = c("target", "lure"),
  tree_id = "item_type",
  restrictions = restr[nzchar(restr)]
)

# MPTinR data: one unnamed column per line of the model file, in file
# order. Name the columns by tree and category, then let mpt_data_reshape()
# stack the trees to get bmm's one row per participant and tree.
wide <- rbind(c(30, 10, 8, 32), c(25, 15, 12, 28))
colnames(wide) <- c("target.old", "target.new", "lure.old", "lure.new")
long <- mpt_data_reshape(wide, model)
long
# the row sums are the trials per row (40 here), but they cannot show
# swapped columns or tree labels: compare one participant with the wide
# row, then run the data checks (bmm_data_check() prints
# "Hard checks: passed" when the columns, tree labels and counts fit)
rowSums(long[c("old", "new")])
long[long$id == 1, ]
wide[1, ]
bmm_data_check(bmf(D ~ 1), long, model)
```

`mpt_tree` *Create a tree for a Multinomial Processing Tree (MPT) model*

Description

Specifies a single tree of a multinomial processing tree model by naming the tree and providing one branch-probability expression per response category. Trees created with `mpt_tree()` are combined into a full model specification with `mpt()`.

Usage

```
mpt_tree(name, branches, ..., impossible = NULL)
```

Arguments

`name`	Character. Label for the tree. For models with multiple trees, the label identifies the tree in the data: the column named by the `tree_id` argument of `mpt()` holds one such label per observation. A tree stands for one kind of trial, such as targets or lures in a recognition test, not for a response; the responses are the branch names.
`branches`	Either a named list of character strings or the first of several formulas. Each list element (or formula) gives the branch probability expression for one response category; the element names (or the left-hand sides) are the response categories. The expressions can use latent parameters (e.g., `D + (1 - D) * g`), numeric constants, and declared covariates (see `mpt()`).

`...` Further formulas of the form `category ~ expression`, one per response category, when `branches` is a formula; they take no argument names. A call uses either formulas or a named list of strings, not a mix; both spellings give the same tree (see the examples).

`impossible` Character vector. Response categories that cannot occur in this tree, for example distractor responses when no distractors were presented. Such categories get no branch expression, and the remaining branches must sum to 1 on their own. It follows `...`, so it has to be named in full.

Details

Nothing in the formulas is evaluated: the left-hand side must be a single response category name, the right-hand side is kept as an expression (with constant arithmetic folded, see below), and the formula environment is dropped. A one-sided formula, a left-hand side that is not a name, and a mix of formulas and strings are errors.

Constant arithmetic such as `1/4` or `1/(2*2)` is folded into a decimal literal (`0.25`) when the tree is created. Stan compiles a bare integer fraction as integer division (`1/4 == 0`), which would silently corrupt the likelihood.

Categories listed in `impossible` keep their column in the data and their place in the response matrix, but their probability is set to zero for the observations belonging to this tree. Because the model is estimated on the log scale, "zero" is implemented as a large negative linear predictor rather than a literal zero; no probability has to be taken away from the remaining branches to compensate.

Value

An object of class `mpt_tree`

Examples

```
# the target tree of the two-high-threshold model: targets (studied items)
# end in the responses "old" and "new"
tree_target <- mpt_tree(
  "target",
  old ~ D + (1 - D) * g,
  new ~ (1 - D) * (1 - g)
)
tree_target

# the same tree with the branches as a named list of strings
tree_target_strings <- mpt_tree(
  name = "target",
  branches = list(
    old = "D + (1 - D) * g",
    new = "(1 - D) * (1 - g)"
  )
)
identical(tree_target, tree_target_strings)

# a three-alternative recognition task in which lures are shown on some
# trials only: `lure` responses cannot occur on the others
```

```
tree_lures <- mpt_tree(
  "lures",
  target ~ D * b + (1 - D) * (1/3),
  lure ~ D * (1 - b) + (1 - D) * (1/3),
  new ~ (1 - D) * (1/3)
)

# the branches of the lure-free tree sum to 1 without the `lure` category;
# no probability is reserved for it
tree_nolures <- mpt_tree(
  "nolures",
  target ~ D + (1 - D) * (1/2),
  new ~ (1 - D) * (1/2),
  impossible = "lure"
)
tree_nolures

# a `tree` column in the data holds "lures" or "nolures" per observation;
# effects of experimental conditions go in the parameter formulas instead,
# e.g. bmf(D ~ 0 + cond)
model <- mpt(list(tree_lures, tree_nolures), tree_id = "tree")
```

`mptdist` *Distribution functions for Multinomial Processing Tree (MPT) models*

Description

Density and random generation functions for multinomial processing tree models specified with `mpt()`. Please note that these functions are currently not vectorized.

Usage

```
dmpt(x, pars, mpt_model, tree = NULL, log = TRUE, covariates = NULL, ...)

rmpt(
  n,
  size,
  pars,
  mpt_model,
  tree = NULL,
  unpack = FALSE,
  covariates = NULL,
  ...
)
```

Arguments

`x`	Integer vector of length `K`, where `K` is the number of response categories, giving the number of observed responses per category. If `x` is a named vector, its

	names are matched to the response categories and the vector is reordered accordingly; an unnamed vector is read in the order of the model's response categories, `mpt_model$resp_vars$resp_cats`, which is the order of the first tree given to `mpt()` (`mpt()` stores the branches of every tree in that order, whatever order they were written in). `rmpt()` returns its columns in the same order.
`pars`	A named vector or list with the values of at least the latent parameters appearing in the branch expressions of the selected tree, on the probability scale and strictly between 0 and 1, as for fixed parameters in `bmm()`. Values for parameters the tree does not use are ignored, so one vector can serve every tree of the model. Parameters of a simplex group must sum to 1.
`mpt_model`	A `bmmodel` object created with `mpt()` specifying the model that densities or random samples should be generated for.
`tree`	Character. For models with multiple trees, the name of the tree to compute probabilities for. Can be omitted for single-tree models.
`log`	Logical; if `TRUE` (default), densities are returned on the log scale.
`covariates`	A named vector or list with the values of all covariates declared in the model that appear in the branch expressions of the selected tree.
`...`	can be used to pass additional values used in the branch expressions, as an alternative to the `covariates` argument.
`n`	Integer. Number of observations to generate data for.
`size`	The total number of observations across all response 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`.

Value

`dmpt` gives the multinomial density of a vector of response counts, and `rmpt` generates random response counts for the response categories.

Note

Unlike the densities of the circular models in this package (`dsdm()`, `dmixture2p()`, `dmixture3p()`, `dimm()`) and unlike `stats::dmultinom()`, `dmpt()` returns the log density by default, matching `dm3()`.

References

Batchelder, W. H., & Riefer, D. M. (1999). Theoretical and empirical review of multinomial process tree modeling. Psychonomic Bulletin & Review, 6(1), 57-86. https://doi.org/10.3758/BF03210812

Examples

```
tree_target <- mpt_tree("target", list(
  old = "D + (1 - D) * g",
  new = "(1 - D) * (1 - g)"
))
```

```
tree_lure <- mpt_tree("lure", list(
  old = "(1 - D) * g",
  new = "D + (1 - D) * (1 - g)"
))
model <- mpt(list(tree_target, tree_lure), tree_id = "item_type")

dmpt(
  x = c(35, 15), pars = c(D = 0.7, g = 0.5),
  mpt_model = model, tree = "target"
)
rmpt(
  n = 10, size = 50, pars = c(D = 0.7, g = 0.5),
  mpt_model = model, tree = "lure"
)
```

`mratio` *Metacognitive efficiency (M-ratio) from a fitted meta-d' SDT model*

Description

Extracts the posterior of the M-ratio ($\text{meta-d}'/d'$) and of meta-d' from a `sdt_rating()` model fit with `version = "metad"`. That model estimates the log M-ratio (`logmratio`) directly, so the M-ratio is `exp(logmratio)` and meta-d' is `exp(logmratio) * d`. The M-ratio is the standard measure of metacognitive efficiency (Maniscalco & Lau, 2012; Fleming, 2017): 1 is ideal metacognition, below 1 is inefficiency, and above 1 is hyper-efficiency. Type-1 and type-2 sensitivity are reported on one scale, d' or, once `sdratio` is estimated, d_a (see `sdt_sensitivity()`), so the ratio is unaffected by `sdratio`. The type-1 criterion keeps its location in the meta-d' space, as in the HMeta-d model equations (Fleming, 2017, Appendix); maximum-likelihood meta-d' constrains meta-c' = c' (Maniscalco & Lau, 2014), so the two differ when the criterion is far from the midpoint.

Usage

```
mratio(fit, conditions = NULL, probs = c(0.025, 0.975), ...)
```

Arguments

`fit`	A `bmmfit` object returned by `bmm()` from an SDT model.
`conditions`	Optional data frame of predictor values at which to evaluate the M-ratio. Its columns must be population-level predictors of `logmratio` or `d` in the formula; grouping variables of random effects are refused. If `NULL` (default), unique predictor combinations are derived from the data, and a fit without such a predictor gives one row.
`probs`	Numeric vector of length 2. Lower and upper quantiles for the credible interval (default `c(0.025, 0.975)`).
`...`	Additional arguments passed to `brms::posterior_linpred()`, such as `draw_ids` to use a subset of the posterior draws. `ndraws` is refused: each parameter is drawn by its own call, so random subsets would not match across parameters.

Value

A data frame of class "bmm_sdt_mratio" summarising the posterior: columns parameter ("mratio" or "metad"), mean, median, lower, upper (the credible-interval bounds at probs), and any condition columns. The full per-draw posteriors are kept in the draws attribute (long format: parameter, value, .draw, plus condition columns) for downstream computation such as plotting or condition contrasts.

References

Maniscalco, B., & Lau, H. (2012). A signal detection theoretic approach for estimating metacognitive sensitivity from confidence ratings. *Consciousness and Cognition*, *21*(1), 422–430. doi:10.1016/j.concog.2011.09.021

Maniscalco, B., & Lau, H. (2014). Signal detection theory analysis of type 1 and type 2 data: meta-d', response-specific meta-d', and the unequal variance SDT model. In S. M. Fleming & C. D. Frith (Eds.), *The cognitive neuroscience of metacognition* (pp. 25–66). Springer.

Fleming, S. M. (2017). HMeta-d: hierarchical Bayesian estimation of metacognitive efficiency from confidence ratings. *Neuroscience of Consciousness*, *2017*(1), nix007. doi:10.1093/nc/nix007

See Also

sdt_rating(), latent_sdt()

---

native_parameters *Posterior draws of model parameters on the native scale*

Description

bmm samples all parameters on their link scale (log for precision and boundary parameters, logit or softmax for mixture weights, tan_half for circular locations). native_parameters() returns posterior draws of those parameters on the scale they are interpreted and reported on, evaluated over a grid of predictor values.

Because draws are returned rather than summaries, any contrast is ordinary arithmetic on the draws, with the baseline made explicit by the grid.

Usage

```
native_parameters(
  x,
  newdata = NULL,
  pars = NULL,
  re_formula = NULL,
  scale = c("native", "sampling"),
  ndraws = NULL,
  draw_ids = NULL,
  summary = FALSE,
  prob = 0.95,
```

```
  robust = FALSE,
  ...
)
```

Arguments

x	A bmmfit object returned by bmm().
newdata	Data to evaluate the parameters on. If NULL (the default), the unique observed combinations of the parameters' predictors are used, taken as rows of the model's own data. Columns you do not supply are filled from the first row of the model data.
pars	Character vector of parameters to return. If NULL (the default), all parameters of the model are returned, including those fixed to a constant. This filters the output only; parameters transformed jointly are always computed together.
re_formula	Which group-level effects to include, as in brms::posterior_linpred(). See Details.
scale	Either "native" (the default) to apply the model's inverse link functions, or "sampling" to return the untransformed linear predictor.
ndraws	Number of posterior draws to use. If NULL (the default), all draws are used. The same draws are used for every parameter.
draw_ids	Indices of the draws to use. Overrides ndraws and makes the result reproducible.
summary	Logical. If FALSE (the default), return draws. If TRUE, return posterior summaries computed *after* the transformation.
prob	Probability mass of the credible interval when summary = TRUE.
robust	Logical. If TRUE, summary = TRUE reports the median and median absolute deviation instead of the mean and standard deviation.
...	Further arguments passed to brms::posterior_linpred(), such as allow_new_levels and sample_new_levels.

Value

If summary = FALSE, a data.frame with one row per draw, grid cell and parameter, with columns .chain, .iteration, .draw, the grid variables, parameter and value. The draw indices refer to the draws of x, so they can be joined with the output of posterior::as_draws_df(x). If summary = TRUE, a data.frame with one row per grid cell and parameter, with columns Estimate, Est.Error and the interval bounds.

Transform first, then summarise

Quantiles are preserved by monotone inverse links but means are not, so the median and the credible interval of a native-scale parameter are exact while its mean is not the inverse link of the linear predictor's mean. This function transforms the draws and only then summarises, which is why summary = TRUE is not the same as transforming the output of summary.bmmfit().

The same order is what makes summary = TRUE correct for the mixture weights, whose softmax is not an elementwise map at all: summarising the transformed draws reports quantiles of the weight's own marginal posterior, whereas transforming the summaries would report the softmax of three separate quantiles, which is not a quantile of anything.

Transforming coefficients is not the same thing

The inverse link applies to the **linear predictor** of a grid cell, not to an individual regression coefficient. For a model with `kappa ~ condition`, `exp(b_kappa_conditionB)` is a multiplicative factor, not kappa in condition B; kappa in condition B is `exp(b_kappa_Intercept + b_kappa_conditionB)`. Building the grid is exactly what removes this step, which is why contrasts are taken between rows of the output rather than read off the coefficients.

Group-level effects

`re_formula = NULL` (the default) returns subject-specific parameters and adds the grouping variables to the grid. `re_formula = NA` sets all group-level effects to zero, which under a non-identity link gives the *median* subject, not the population mean. The population mean requires marginalising over the distribution of group-level effects, which is done by predicting for a new level:

```
native_parameters(fit, newdata = transform(nd, id = "new"),
                  allow_new_levels = TRUE, sample_new_levels = "gaussian")
```

These three quantities differ substantially at between-subject standard deviations typical for working memory data.

What is returned

Parameters fixed to a constant are returned at that constant, transformed to the native scale. Because `bmm` fixes parameters on the *link* scale, the native value can differ from the value shown by `parameter_info()`: the `ddm` relative starting point `zr` is fixed at `0` under a `logit` link and is therefore reported as `0.5`, and the `ezdm` and `cswald` diffusion constant `s` is fixed at `0` under a `log` link and is reported as `1`.

Trial-level derived quantities, such as the response probabilities of the mixture models or the category probabilities of `m3`, are not returned; use `brms::posterior_epred()` for those.

The default grid contains the observed combinations of the predictors, not their full crossing, so cells that were never presented do not appear. A continuous predictor therefore produces one grid cell per observed value; supply `newdata` for such models.

Circular location parameters

The circular models sample their location parameter (`mu` for `sdm`, `mu1` for `mixture2p`, `mixture3p` and `imm`) through a `tan_half` link, and it is returned in radians in `(-pi, pi)` — a response bias relative to the target, since the response variable is the angular error. It is fixed to `0` unless the `bmmformula` predicts it explicitly, so an all-zero `mu1` means the model never estimated one.

The inverse link is `2 * atan()`, and the caveat above applies to it with particular force: `2 * atan(b_mu1_conditionB)` is neither the bias in condition B nor the difference between conditions. The bias in condition B is `2 * atan(b_mu1_Intercept + b_mu1_conditionB)`, and the difference between the conditions is that value minus `2 * atan(b_mu1_Intercept)`, taken draw by draw.

Mixture weights

For `mixture3p` the returned `thetat` and `thetant` are probabilities but they **do not sum to 1**: `brms` holds the linear predictor of one mixture component at zero, and that component is the guessing distribution, so the remaining mass `1 - thetat - thetant` is the probability of a guess. `thetant` is the *total* probability of a non-target response, summed over the lures, not the probability per lure.

At set sizes where no non-target was presented the model switches its non-target components off, so `thetant` is reported as exactly `0` and `thetat` as `plogis()` of its linear predictor, rather than as the value the set-size regression extrapolates to. This correction reads the lure indicators out of the prediction grid; if you supply `newdata`, columns you do not supply are filled from the first row of the model data rather than recomputed, and `native_parameters()` warns when the two disagree.

See Also

`parameter_info()`, `native_transform()`, `conditional_effects.bmmfit()`

Examples

```
fit <- bmm(
  bmf(c ~ 0 + set_size, kappa ~ 1),
  data = oberauer_lin_2017,
  model = sdm(resp_error = "dev_rad")
)

# draws of c and kappa for every set size, on the native scale
np <- native_parameters(fit, re_formula = NA)
head(np)

# a contrast is plain arithmetic on the draws
c_draws <- subset(np, parameter == "c")
quantile(
  c_draws$value[c_draws$set_size == 1] - c_draws$value[c_draws$set_size == 4],
  probs = c(0.025, 0.5, 0.975)
)

# posterior summaries of the transformed draws
native_parameters(fit, re_formula = NA, summary = TRUE)
```

`native_transform` *Transform parameter draws from the sampling scale to the native scale*

Description

Applies a `bmmodel`'s inverse link functions to posterior draws of its parameters. This is the extension point used by `native_parameters()`: the transformation is a property of the *model*, not of an individual parameter, because some models map several parameters jointly (e.g. mixture weights through a softmax).

A model whose parameters transform elementwise, or through one softmax group that is active on every row of the data, needs no method — declaring `links` in its `.model_*()` constructor is sufficient, and that covers every model bmm currently ships. Write a method when either condition fails. Two cases that are known to fail it: a model with several independent multinomial branches, because a `"softmax"` link carries no group identity and all such parameters are softmaxed together as one group; and a transformation that depends on the design rather than only on the parameter's own value.

Usage

```
native_transform(model, linpred, data, ...)

## Default S3 method:
native_transform(model, linpred, data, ...)

## S3 method for class 'non_targets'
native_transform(model, linpred, data, ...)
```

Arguments

`model`	A bmmodel object.
`linpred`	A named list of matrices of linear predictor draws, one per model parameter, each with draws in rows and prediction grid cells in columns. These are `brms::posterior_linpred()` output, so they are on the *link* scale for distributional parameters and are the raw value for non-linear parameters. For bmm's `nlpar`-based models the link is a fiction maintained by the model's own `nlf()` expression — `imm` declares `c = "log"` because `configure_model.imm_abc()` writes `exp(c)` into the formula, not because brms applies a link — so a model that declares a link it does not actually apply in its `nlf()` will get silently wrong output here.
`data`	The prediction grid the draws were computed on, with one row per column of the `linpred` matrices. These are rows of the fitted model's own data, so they carry the bmm-internal columns a design-dependent transformation keys on (`LureIdx*`, `inv_ss`, `Idx_*`, matrix columns). `native_transform.non_targets()` is the worked example.
`...`	Currently unused.

Details

Methods for specific models follow the bmm S3 chain, which runs from the general classes to the specific ones. A method should therefore transform the parameters it owns, remove them from `linpred`, and call `NextMethod()` so the remaining parameters reach the default method:

```
native_transform.mymodel <- function(model, linpred, data, ...) {
  own <- .my_joint_transform(linpred[c("p1", "p2")])
  linpred <- linpred[not_in(names(linpred), c("p1", "p2"))]
  c(NextMethod(), own)
}
```

Removing them before `NextMethod()` is what keeps a method idempotent with respect to the parameters it does not own: a parameter left in `linpred` is transformed a second time by the next method in the chain, which does not error and does not warn.

Value

A named list of matrices on the native scale. A method **must** return one element for every name it was given and preserve each matrix's dimensions; `native_parameters()` checks both and errors, naming the model, if either is broken. The order of the list is free.

See Also

`native_parameters()`

---

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

parameter_info *Get parameter information for a bmm model*

Description

Returns a data frame with information about the model parameters: their descriptions, whether they are fixed (and to which value), and their link functions. Use `prior_info()` for the priors of a fitted model.

Usage

```
parameter_info(x, ...)

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

## S3 method for class 'bmmfit'
parameter_info(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. Only activation symbols that have their own formula are listed; without data, any other symbol is read as a data column. 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
parameter_info(sdm(resp_error = "y"))

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

---

plot.bmm_sdt_auc *Plot the posterior AUC distribution from a fitted SDT model*

Description

Plot the posterior AUC distribution from a fitted SDT model

Usage

```
## S3 method for class 'bmm_sdt_auc'
plot(x, condition_col = NULL, ...)
```

Arguments

x	A "bmm_sdt_auc" object from auc_sdt().
condition_col	Optional character. Condition column for colour. If NULL (default), auto-detected.
...	Ignored.

Value

A ggplot2 object.

See Also

auc_sdt(), roc_sdt()

---

plot.bmm_sdt_latent *Plot the latent decision-variable distributions of an SDT model*

Description

Draws the model-implied noise and signal evidence densities on the latent decision axis. For sdt_yn()/sdt_rating() the response criterion or the K-1 confidence thresholds are added as dashed vertical lines with a shaded credible band; when several boundaries are shown together (e.g. base-rate criteria or the rating thresholds) they are colour-coded. sdt_mafc() and sdt_ranking() have no boundary; with show_competitors = TRUE in latent_sdt() the max-of-distractors densities are overlaid as dashed lines, one per set size. Distinct d/sdratio conditions are faceted.

Usage

```
## S3 method for class 'bmm_sdt_latent'
plot(x, condition_col = NULL, line_alpha = 0.9, ...)
```

Arguments

x	A "bmm_sdt_latent" object from latent_sdt().
condition_col	Optional character. Condition column(s) for faceting. If NULL (default), all condition columns are used.
line_alpha	Numeric. Opacity of the criterion/threshold lines (default 0.9).
...	Ignored.

Value

A ggplot2 object.

See Also

latent_sdt(), roc_sdt()

plot.bmm_sdt_roc *Plot a model-implied SDT ROC curve*

Description

Shows the smooth model-implied ROC as a credible-band ribbon, with the model-implied operating points overlaid as colour-coded markers with crosshair error bars (uncertainty in both directions). For **binary** multi-criteria fits the points are the several criterion levels; for **rating** models they are the K-1 confidence thresholds (labelled `t1`..`t(K-1)`). Pass `observed` to overlay empirical points from `roc_observed()`.

Usage

```
## S3 method for class 'bmm_sdt_roc'
plot(
  x,
  observed = NULL,
  condition_col = NULL,
  add_diagonal = TRUE,
  scale = c("probability", "quantile", "z"),
  ribbon_alpha = 0.25,
  point_size = 2.5,
  ...
)
```

Arguments

`x`	A `"bmm_sdt_roc"` object from `roc_sdt()`.
`observed`	Optional `"bmm_sdt_roc_observed"` object from `roc_observed()` to overlay as empirical points.
`condition_col`	Optional character. Condition column for colour/faceting. If `NULL` (default), auto-detected.
`add_diagonal`	Logical. Draw the chance-level diagonal (default `TRUE`).
`scale`	Either `"probability"` (default) for the usual hit vs. false- alarm-rate axes, or `"quantile"` (alias `"z"`) for the quantile-transformed axes (the inverse CDF of the fitted noise distribution).
`ribbon_alpha`	Numeric. Transparency of the credible band (default `0.25`).
`point_size`	Numeric. Size of operating-point markers (default `2.5`).
`...`	Ignored.

Details

With `scale = "quantile"` (or its alias `scale = "z"`) the rates are read on the distribution's quantile axis via the transform `-qf(1 - rate)`, where `qf` is the inverse CDF of the fitted noise distribution, and the axis label names that transform (`z` for normal, `logit` for logistic, `loglog`/`cloglog`

for the Gumbel distributions). On this axis the binary model ROC is a straight line with slope `1 / exp(sdratio)` and intercept `d * sqrt((1 + exp(sdratio)^2) / 2) / exp(sdratio)` – the separation in noise-SD units over the signal SD, since `d` is d_a – so a slope below 1 is the unequal-variance signature (signal SD > noise SD), and departures of the observed points from a straight line diagnose misfit. This linearity holds for all four distributions. The (0,0) and (1,1) endpoints map to infinity and are dropped on the transformed scale.

Value

A `ggplot2` object. The credible band (ribbon) is the pointwise posterior interval of the hit rate at each criterion value, plotted at that criterion's posterior-mean false-alarm rate; it does not include uncertainty in the false-alarm rate.

See Also

`roc_sdt()`, `roc_observed()`, `auc_sdt()`

---

plot.mpt_tree *Plot the structure of MPT trees*

Description

Draws the processing-tree diagram implied by the branch probability expressions: each branch expression is expanded into its root-to-leaf paths (a sum of products, with the factors in writing order), paths with a common prefix share the corresponding tree nodes, and the leaves show the response categories. Calling `plot()` on a model created with `mpt()` draws all trees of the model.

Usage

```
## S3 method for class 'mpt_tree'
plot(x, cex = 0.9, ...)

## S3 method for class 'mpt'
plot(x, cex = 0.9, ...)
```

Arguments

`x`	An `mpt_tree` object or a `bmmodel` object created with `mpt()`.
`cex`	Character expansion factor for the edge and leaf labels.
`...`	Ignored.

Details

Edges are drawn in the order the branch expressions first mention them, bare constants included. A complement edge `1 - X` is placed directly below its sibling `X`; the complement of a product, `1 - PG1 * PG2`, is one atomic edge. A numeric constant factor above 1 is not a separate edge: the constant and all factors after it form one edge, as in Batchelder & Riefer (1999, Fig. 1). For example, `B * 2 * u * (1 - u)` draws one edge `B`, then one edge `2 * u * (1 - u)`. If the constant is at the start (`2 * (1 - c) * u * (1 - u)`), the entire expression is one root edge beside its siblings, drawn without a warning, so write the constant after the process steps it follows, as in `(1 - c) * 2 * u * (1 - u)`. If it is in the middle (`u * 2 * (1 - u)`), the constant and the factors after it join into one edge, and a trailing constant joins the factor before it. The plot warns when this merging hides a process step: a factor after the constant that has no complement among the merged factors (`D * 2 * g * (1 - g) * r`), a factor separated from its complement by the constant (`u * 2 * (1 - u)`), or a quotient whose denominator does not contain the complemented parameter (`(1 - Pi) * 1 / 15` is one edge; write `(1 - Pi) * (1/15)`). A quotient normalised by a sum that contains the parameter, as in `x * (1 - d) / (x * (1 - d) + y)`, does not warn. Expressions are expanded distributively, so `Pm * (Pb + (1 - Pb) * 0.25)` displays two paths sharing the `Pm` edge.

The plot warns when edges leaving an internal node do not sum to 1, but only for nodes whose edge labels are all constants, plain parameters, or `1 - parameter`; product edges, quotient edges, function calls, and covariate fans that lack complements are not checked, nor are nodes with a single edge or, when plotting a bare tree, fans of plain parameters (which a model may declare as a simplex). Write factors in the same order across all categories to avoid mispairing due to different factor orders. A bare `mpt_tree()` whose constants were already substituted (such as the stored restricted trees of a model) does not recognise a complement written as a number (`0.8` beside `0.2`), so those edges keep the order of first mention; `plot()` of a model draws the unrestricted trees and is not affected.

For a model created with `mpt()`, the plot draws the trees as written and marks restricted parameters: the edge of a fixed parameter reads `g = 0.5`, the edge of an equated parameter `Dn (= Do)`, and an edge that contains a restricted parameter shows the restriction in parentheses, as in `1 - g (g = 0.5)`. Impossible categories are omitted. A literal 0 in an expression (as in `D + 0 * x`) is drawn as an edge labelled 0. The `...` argument is ignored; the title cannot be changed. Every product of sums is expanded into all its paths, so each multiplied sum such as `(a + b)` doubles the number of paths: trees with more than about ten multiplied sums draw slowly. Leaf labels wider than about 90% of a panel are clipped, and long edge labels near the root can be clipped earlier, when the leaf labels leave little room.

Value

The object `x`, invisibly.

References

Batchelder, W. H., & Riefer, D. M. (1999). Theoretical and empirical review of multinomial process tree modeling. Psychonomic Bulletin & Review, 6(1), 57-86. https://doi.org/10.3758/BF03210812

Examples

```
tree_target <- mpt_tree("target", list(
  old = "D + (1 - D) * g",
  new = "(1 - D) * (1 - g)"
```

```
))
plot(tree_target)

model <- mpt(
  list(
    tree_target,
    mpt_tree("lure", list(
      old = "(1 - D) * g",
      new = "D + (1 - D) * (1 - g)"
    ))
  ),
  tree_id = "item_type"
)
plot(model)
```

`posterior_epred.bmmfit` *Expected response of a fitted bmm model*

Description

`posterior_epred()` returns draws of the expected value of the response for each observation: the mean of what `brms::posterior_predict()` simulates for it. What that is depends on the model:

Model	Expected response
`ddm()`, `cswald()`	Mean response time, averaged over both responses, under the diffusion p
`ezdm()`, version `"3par"`	Mean response time of the cell
`ezdm()`, version `"4par"`	Mean response time of the cell's upper-boundary responses
`sdt_yn()`	Number of "old"/"signal" responses, the number of trials times their pro
`sdt_mafc()`	Number of correct responses, the number of trials times their probability
`m3()`, `sdt_rating()`, `sdt_ranking()`, `sdt_cdp()`	Expected count of each response category, from the multinomial family
`sdm()`, `mixture2p()`, `mixture3p()`, `imm()`	Not defined: the mean of a circular response error is not a useful quantit

With `dpar` or `nlpar`, `posterior_epred()` returns draws of that model parameter for every model, as in **brms**: a `dpar` on its native scale, an `nlpar` on the scale of its link. `native_parameters()` returns the model parameters on their native scale over a grid of predictor values.

Usage

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

## S3 method for class 'bmmfit'
fitted(object, ..., scale = c("response", "linear"), dpar = NULL, nlpar = NULL)
```

Arguments

object	A bmmfit object.
...	Further arguments passed to brms::posterior_epred(), such as newdata or ndraws.
dpar, nlpar	Name of a distributional or non-linear parameter whose draws are returned instead of the expected response.
scale	As in brms::fitted.brmsfit(): "response" is the expected response, "linear" the linear predictor of mu.

Value

A draws by observations matrix (an array with a third dimension for the response categories of the multinomial models).

See Also

brms::posterior_epred(), native_parameters()

Examples

```
## Not run:
fit <- bmm(
  bmf(drift ~ 1, bound ~ 1, ndt ~ 1),
  data = rddm(200, drift = 1.5, bound = 1.2, ndt = 0.3),
  model = ddm(rt = "rt", response = "response"),
  backend = "cmdstanr"
)
# expected response time of each observation, one row per draw
epred <- posterior_epred(fit)

## End(Not run)
```

---

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 = NULL,
  ndraws = NULL,
  group = NULL,
  resp_var = NULL,
  ...
)
```

Arguments

object	A bmmfit object returned by bmm().
type	Character. Type of pp_check. When NULL (default), resolves to "dens_overlay", or to the selected observable's default type when resp_var is specified. 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. With resp_var, type = "bars_binned" is also available: it bins a continuous statistic like a histogram, with bars for the observed number of observations per bin and points with intervals for the predicted number. It is the default for the ezdm() accuracy check.
ndraws	Integer. Number of posterior draws. Defaults to 100 for multinomial models and 10 when resp_var is specified; 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. For the sdt_rating() model group defaults to the stimulus variable so the signal and noise rating distributions are checked separately; pass group = NA to pool them into a single profile. The sdt_ranking() model shows the rank positions as the categories of the proportion profile; for mixed set sizes, pass the set-size column as group so each facet has a homogeneous denominator. For mpt() fits, the tree_id column can be named even if no parameter formula uses it, giving one facet per tree; the default pools the trees. The custom-family models sdt_yn() and sdt_mafc() delegate to brms::pp_check(); for their aggregated counts an overlaid empirical CDF (type = "ecdf_overlay", adding group = "stimulus" for sdt_yn()) is usually clearer than the default bar view.
resp_var	Character. For models that declare several observables, the name of the observable to check, or "all" for a panel of all available checks built from one shared simulation. See pp_check_vars() for the options of a fitted model. The default NULL checks the primary response via brms::pp_check(), except for an ezdm(version = "4par") fit in which some cells have no usable summaries at the upper boundary: the primary response, mean_rt_upper, holds placeholders there, so NULL means resp_var = "mean_rt_upper", which leaves those cells out but takes neither newdata nor the loo_* types. For the RT models, passing negative_rt = TRUE (a brms::posterior_predict() argument) is redirected

to `resp_var = "signed_rt"`, so that observed and predicted response times are both signed by the response.

`...` Additional arguments. Without `resp_var`, forwarded to `brms::pp_check()`, or for multinomial models to `brms::posterior_predict()` (`probs`, a numeric vector of length 2 with default `c(0.025, 0.975)`, sets the credible interval). With `resp_var`, `draw_ids` and `re_formula` go to `brms::prepare_predictions()` and the rest to the `bayesplot::ppc_*` function. `type = "bars_binned"` takes `breaks` (bin edges that cover the observed and predicted values), `prob` (interval width, default `0.9`) and `freq` (`FALSE` for proportions instead of counts). `re_formula = NA` predicts at the population level on every path.

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.

Some models describe several observables jointly (e.g. `ddm()`: response times *and* responses; `ezdm()`: mean RT, RT variance *and* accuracy), but brms only ever checks the primary response. For these models the `resp_var` argument selects which observable to check; `pp_check_vars()` lists the available checks. All selected observables are drawn from **one** joint posterior predictive simulation, so `resp_var = "all"` panels are mutually consistent.

Some observables are undefined for some cells — an `ezdm()` boundary has no mean response time when fewer than two responses reach it. Observations whose *observed* value is undefined are dropped from the check; undefined values in the *simulated* replicates are absorbed by dropping those posterior draws instead, so the number of observations checked does not depend on `ndraws`. Dropped draws are not missing at random — they are draws whose parameters made a boundary sparse — so the retained predictive is mildly conditioned; both reductions are reported with a warning. With `resp_var = "all"` one reduction is shared by every panel, so the panels are computed on the same observations and draws.

Value

For multinomial models, for a 4-parameter `ezdm()` fit with placeholders in `mean_rt_upper`, or when `resp_var` is specified, a `ggplot2` object (a `bayesplot_grid` for `resp_var = "all"`). For other models, the result of `brms::pp_check()`.

See Also

`brms::pp_check()`, `pp_check_vars()`

`pp_check_vars` *List the posterior predictive checks available for a model fit*

Description

Lists the values `resp_var` accepts in `pp_check.bmmfit()` for a fitted model whose likelihood involves several observables.

Usage

```
pp_check_vars(fit)
```

Arguments

fit	A bmmfit object returned by bmm().

Value

A data.frame with one row per available check (columns resp_var, label, default_type, slot, listing the brms standata slots the check reads, and default, flagging the observable that pp_check() plots when resp_var is not specified), or NULL invisibly for models without multi-observable support.

See Also

pp_check.bmmfit()

Examples

```
## Not run:
fit <- bmm(bmmformula(drift ~ condition), data, ddm(rt = "rt", response = "response"))
pp_check_vars(fit)
pp_check(fit, resp_var = "response")

## End(Not run)
```

---

ppp *Posterior predictive p-values for count models*

Description

Computes the goodness-of-fit statistics T_1 and T_2 of Klauer (2010) and their posterior predictive p-values (PPP) for models that fit aggregated response counts: mpt(), m3(), sdt_yn(), sdt_rating(), sdt_mafc(), sdt_ranking() and sdt_cdp(). TreeBUGS' PPP() reports the same two statistics; Details says where their values differ.

Usage

```
ppp(fit, ...)

## S3 method for class 'bmmfit'
ppp(
  fit,
  statistic = c("T1", "T2"),
  by = NULL,
  ndraws = NULL,
  level = c("group", "participant"),
```

```
  participant = NULL,
  ...
)
```

Arguments

`fit`	A `bmmfit` object returned by `bmm()`.
`...`	Unused.
`statistic`	Character. `"T1"`, `"T2"`, or both (the default). If the participants cannot be identified, the default computes `"T1"` only.
`by`	Character vector naming the columns of the model data that define the cells. `NULL` (default) uses the cells described in Details; `character(0)` pools all rows into one cell. For `mpt()` fits, the tree column can be named even if it is not a predictor.
`ndraws`	Integer. Number of posterior draws to use. `NULL` (default) uses all draws.
`level`	Character. `"group"` (default) for the statistics of the whole data set; `"participant"` for one T_1 and PPP per participant, as TreeBUGS reports them, to find the participants the model fits poorly.
`participant`	Character. The column that identifies participants, needed for T_2 and for `level = "participant"`. It must be a variable of the model formula, because the model data contain only those; a model without random effects has one only if a predictor such as `0 + id` names the participants. `NULL` (default) uses the grouping variable of the random effects; when the model has more than one (e.g. participants and items), name it here.

Details

Both statistics are computed once per posterior draw, for the observed counts and for counts simulated from that draw with `brms::posterior_predict()`, each against the expected counts of the same draw from `brms::posterior_epred()`. The simulated data are new data from the same participants. The PPP is the proportion of draws in which the simulated counts deviate more from the expectation than the observed counts do. A PPP near 0 flags misfit; a PPP away from 0 means that the model reproduces the aspect of the data the statistic measures.

T_1**: mean structure.** The rows of the data are summed into cells, and T_1 is Pearson's X^2 over all cells and response categories,

$$T_1 = \sum_{c,k} \frac{(n_{ck} - \hat{n}_{ck})^2}{\hat{n}_{ck}},$$

where n_{ck} is the observed and $\hat{n}_{ck}$ the expected count of category k in cell c. Klauer (2010, Eq. 17) and TreeBUGS average the counts over participants instead of summing them. With the same number T of participants in every cell, the two give the same PPP and T_1 here is T times theirs; otherwise the mean gives the cells with fewer participants more weight.

T_2**: covariance structure.** For every participant, the counts are summed per cell, giving one vector per participant over all cells and categories. T_2 compares the covariance matrix s of these vectors across participants with the covariance matrix σ the model expects (Klauer, 2010, Eq. 18):

$$T_2 = \sum_{i,j} \frac{(s_{ij} - \sigma_{ij})^2}{\sqrt{\sigma_{ii}\sigma_{jj}}}.$$

σ is the covariance of the expected counts across participants plus the average multinomial covariance within a participant, which is the expected value of s given the draw. Both covariances divide by $T - 1$, where T is the number of participants. Klauer (2010) multiplies the multinomial part by $(T - 1)/T$, the expected value for covariances that divide by T; that version of T_2 is $(T - 1)/T$ times this one and has the same PPP. TreeBUGS combines the factor with covariances that divide by $T - 1$, so its σ falls short of the expected s by $1/T$ of the multinomial part. The sum runs over the full matrix, so the covariances between trees or conditions, which carry the correlations between the person parameters, are part of the check. When participants differ in the cells they have, as in a between-participants design, they are grouped by their set of cells, T_2 is computed per group, and the results are summed. A participant who shares the set of cells with nobody else has no covariance and is left out, with a message.

Cells. By default, a cell is a combination of the tree (`mpt()`), the stimulus (the SDT models) or the set size (`sdt_ranking()`), and the categorical population-level predictors of the model formula. Numeric predictors are not part of the default, because every distinct value would form a cell of its own. Rows of a cell are summed, so the trees or stimuli must be part of by for the statistic to compare each tree's own categories.

Impossible categories. Categories with an expected count of about 0 in every draw, such as the impossible categories of an `mpt()` or `m3()` model or the ranks beyond the set size of an `sdt_ranking()` model, are left out of both statistics. A cell with a single possible category, such as an `m3()` set size without other candidates, is left out of T_2: the number of trials fixes its count.

Value

A data frame of class bmm_ppp with one row per statistic (and per participant for `level = "participant"`) and the columns `statistic`, `observed` and `replicated` (the means of the statistic over draws, for the observed and the simulated counts, as reported by Klauer, 2010, and TreeBUGS), ppp and `ndraws`. The draws of the statistics are stored in the attribute `"draws"`.

References

Klauer, K. C. (2010). Hierarchical multinomial processing tree models: A latent-trait approach. *Psychometrika, 75*(1), 70–98. https://doi.org/10.1007/s11336-009-9141-0

See Also

`pp_check.bmmfit()` for the graphical check.

Examples

```
trees <- list(
  mpt_tree("target", list(old = "D + (1 - D) * g", new = "(1 - D) * (1 - g)")),
  mpt_tree("lure", list(old = "(1 - D) * g", new = "D + (1 - D) * (1 - g)"))
)
dat <- expand.grid(id = factor(1:20), item_type = c("target", "lure"),
                   stringsAsFactors = FALSE)
dat$old <- rbinom(nrow(dat), 50, ifelse(dat$item_type == "target", 0.8, 0.2))
dat$new <- 50 - dat$old

fit <- bmm(
  bmf(D ~ 1 + (1 | id), g ~ 1 + (1 | id)), dat, mpt(trees, "item_type"),
```

```
  cores = 4, backend = "cmdstanr"
)
ppp(fit)
ppp(fit, level = "participant")
```

`prior_info` *Priors of a fitted bmm model and where they came from*

Description

Lists the priors a fitted bmm model was actually sampled with and where each one came from: a bmm default, a brms default, or a user-specified prior. Each row also gives the link function of the parameter, because the prior applies on that sampling scale. Parameters without a proper prior are flagged as flat.

Usage

```
prior_info(fit, format = "table")
```

Arguments

`fit`	A `bmmfit` object returned by `bmm()`
`format`	Character. `"table"` (default) prints the report as a table; `"text"` prints sentences ready for a methods section.

Details

All priors and constants apply on the sampling scale set by each parameter's link function: a prior for a log-link parameter describes the log of that parameter, and a constant fixes the parameter on that scale (e.g. `constant(0)` with a log link fixes the parameter to 1 on the native scale). Constants are therefore printed together with their exact native-scale value whenever the two differ.

The provenance of each prior is determined by re-deriving the default priors from the model, formula and data stored in the fit. A user-specified prior that is identical to the bmm default is therefore reported as a default. Coefficients that inherit their prior from a more general class are collapsed into the row of the prior they inherit from.

The one exception is a parameter fixed to a constant in the formula: bmm folds such a constant into the model object at fit time, overwriting the default it replaced, so the original default is no longer recoverable from the fit. Constants named in the formula are therefore always reported as `"user"`, even when the value restates the bmm default.

Re-derivation uses the `default_priors` stored inside the fit, so later changes to a model's default *values* do not affect the report of an older fit. It does use the currently installed code that turns those values into prior rows, so if that construction changes between versions, the reported provenance of a fit made with an earlier version may differ from what was actually applied. No check against `fit$version$bmm` is performed.

Flat priors are flagged because they are improper: Bayes factors via bridge sampling are undefined when any parameter has an improper prior.

Parameters that exist only because the family machinery requires them are omitted: a fixed parameter the model does not declare (the `mu` brms forces on the custom families of the response-time models, or the `mu2` and `kappa2` of a two-component `brms::mixture()` family), and the mixture-weight reference component `theta2`, which brms lists with a default prior although the sampler holds it at zero. Fixed parameters the model does declare are reported, since the user can estimate them: `mu` for `sdm()`, `mu1` for the circular mixture models, `zr` for `ddm()`.

Value

A `data.frame` of class `bmm_prior_info` with columns `parameter`, `link`, `class`, `coef`, `group`, `prior` and `source`. Subsetting the report with `[` returns a plain `data.frame`, as the report-specific printing depends on columns and attributes that subsetting drops.

See Also

`default_prior()`, `parameter_info()`

Examples

```
fit <- bmm(
  bmf(c ~ 0 + set_size, kappa ~ 1),
  data = oberauer_lin_2017,
  model = sdm(resp_error = "dev_rad")
)
prior_info(fit)
prior_info(fit, format = "text")
```

---

rejection_sampling *Rejection Sampling*

Description

Performs rejection sampling to generate samples from a target distribution. Each draw can come from its own target: draw `i` is sampled from `f` evaluated at the `i`-th element of every per-draw argument in `...`, under the envelope `max_f[i]`.

Usage

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

Arguments

n	Integer. The number of samples to generate.
f	Function. The target density divided by the proposal density, up to a constant; with a uniform proposal, the target density itself. Its first argument takes a vector of proposals, generally not of length n; f must be vectorized over it and over the per-draw arguments in ....
max_f	Numeric. A finite upper bound of f, either a single value or one value per draw (length n). A bound below the maximum of f biases the draws without a warning.
proposal_fun	Function. A function that generates samples from the proposal distribution.
...	Additional arguments to be passed to the target density function f. With n > 1, arguments of length n are taken per draw, so draw i uses their i-th elements. Arguments of any other length are passed whole to every call of f; recycle them with rep_len(x, n) to use them per draw. Pass constants whose length may equal n, such as a lookup table, through the closure of f instead of ....

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)

# one location per draw
mu <- rep(c(0, 2), 5000)
samples <- rejection_sampling(
  10000, brms::dvon_mises, max_f = brms::dvon_mises(0, 0, 10), proposal,
  mu = mu, kappa = 10
)
tapply(samples, mu, mean)
```

---

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)
```

---

roc_observed *Observed ROC points from a fitted SDT model*

Description

Computes empirical (observed) ROC points from the response-count data used to fit a sdt_rating() or sdt_yn() model. Rating models pool counts across observations within each stimulus type (and optional condition) to produce cumulative hit/false-alarm rates at each confidence threshold; binary models produce one operating point per criterion level (the empirical counterpart of the model-implied points from roc_sdt()).

Usage

```
roc_observed(fit, conditions = NULL)
```

Arguments

fit	A bmmfit object from a rating or binary SDT model.
conditions	Optional character vector of column names to condition on. For rating models, NULL pools all observations into one ROC. For binary models, NULL auto-detects the criterion-varying predictor(s).

Value

A data frame of class "bmm_sdt_roc_observed" with columns FA, Hit, and any condition columns. Rating models additionally include the (0,0) and (1,1) endpoints. The attribute model_type is "rating" or "binary"; rating results also carry n_ratings.

See Also

roc_sdt(), plot.bmm_sdt_roc()

Examples

```
## Not run:
# Three base-rate conditions shift the criterion, which identifies sdratio
dat <- expand.grid(id = 1:20, stimulus = c(0L, 1L),
                   condition = c("liberal", "neutral", "strict"))
dat$n_trials <- 100L
# simulated as bias, measured from the midpoint between the means (0 = boundary at the midpoint)
criteria <- c(liberal = -0.5, neutral = 0, strict = 0.5)
dat$n_old <- rsdt_yn(nrow(dat), dat$n_trials, dat$stimulus, d = 1.5,
                     bias = criteria[as.character(dat$condition)],
                     sdratio = 1.3)

fit <- bmm(
  formula = bmf(d ~ 1, criterion ~ 0 + condition, sdratio ~ 1),
  data = dat,
  model = sdt_yn(response = "n_old", stimulus = "stimulus",
                 n_trials = "n_trials"),
  cores = 4,
  backend = "cmdstanr"
)

# One point per base-rate condition, from the response counts
obs <- roc_observed(fit)
obs
plot(roc_sdt(fit), observed = obs)

## End(Not run)
```

`roc_sdt` *Model-implied ROC curve from a fitted SDT model*

Description

Computes the receiver operating characteristic (ROC) curve implied by the posterior distribution of a signal detection model fit with `bmm()`. ROC curves require a response criterion, so they are defined for `sdt_yn()` and `sdt_rating()` only (not the criterion-free `sdt_mafc()` and `sdt_ranking()`).

Usage

```
roc_sdt(
  fit,
  conditions = NULL,
  n_points = 100,
  probs = c(0.025, 0.975),
  criterion_points = NULL,
  ...
)
```

Arguments

fit	A bmmfit object returned by bmm() from an SDT model.
conditions	Optional data frame of predictor values at which to evaluate the model. Column names must match predictor variables used in the formula. If NULL (default), unique predictor combinations are derived from the data.
n_points	Integer. Number of equally-spaced points on the smooth model-implied ROC curve (default 100); used for both binary and rating models. The rating data frame itself still holds K+1 points per draw (K-1 threshold points plus the (0,0) and (1,1) endpoints).
probs	Numeric vector of length 2. Lower and upper quantiles for the credible band (default c(0.025, 0.975)).
criterion_points	Optional. Control of the binary multi-criteria behaviour. NULL (default) auto-detects predictors that vary the criterion only (not d/sdratio) and treats their levels as operating points on one curve. Pass a character vector of column names to force that classification, or FALSE to disable it (one separate curve per predictor combination). Ignored for rating models.
...	Additional arguments passed to brms::posterior_linpred(), such as draw_ids to use a subset of the posterior draws. ndraws is refused: each parameter is drawn by its own call, so random subsets would not match across parameters.

Details

For **binary** models the ROC is traced analytically from the posterior of d (and sdratio for unequal-variance SDT) over a grid of criterion values. When the criterion varies across conditions but d/sdratio do not (e.g. a base-rate manipulation, as in broeder_schuetz_2009_e3), the several criteria are operating points on a single curve: roc_sdt() returns one smooth curve and attaches the model-implied points (one per criterion level) as a points attribute.

For **rating** models the K-1 confidence thresholds define K-1 empirical ROC points per posterior draw (returned as the data frame). The smooth model-implied curve is traced over a virtual cut from the posterior of d (and sdratio) and attached as the summary attribute, with the K-1 thresholds attached as the points attribute (labelled t1..t(K-1)) so they fall on the curve. For version = "metad" the smooth curve is the type-1 (d, sdratio) ROC and the threshold points are the model's confidence operating points; they lie inside the curve when the M-ratio is below 1, by the amount of the metacognitive inefficiency.

Value

A data frame of class "bmm_sdt_roc" with columns FA, Hit, .draw, and any condition columns. The object carries a summary attribute (FA, Hit_mean, Hit_lower, Hit_upper) with the smooth model-implied curve, and a points attribute with the model-implied operating points: one per criterion level for binary multi-criteria fits, or the K-1 confidence thresholds (labelled t1..t(K-1)) for rating fits. It also carries the attributes probs, model_class, dist, is_rating and conditions, which the print() and plot() methods read. The Hit_lower and Hit_upper columns in the summary attribute are the pointwise posterior quantiles of the hit rate at each criterion value, plotted at that criterion's posterior-mean false-alarm rate; this band does not include uncertainty in the false-alarm rate and is narrower than a credible band at a fixed false-alarm rate.

See Also

auc_sdt(), roc_observed(), plot.bmm_sdt_roc()

Examples

```
## Not run:
# Three base-rate conditions shift the criterion, which identifies sdratio
dat <- expand.grid(id = 1:20, stimulus = c(0L, 1L),
                   condition = c("liberal", "neutral", "strict"))
dat$n_trials <- 100L
# simulated as bias, measured from the midpoint between the means (0 = boundary at the midpoint)
criteria <- c(liberal = -0.5, neutral = 0, strict = 0.5)
dat$n_old <- rsdt_yn(nrow(dat), dat$n_trials, dat$stimulus, d = 1.5,
                     bias = criteria[as.character(dat$condition)],
                     sdratio = 1.3)

fit <- bmm(
  formula = bmf(d ~ 1, criterion ~ 0 + condition, sdratio ~ 1),
  data = dat,
  model = sdt_yn(response = "n_old", stimulus = "stimulus",
                 n_trials = "n_trials"),
  cores = 4,
  backend = "cmdstanr"
)

roc <- roc_sdt(fit)
roc
plot(roc, observed = roc_observed(fit))
plot(roc, observed = roc_observed(fit), scale = "quantile")

## End(Not run)
```

---

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. https://doi.org/10.1037/rev0000328
- **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: normal(0, 0.5)
    - effects: normal(0, 0.25)
    - sd: exponential(4)
  - kappa:
    - main: student_t(5, 1.75, 0.75)
    - effects: normal(0, 1)
    - sd: exponential(1)
  - c:
    - main: student_t(5, 2, 0.75)
    - effects: normal(0, 1)
    - sd: exponential(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")
```

---

sdt_cdp *Continuous Dual-Process Signal Detection Theory Model*

Description

Continuous Dual-Process Signal Detection Theory Model

Usage

```
sdt_cdp(
  response = "",
  stimulus,
  n_new,
  n_old,
  dist = "normal",
  threshold_type = c("parsimonious", "equidistant", "log_distance"),
  links = NULL,
  ...
)
```

Arguments

response	An optional common prefix for the response count columns (default "", i.e. the bare canonical names new1, know2, ... that aggregate_sdt_cdp_data() and rsdt_cdp() produce).
stimulus	The name of the variable coding the stimulus type. Must be coded as 0 (new/lure) and 1 (old/target).
n_new, n_old	Integer numbers of "new" and "old" confidence levels. Together with response they determine the response count columns and the threshold parameters.
dist	The noise distribution. Only "normal" is currently supported (the CDP model is inherently Gaussian).
threshold_type	Character. Threshold parameterization on the strength axis: "parsimonious" (default) and "equidistant" use a single spacing parameter; "log_distance" (Paulewicz & Blaut, 2022) estimates the n_new + n_old - 2 distances between adjacent thresholds freely, each as a deltaN parameter on the log scale (the log width of the interval between thresholds N and N + 1, as in sdt_rating()). criterion stays on the old/new boundary, threshold n_new.
links	A named list of link functions for the parameters, e.g. links = list(drec = "log"). Only dfam, drec, criterion and rcrit can be set, to "log", "softplus", "logit", "probit" or "identity", the links the model can invert inside its formula. sigmar, rho, kcrit and the threshold parameters keep their identity links, because their fixed values and the model's own transformations (exp(), tanh()) assume it. A positive-range link ("log", "softplus") on criterion or rcrit states that the criterion is positive: the sampler then starts next to 0 on the native scale and cannot reach negative values.
...	used internally for testing, ignore it

Details

- **Domain:** Recognition Memory
- **Task:** Old/New Recognition with Remember/Know Judgments
- **Name:** Continuous Dual-Process Signal Detection Theory (CDP)
- **Citation:**
  - Wixted, J. T., & Mickes, L. (2010). A continuous dual-process model of remember/know judgments. Psychological Review, 117(4), 1025-1054. https://doi.org/10.1037/a0020874
- **Requirements:**

  Provide aggregated data with one row per cell (unique combination of predictors and stimulus class) and one integer count column per response category:
  - new1 ... new<n_new> for 'new' judgments
  - know<k> and remember<k> (and optionally guess<k>) for 'old' judgments, with k on the unified confidence scale n_new+1 ... n_new+n_old
  - an optional common column prefix is set via response
  - stimulus: 0 = new/lure, 1 = old/target

Use aggregate_sdt_cdp_data() to build these columns from long-format (trial-level) data

- **Parameters:**
  - `dfam`: Familiarity sensitivity: target mean on the familiarity axis, in lure-SD units (d' on that axis, where targets and lures share SD 1)
  - `drec`: Recollection sensitivity: target mean on the recollection axis, in lure-SD units. The target SD is exp(sigmar), so once sigmar is estimated drec is not the d_a the other SDT models report as d
  - `criterion`: Response bias: old/new boundary on the strength (F+R) axis
  - `spacing`: Threshold spacing: controls distance between adjacent thresholds (exp(spacing) ensures positive spacing)
  - `rcrit`: Remember criterion: threshold on the recollection axis
  - `sigmar`: Log SD of the recollection target distribution, so exp(sigmar) is the SD and 0 means SD = 1
  - `rho`: Familiarity-recollection correlation on an unconstrained scale; tanh(rho) is the correlation and 0 means independent processes
  - `kcrit`: Know criterion: threshold on the familiarity axis that splits Know from Guess. Active only when the data include 'guess' counts
- **Fixed parameters:**
  - `sigmar` = 0
  - `rho` = 0
  - `kcrit` = -100
- **Default parameter links:**
  - dfam = identity; drec = identity; criterion = identity; spacing = identity; rcrit = identity; sigmar = identity; rho = identity; kcrit = identity
- **Default priors:**
  - `dfam`:
    - `main`: normal(1, 1)
    - `effects`: normal(0, 0.5)
    - `sd`: exponential(1)
  - `drec`:
    - `main`: normal(1, 1)
    - `effects`: normal(0, 0.5)
    - `sd`: exponential(1)
  - `criterion`:
    - `main`: normal(0, 1.5)
    - `effects`: normal(0, 0.5)
    - `sd`: exponential(2)
  - `spacing`:
    - `main`: normal(0, 0.5)
    - `effects`: normal(0, 0.3)
    - `sd`: exponential(2)
  - `rcrit`:
    - `main`: normal(0, 1)

    * effects: normal(0, 0.5)
    * sd: exponential(2)
  - sigmar:
    * main: normal(0, 0.5)
    * effects: normal(0, 0.3)
    * sd: exponential(2)
  - rho:
    * main: normal(0, 0.5)
    * effects: normal(0, 0.3)
    * sd: exponential(2)
  - kcrit:
    * main: normal(0, 1)
    * effects: normal(0, 0.5)
    * sd: exponential(2)

The continuous dual-process model (Wixted & Mickes, 2010) assumes two continuous memory signals per item, Familiarity (F) and Recollection (R), with correlation tanh(rho). Old/new confidence is read off the aggregate strength S = F + R; the Remember/Know judgment splits "old" responses on R (against rcrit); an optional Know/Guess split uses F (against kcrit).

Response format. The model is fit to aggregated counts, like sdt_rating(): one row per cell (e.g. participant x stimulus class x condition) and one integer count column per response category. The columns follow a fixed naming scheme – new1 ... new<n_new> for "new" judgments, and know<k> / remember<k> (optionally guess<k>) for "old" judgments, where k runs over the unified confidence scale n_new + 1 ... n_new + n_old (1 = most confident "new", K = most confident "old") – so the constructor only needs the numbers of confidence levels, not a long vector of column names. An optional common prefix is set via response (e.g. response = "cdp_" for columns cdp_new1, cdp_know2, ...). Use aggregate_sdt_cdp_data() to build these columns from long-format (trial-level or count) data; rsdt_cdp() generates them directly. The numbers of "new" and "old" confidence levels need not be equal (e.g. the 1-new / 5-old scale of Rotello et al., 2005). Whether the Know/Guess split is modelled is **driven by the data**: include guess<k> columns to fit the three-way Remember/Know/Guess model.

Variants via the formula. By default sigmar, rho, and kcrit are fixed (equal recollection variance, independent F and R, no Know/Guess split), recovering the classic independent-process CDP. Free them through the formula:

- sigmar ~ 1 estimates unequal recollection variance.
- rho ~ 1 estimates the within-item F-R correlation; rho also accepts predictors (e.g. rho ~ condition), bounded to (-1, 1) via an internal tanh. This structural, within-item correlation is distinct from a between-subject correlation of random effects, which is available separately through brms syntax such as (1 |p| id) on any parameter.
- kcrit ~ 1 estimates the Know/Guess criterion. The data must then contain the guess<k> count columns, and data with those columns must free kcrit: with kcrit at its fixed value every Guess response has probability 0, so bmm() refuses that combination. The fixed value -100 is a sentinel that switches the split off; dsdt_cdp() and rsdt_cdp() use kcrit = NULL for the same purpose.

In the three-way model a Guess response in confidence bin k is possible only while `rcrit + kcrit` exceeds the lower threshold of that bin, because Guess needs R below `rcrit` and F below `kcrit`, so S = F + R below their sum. Below that boundary the Guess category has probability 0, and when `rcrit` (or `kcrit`) varies across participants the posterior explores it: hierarchical Remember/Know/Guess fits can report divergent transitions near this boundary without biased estimates. If they do, raise `adapt_delta` (0.95 or 0.99), give `rcrit` a tighter group-level prior, or check which participants' Guess counts in the top bins drive it. The Remember/Know model has no such boundary.

The kernel's gradient is wrong, with the value right, on two measure-zero sets that `init = 0` starts on (a threshold exactly on the strength mean, and `rcrit == kcrit`); bmm's own initial values avoid them, and a fit started on them walks off in its first step.

Value

An object of class `bmmodel`

Sensitivity scales

`dfam` and `drec` are the target means μ_F and μ_R of Wixted and Mickes (2010), each in units of the lure SD on its own axis, so estimates can be compared with the values they report. On the familiarity axis targets and lures both have SD 1, and `dfam` is d' there. On the recollection axis the target SD is `exp(sigmar)`, so `drec` is standardized by the lure SD alone. With `sigmar` fixed at 0 it is d' as well; with `sigmar` estimated it is not the balanced index d_a that `sdt_yn()` and `sdt_rating()` report as `d`.

This matters when `drec` is compared across conditions whose `sigmar` differs: a difference in the recollection SD alone can then appear as a credible difference in `drec`, the problem the sensitivity section of `sdt_yn()` describes for d'. To compare such conditions, compute the balanced recollection sensitivity from the posterior draws as `drec / sqrt((1 + exp(2 * sigmar)) / 2)`, or give `sigmar` the same formula in every condition being compared.

Neither `dfam` nor `drec` is the discriminability of the old/new decision, which is read off the aggregate strength $F + R$ and depends on both sensitivities, on `sigmar` and on `rho`.

When no Remember/Know split is available (confidence ratings only), use `sdt_rating()` instead.

References

Wixted, J. T., & Mickes, L. (2010). A continuous dual-process model of remember/know judgments. *Psychological Review*, *117*(4), 1025–1054. doi:10.1037/a0020874

Paulewicz, B., & Blaut, A. (2022). The general causal cumulative model of ordinal response. *PsyArXiv*. doi:10.31234/osf.io/e7a3x

Rotello, C. M., Macmillan, N. A., Reeder, J. A., & Wong, M. (2005). The remember response: Subject to bias, graded, and not a process-pure indicator of recollection. *Psychonomic Bulletin & Review*, *12*(5), 865–873. doi:10.3758/BF03196778

Examples

```
## Not run:
# Simulate a Remember/Know data set (3 new + 3 old confidence levels) for
# 20 subjects: rsdt_cdp() returns the count columns sdt_cdp() expects
```

```
dat <- expand.grid(id = 1:20, stimulus = c(0L, 1L))
thresholds <- c(-1.1, -0.5, 0, 0.6, 1.3)
dat <- cbind(dat, rsdt_cdp(nrow(dat), 200, dat$stimulus,
                           dfam = 0.8, drec = 1.0,
                           thresholds = thresholds, rcrit = 0.7,
                           n_new = 3))

model <- sdt_cdp(stimulus = "stimulus", n_new = 3, n_old = 3)

fit <- bmm(
  formula = bmf(dfam ~ 1, drec ~ 1, criterion ~ 1,
                spacing ~ 1, rcrit ~ 1),
  data = dat, model = model, backend = "cmdstanr"
)

# Estimate unequal recollection variance and the F-R correlation
fit_uv <- bmm(
  formula = bmf(dfam ~ 1, drec ~ 1, criterion ~ 1,
                spacing ~ 1, rcrit ~ 1, sigmar ~ 1, rho ~ 1),
  data = dat, model = model, backend = "cmdstanr"
)

## End(Not run)
```

---

sdt_cdp_dist *Distribution functions for Continuous Dual-Process SDT (CDP)*

Description

Density and random generation for the continuous dual-process signal detection theory model (Wixted & Mickes, 2010). Two correlated continuous dimensions, Familiarity (F) and Recollection (R), generate the aggregate strength S = F + R that drives old/new confidence; Remember/Know judgments split "old" responses on R, and an optional Know/Guess split uses F. The response is a vector of counts across the response categories (multinomial likelihood), in the same format `sdt_cdp()` is fit to. See `sdt_cdp()` for the model and `sdt_rating()` when no R/K split is available.

Usage

```
dsdt_cdp(
  counts,
  stimulus,
  dfam,
  drec,
  thresholds,
  rcrit,
  kcrit = NULL,
  sigmar = 0,
  rho = 0,
```

```
  n_new = NULL,
  dist = "normal",
  log = FALSE
)

rsdt_cdp(
  n,
  n_trials,
  stimulus,
  dfam,
  drec,
  thresholds,
  rcrit,
  kcrit = NULL,
  sigmar = 0,
  rho = 0,
  n_new = NULL,
  dist = "normal"
)
```

Arguments

counts	Integer matrix with one row per observation and one column per response category in the canonical order new, [guess], know, remember (each block ordered by confidence; see sdt_cdp()), or a vector for a single observation. n_new + 2 * n_old columns (R/K) or n_new + 3 * n_old (R/K/G).
stimulus	Integer vector (0/1). Stimulus type: 0 = new/lure, 1 = old/target.
dfam, drec	Numeric vectors. Familiarity and recollection sensitivities: the target means on the F and R axes, each in units of the corresponding lure SD (both lure SDs are 1). They are component means of a bivariate latent space, not the balanced d_a that the other SDT models report as d – the old/new decision here is read off the aggregate strength S = F + R, so the model's discriminability is a derived quantity rather than either of these.
thresholds	Numeric vector of n_new + n_old - 1 ordered confidence thresholds on the aggregate F+R axis, or a matrix with one row per observation.
rcrit	Numeric vector. Remember criterion on the recollection axis.
kcrit	Know criterion on the familiarity axis. NULL (default) for the R/K model; finite values enable the R/K/G model.
sigmar	Numeric vector. Log SD of the recollection target distribution (0 = SD 1).
rho	Numeric vector. F-R correlation on the unconstrained scale; tanh(rho) is the correlation. 0 (default) = independent processes (classic CDP).
n_new	Integer number of "new" confidence levels. Defaults to half the number of confidence levels, rounded down (so 3 of 7); the number of "old" levels follows as length(thresholds) + 1 - n_new.
dist	Noise distribution. Only "normal" is currently supported.
log	Logical; if TRUE return the log-density (default FALSE).

n	Integer. Number of observations to generate. n_trials, stimulus, thresholds, and the model parameters are recycled to this length.
n_trials	Integer vector. Number of trials per observation.

Value

dsdt_cdp returns the (log-)density (multinomial probability). rsdt_cdp returns an integer matrix with one row per observation and the canonical response count columns (new1, ..., remember<K>) that sdt_cdp() expects – cbind() it to a design data frame for a ready-to-fit data set.

References

Wixted, J. T., & Mickes, L. (2010). A continuous dual-process model of remember/know judgments. *Psychological Review*, *117*(4), 1025–1054. doi:10.1037/a0020874

Examples

```
# CDP density (R/K, 1 new + 3 old levels: 3 thresholds, 7 count columns)
dsdt_cdp(
  counts = c(40, 5, 12, 30, 10, 18, 60), stimulus = 1,
  dfam = 0.8, drec = 1.0,
  thresholds = c(-0.5, 0.3, 1.0),
  rcrit = 0.7, n_new = 1
)
# Generate CDP count data (R/K, 3 new + 3 old levels) for 10 subjects
dat <- expand.grid(id = 1:10, stimulus = c(0L, 1L))
dat <- cbind(dat, rsdt_cdp(nrow(dat), 100, dat$stimulus,
                           dfam = 0.8, drec = 1.0,
                           thresholds = c(-1.1, -0.5, 0, 0.6, 1.3),
                           rcrit = 0.5, n_new = 3))
head(dat)
```

---

sdt_dpsdt_dist *Distribution functions for dual-process SDT (DPSDT)*

Description

Density and random generation for the dual-process signal detection model (Yonelinas, 1994), with recall-to-reject of new items (Rn; as in Yonelinas, 2024). Extends rating SDT with recollection probabilities Ro (old items recollected as old) and Rn (new items recall-rejected) that add mass to the most-confident rating category. These are the simulation counterparts of the dpsdt version of sdt_rating(); here Ro/Rn are supplied directly as probabilities in [0, 1].

Usage

```
dsdt_dpsdt(
  counts,
  stimulus,
```

```
  d,
  thresholds,
  Ro,
  Rn = 0,
  sdratio = 1,
  dist = c("normal", "gumbel_min", "gumbel_max", "logistic"),
  log = FALSE
)

rsdt_dpsdt(
  n,
  n_trials,
  stimulus,
  d,
  thresholds,
  Ro,
  Rn = 0,
  sdratio = 1,
  dist = c("normal", "gumbel_min", "gumbel_max", "logistic")
)
```

Arguments

counts	Integer matrix with one row per observation and one column per rating category, ordered from "definitely noise" (1) to "definitely signal" (K), or a vector for a single observation.
stimulus	Integer vector (0/1). Stimulus type: 0 = noise, 1 = signal.
d	Numeric vector. Sensitivity: d' when sdratio is 1, and otherwise the balanced index d_a (see sdt_yn()). The separation between the distributions in noise units is d * sqrt((1 + sdratio^2) / 2).
thresholds	Numeric vector of length K-1 with the ordered decision thresholds, or an n-by-(K-1) matrix with one row per observation. The thresholds are positions on the noise-standardized axis, measured from the mean of the noise distribution (the signal mean sits at d * sqrt((1 + sdratio^2) / 2)), and are not rescaled by sdratio.
Ro	Numeric vector in [0, 1]. Recollection probability for old (signal) items.
Rn	Numeric vector in [0, 1]. Recall-to-reject probability for new (noise) items. Defaults to 0, the classic one-sided model, as the dpsdt version fixes it off unless Rn is in the formula.
sdratio	Numeric vector. Ratio of signal to noise standard deviations (default 1, i.e., equal variance). Must be positive. Note that this is the natural scale: the sdratio parameter of sdt_rating() is sampled on the log scale, so it corresponds to log(sdratio) here.
dist	Character. Noise distribution: "normal" (default), "logistic", "gumbel_min", or "gumbel_max".
log	Logical. If TRUE, returns log-density (default FALSE).

n	Integer. Number of observations to generate. n_trials, stimulus, thresholds, and the model parameters are recycled to this length.
n_trials	Integer vector. Number of trials per observation.

Value

dsdt_dpsdt returns the (log-)density (multinomial probability). rsdt_dpsdt returns an integer matrix with one row per observation and one rating-count column per category (r1 ... rK).

References

Yonelinas, A. P. (1994). Receiver-operating characteristics in recognition memory: Evidence for a dual-process model. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *20*(6), 1341–1354. doi:10.1037/02787393.20.6.1341

Yonelinas, A. P. (2024). The role of recollection and familiarity in visual working memory: A mixture of threshold and signal detection processes. *Psychological Review*, *131*(2), 321–348. doi:10.1037/rev0000432

Examples

```
# Density for a single observation (K=4) with recollection of old items;
# Rn defaults to 0 (no recall-to-reject), the one-sided model
dsdt_dpsdt(counts = c(2, 8, 20, 70), stimulus = 1,
           d = 1.5, thresholds = c(0.25, 0.75, 1.25), Ro = 0.3)
# Generate DPSDT rating data (K=4) for 10 subjects and both stimulus types
dat <- expand.grid(id = 1:10, stimulus = c(0L, 1L))
dat <- cbind(dat, rsdt_dpsdt(nrow(dat), 100, dat$stimulus, d = 1.5,
                             thresholds = c(0.25, 0.75, 1.25),
                             Ro = 0.3, Rn = 0.1))
head(dat)
```

---

sdt_mafc *m-Alternative Forced Choice Signal Detection Theory Model*

Description

m-Alternative Forced Choice Signal Detection Theory Model

Usage

```
sdt_mafc(
  response,
  n_trials,
  m,
  dist = c("normal", "gumbel_min", "gumbel_max", "logistic"),
  links = NULL,
  ...
)
```

Arguments

`response`	A single string naming the column with counts of correct responses.
`n_trials`	The name of the variable containing the total number of trials per cell.
`m`	Either a single integer >= 2 giving the number of alternatives (constant across all rows), or a single string naming a data column that gives the number of alternatives per row. A column lets trials with different set sizes be fit jointly.
`dist`	The distribution assumed for the latent evidence, given here by its cumulative distribution function. One of: • "normal" (default): Gaussian m-AFC, $\Phi(x)$ • "gumbel_min": smallest-extreme-value m-AFC, $1 - \exp(-\exp(x))$ (complementary log-log) • "gumbel_max": largest-extreme-value m-AFC, $\exp(-\exp(-x))$ (log-log, as in `evd::pgumbel`) • "logistic": logistic m-AFC, $1/(1 + \exp(-x))$
`links`	A named list of link functions for the parameters.
`...`	used internally for testing, ignore it

Details

- **Domain:** Perception & Recognition Memory
- **Task:** m-Alternative Forced Choice
- **Name:** Signal Detection Theory (m-AFC)
- **Citation:**
  - Green, D. M., & Swets, J. A. (1966). Signal detection theory and psychophysics. Wiley.
  - DeCarlo, L. T. (2012). On a signal detection approach to m-alternative forced choice with bias, with maximum likelihood and Bayesian approaches to estimation. Journal of Mathematical Psychology, 56(3), 196-207. https://doi.org/10.1016/j.jmp.2012.02.004
- **Requirements:**

  Provide pre-aggregated accuracy data with the following columns:

- Response counts (response): number of correct responses
- Number of trials (n_trials): total trials per cell No stimulus column needed (each trial has exactly one signal alternative)

- **Parameters:**
  - `d`: Sensitivity: d', the distance between the signal and distractor distributions in SD units (m-AFC assumes a common SD, so this is also the d_a that sdt_yn reports under unequal variance)
- **Fixed parameters:**
- **Default parameter links:**
  - d = identity
- **Default priors:**

- d:
  - main: normal(1, 1)
  - effects: normal(0, 0.5)
  - sd: exponential(1)

Models accuracy in m-AFC tasks where each trial presents one signal among m alternatives and the observer chooses the strongest one. Only d is estimated (m-AFC has no response bias). The probability correct, $P_c = \int f(x-d)\, F(x)^{m-1}\, dx$, is computed per noise distribution: a closed-form softmax for gumbel_max, a closed-form Gamma ratio for gumbel_min, Gauss-Hermite quadrature for normal ($\Phi(d/\sqrt{2})$ at m = 2), and Gauss-Legendre quadrature for logistic.

Value

An object of class bmmodel

Sensitivity is on the same scale as sdt_yn()

d is d'. m-AFC assumes the signal and distractor distributions share an SD, so there is no sdratio parameter, and d' coincides with the balanced index d_a that sdt_yn() reports once its sdratio is estimated.

At m = 2 the two models agree: 2AFC proportion correct equals the area under the yes/no ROC (Green's theorem), and for Gaussian noise both give $P_c = \Phi(d/\sqrt{2})$. The same observer therefore yields the same d whether it is measured by a yes/no ROC or by 2AFC accuracy, which is the property that makes d_a the right common scale for the SDT family. For dist = "normal" this holds even when sdt_yn() estimates unequal variance. For the other distributions it holds when sdt_yn() keeps sdratio at its default, because only the Gaussian d_a is exactly the AUC-equivalent index (see the sensitivity section of sdt_yn()).

References

Green, D. M., & Swets, J. A. (1966). *Signal detection theory and psychophysics*. Wiley.

DeCarlo, L. T. (2012). On a signal detection approach to m-alternative forced choice with bias, with maximum likelihood and Bayesian approaches to estimation. *Journal of Mathematical Psychology*, *56*(3), 196–207. doi:10.1016/j.jmp.2012.02.004

Examples

```
## Not run:
dat <- data.frame(id = 1:20, n_trials = 200L)
dat$n_correct <- rsdt_mafc(nrow(dat), dat$n_trials, m = 4,
                           d = rnorm(20, 1.5, 0.4))

model <- sdt_mafc(
  response = "n_correct",
  n_trials = "n_trials",
  m = 4
)

fit <- bmm(
  formula = bmf(d ~ 1),
```

```
  data = dat,
  model = model,
  cores = 4,
  backend = "cmdstanr"
)

## End(Not run)
```

`sdt_mafc_dist` *Distribution functions for m-AFC SDT*

Description

Density and random generation for m-alternative forced choice signal detection theory (DeCarlo, 2012). Models accuracy in tasks where one of `m` alternatives contains the signal. Only the `d` parameter is estimated (no criterion). All arguments are recycled to the length of the longest one, so passing vectors of `d`, `m`, or `n_trials` generates (or evaluates) one observation per element.

Usage

```
dsdt_mafc(
  n_correct,
  n_trials,
  m,
  d,
  dist = c("normal", "gumbel_min", "gumbel_max", "logistic"),
  log = FALSE
)

rsdt_mafc(
  n,
  n_trials,
  m,
  d,
  dist = c("normal", "gumbel_min", "gumbel_max", "logistic")
)
```

Arguments

`n_correct`	Integer vector. Number of correct responses.
`n_trials`	Integer vector. Total number of trials per observation.
`m`	Integer vector. Number of alternatives per observation. Must be at least 2.
`d`	Numeric vector. Sensitivity d': the distance between the signal and distractor distributions in SD units. m-AFC assumes a common scale for the two distributions, so this is also the balanced index d_a that `sdt_yn()` reports.
`dist`	Character. The distribution assumed for the latent evidence, given here by its cumulative distribution function:

- "normal" (default): Gaussian, $\Phi(x)$
- "gumbel_min": smallest extreme value, $1 - \exp(-\exp(x))$ (the complementary log-log distribution)
- "gumbel_max": largest extreme value, $\exp(-\exp(-x))$ (the log-log distribution, as in `evd::pgumbel`)
- "logistic": $1/(1 + \exp(-x))$

`log` Logical. If `TRUE`, returns log-density (default `FALSE`).

`n` Integer. Number of observations to generate. `n_trials`, `m`, and `d` are recycled to this length.

Value

`dsdt_mafc` returns the (log-)density (binomial probability). `rsdt_mafc` returns an integer vector with the number of correct responses per observation.

References

DeCarlo, L. T. (2012). On a signal detection approach to m-alternative forced choice with bias, with maximum likelihood and Bayesian approaches to estimation. *Journal of Mathematical Psychology*, *56*(3), 196–207. doi:10.1016/j.jmp.2012.02.004

Examples

```
# 4-AFC density
dsdt_mafc(n_correct = 80, n_trials = 100, m = 4, d = 1.5)
# Generate 4-AFC data for 20 subjects with varying sensitivity
dat <- data.frame(id = 1:20, n_trials = 200L)
dat$n_correct <- rsdt_mafc(nrow(dat), dat$n_trials, m = 4,
                           d = rnorm(20, 1.5, 0.4))
head(dat)
```

---

sdt_metad_dist *Distribution functions for meta-d' SDT*

Description

Density and random generation for the meta-d' model (Maniscalco & Lau, 2012). Confidence thresholds are placed using the metacognitive sensitivity `metad`, then rescaled so the total "old"/"new" response rates match what type-1 `d` predicts. These are the simulation counterparts of the `metad` version of `sdt_rating()`. The old/new boundary is the middle threshold, so the number of rating categories must be even (an odd number of `thresholds`).

Usage

```
dsdt_metad(
  counts,
  stimulus,
  d,
  thresholds,
  metad,
  sdratio = 1,
  dist = c("normal", "gumbel_min", "gumbel_max", "logistic"),
  log = FALSE
)

rsdt_metad(
  n,
  n_trials,
  stimulus,
  d,
  thresholds,
  metad,
  sdratio = 1,
  dist = c("normal", "gumbel_min", "gumbel_max", "logistic")
)
```

Arguments

counts	Integer matrix with one row per observation and one column per rating category, ordered from "definitely noise" (1) to "definitely signal" (K), or a vector for a single observation.
stimulus	Integer vector (0/1). Stimulus type: 0 = noise, 1 = signal.
d	Numeric vector. Sensitivity: d' when `sdratio` is 1, and otherwise the balanced index d_a (see `sdt_yn()`). The separation between the distributions in noise units is `d * sqrt((1 + sdratio^2) / 2)`.
thresholds	Numeric vector of length K-1 with the ordered decision thresholds, or an n-by-(K-1) matrix with one row per observation. The thresholds are positions on the noise-standardized axis, measured from the mean of the noise distribution (the signal mean sits at `d * sqrt((1 + sdratio^2) / 2)`), and are not rescaled by `sdratio`.
metad	Numeric vector. Metacognitive sensitivity (type-2 sensitivity), on the same scale as `d` (d', or d_a when `sdratio` is not 1), so the M-ratio `metad / d` is unaffected by `sdratio`. `metad = d` corresponds to ideal metacognition (recovers rating SDT).
sdratio	Numeric vector. Ratio of signal to noise standard deviations (default 1, i.e., equal variance). Must be positive. Note that this is the natural scale: the `sdratio` parameter of `sdt_rating()` is sampled on the log scale, so it corresponds to `log(sdratio)` here.
dist	Character. Noise distribution: "normal" (default), "logistic", "gumbel_min", or "gumbel_max".

log	Logical. If TRUE, returns log-density (default FALSE).
n	Integer. Number of observations to generate. n_trials, stimulus, thresholds, and the model parameters are recycled to this length.
n_trials	Integer vector. Number of trials per observation.

Value

dsdt_metad returns the (log-)density (multinomial probability). rsdt_metad returns an integer matrix with one row per observation and one rating-count column per category (r1 ... rK).

References

Maniscalco, B., & Lau, H. (2012). A signal detection theoretic approach for estimating metacognitive sensitivity from confidence ratings. *Consciousness and Cognition*, *21*(1), 422–430. doi:10.1016/j.concog.2011.09.021

Examples

```
# Density for a single observation (K=4) with imperfect metacognition
dsdt_metad(counts = c(5, 15, 25, 55), stimulus = 1,
           d = 1.5, thresholds = c(0.25, 0.75, 1.25), metad = 1.0)
# Generate meta-d' rating data (K=4) for 10 subjects and both stimulus types
dat <- expand.grid(id = 1:10, stimulus = c(0L, 1L))
dat <- cbind(dat, rsdt_metad(nrow(dat), 100, dat$stimulus, d = 1.5,
                             thresholds = c(0.25, 0.75, 1.25), metad = 1.0))
head(dat)
```

---

sdt_ranking *Ranking Signal Detection Theory Model*

Description

Ranking Signal Detection Theory Model

Usage

```
sdt_ranking(response, m, dist = c("gumbel_min", "normal"), links = NULL, ...)
```

Arguments

response	A character vector of column names with the target rank-count columns, ordered from rank 1 (most likely target) to rank m (least). With a constant m, supply exactly m columns; with a varying set size, supply max(m) columns.
m	Either a single integer >= 2 giving the number of ranked items (constant across all rows), or a single string naming a data column that gives the number of ranked items per row.
dist	Character. The distribution assumed for the latent evidence, given here by its cumulative distribution function:

- "gumbel_min" (default): smallest extreme value, $1 - \exp(-\exp(x))$ (complementary log-log). Closed form via gamma-function ratios.
- "normal": Gaussian, $\Phi(x)$ (supports unequal variance via `sdratio`)

`links` A named list of link functions for the parameters. Only the link of `d` can be changed, and only for `dist = "gumbel_min"`: with `dist = "normal"` the quadrature loses accuracy at the large `d` a log link reaches, so `d` keeps the identity link. `sdratio` always keeps the identity link.

`...` used internally for testing, ignore it

Details

- **Domain:** Perception & Recognition Memory
- **Task:** Ranking Task
- **Name:** Signal Detection Theory (Ranking)
- **Citation:**
  - Meyer-Grant, C. G., Kellen, D., Harding, S. M., & Singmann, H. (2026). Extreme-value signal detection theory for recognition memory: The parametric road not taken. Psychological Review. Advance online publication. https://doi.org/10.1037/rev0000615
- **Requirements:**

  Provide pre-aggregated ranking counts in wide format:

- Rank-count columns (response): one column per rank position, each giving the number of trials in which the target received that rank (column 1 = most likely target, column m = least)
- Set size (m): a constant or a column giving the number of ranked items per row; rows with fewer ranks leave the surplus columns at 0 No stimulus column needed (all trials include exactly one target)

- **Parameters:**
  - `d`: Sensitivity: d' under equal variance (g' for gumbel_min). When sdratio is estimated, d is d_a, the distance between the target and lure distributions in units of their root-mean-square SD
- **Fixed parameters:**
- **Default parameter links:**
  - d = identity
- **Default priors:**
  - `d`:
    - `main`: normal(1, 1)
    - `effects`: normal(0, 0.5)
    - `sd`: exponential(1)

Models the rank ordering of `m` items by perceived strength. Only `d` is estimated (no criterion). Supports `dist = "gumbel_min"` (closed-form via lgamma ratios) and `dist = "normal"` (numerical quadrature, accurate to 1e-6 in each rank probability for SD ratios from 0.2 to 8, `d` from -1 to 4 and set sizes up to 20).

The model uses the native brms multinomial family: each rank position is a multinomial category whose logit is set to `log p(rank)`, so `softmax` recovers the rank distribution exactly. This means `log_lik`, `posterior_predict`, `posterior_epred`, and `pp_check` come from brms as proper joint multinomial draws.

Value

An object of class `bmmodel`

Sensitivity is on the same scale as `sdt_yn()`

`d` is d' whenever the target and lure distributions share an SD: always for `dist = "gumbel_min"`, where it is the g' of Meyer-Grant et al. (2026), and for `dist = "normal"` unless you give `sdratio` a formula. With `sdratio` estimated, `d` is the balanced index d_a that `sdt_yn()` reports, the separation divided by the root-mean-square of the two SDs; the noise-standardized separation is then `d * sqrt((1 + exp(sdratio)^2) / 2)`.

Ranking has no criterion, so d_a is not read off an ROC here. It is adopted to keep `d` on the same scale across the SDT family, and it has a direct meaning for rankings: it fixes the two-item accuracy at $\Phi(d/\sqrt{2})$ whatever `sdratio` is (see below).

Ranking is the one SDT design that identifies the variance ratio from a single condition. The rank distribution supplies `m - 1` free probabilities per set size, so with `m >= 3` there is enough information to separate `d` from `sdratio` without the criterion sweep that `sdt_yn()` needs – the *shape* of the rank distribution, not just its mean, carries the ratio.

At `m = 2` the model reduces to 2AFC and the two parameters are no longer separable: the probability of ranking the target first is the area under the yes/no ROC, which for Gaussian noise is $\Phi(d/\sqrt{2})$ whatever `sdratio` is. Keep `sdratio` fixed for two-item designs.

For Gaussian ranking (`dist = "normal"`), `sdratio` is fixed to 0 by default. Add `sdratio ~ 1` to the formula for unequal-variance ranking.

The set size `m` may be a constant or the name of a data column. Supply a column to fit trials with different set sizes in a single model: the response has `max(m)` columns, and rows with a smaller set size switch off the surplus rank categories (the multinomial then renormalizes over the valid ranks).

References

Meyer-Grant, C. G., Kellen, D., Harding, S. M., & Singmann, H. (2026). Extreme-value signal detection theory for recognition memory: The parametric road not taken. *Psychological Review*. Advance online publication. doi:10.1037/rev0000615

Examples

```
## Not run:
dat <- data.frame(id = 1:20)
dat <- cbind(dat, rsdt_ranking(20, 200, m = 4, d = 1.5))

model <- sdt_ranking(
  response = c("rank1", "rank2", "rank3", "rank4"),
  m = 4
)
```

```
fit <- bmm(
  formula = bmf(d ~ 1),
  data = dat,
  model = model,
  cores = 4,
  backend = "cmdstanr"
)

# Mixed set sizes in one model: response has max(m) columns, m is a column
model_mixed <- sdt_ranking(
  response = c("rank1", "rank2", "rank3", "rank4", "rank5"),
  m = "set_size"
)

## End(Not run)
```

`sdt_ranking_dist` *Distribution functions for Ranking SDT*

Description

Density and random generation for ranking signal detection theory (Meyer-Grant et al., 2026). Models rank ordering of m items by perceived strength. Only `d` is estimated (no criterion or stimulus column). Supports Gumbel-min (closed form) and Gaussian UV-SDT (numerical integration).

Usage

```
dsdt_ranking(
  counts,
  m,
  d,
  sdratio = 1,
  dist = c("gumbel_min", "normal"),
  log = FALSE
)

rsdt_ranking(n, n_trials, m, d, sdratio = 1, dist = c("gumbel_min", "normal"))
```

Arguments

`counts`	Integer matrix with one row per observation and one rank-count column per rank position (1 = most likely target), or a vector for a single observation. Columns beyond a row's set size `m` must be 0.
`m`	Integer vector. Number of ranked items per observation. Must be at least 2 and no larger than the number of count columns.

d	Numeric vector. Sensitivity: the distance between the target and lure distributions. It is d' when sdratio is 1 and, for dist = "gumbel_min", the g' of Meyer-Grant et al. (2026). With another sdratio it is the balanced index d_a that sdt_yn() reports (in root-mean-square SD units).
sdratio	Numeric vector. Ratio of signal to noise standard deviations (default 1, i.e., equal variance). Must be positive. Only used when dist = "normal".
dist	Character. The distribution assumed for the latent evidence: "gumbel_min" (default), the smallest extreme value distribution with cumulative distribution function $1 - \exp(-\exp(x))$, evaluated in closed form; or "normal", Gaussian UV-SDT by numerical integration.
log	Logical. If TRUE, returns log-density (default FALSE).
n	Integer. Number of observations to generate. n_trials, m, d, and sdratio are recycled to this length.
n_trials	Integer vector. Number of ranking trials per observation.

Value

dsdt_ranking returns the (log-)density (multinomial probability). rsdt_ranking returns an integer matrix with one row per observation and one rank-count column per rank position (rank1 ... rank max(m)); rows with a smaller set size have structural zeros in the surplus columns, matching the wide format sdt_ranking() expects.

Parameter scales

These functions take sdratio as the ratio itself, while sdt_ranking() estimates its logarithm (0 = equal variance): a fitted sdratio of 0.375 is a ratio of exp(0.375) = 1.455, and passing 0.375 here instead asks for a signal distribution 2.7 times narrower than the noise. That is a legal value and raises no error, so exponentiate first. d carries across unchanged.

References

Meyer-Grant, C. G., Kellen, D., Harding, S. M., & Singmann, H. (2026). Extreme-value signal detection theory for recognition memory: The parametric road not taken. *Psychological Review*. Advance online publication. doi:10.1037/rev0000615

Examples

```
# Gumbel-min ranking density
dsdt_ranking(counts = c(40, 30, 20, 10), m = 4, d = 1.0)
# Generate ranking data (m=4, Gumbel-min) for 10 subjects
dat <- data.frame(id = 1:10, set_size = 4L)
dat <- cbind(dat, rsdt_ranking(10, 100, m = 4, d = 1.0))
head(dat)
```

sdt_rating *Confidence Rating Signal Detection Theory Model*

Description

Confidence Rating Signal Detection Theory Model

Usage

```
sdt_rating(
  response,
  stimulus,
  dist = c("normal", "gumbel_min", "gumbel_max", "logistic"),
  threshold_type = c("parsimonious", "equidistant", "log_distance", "log_ratio",
    "softmax"),
  version = c("standard", "dpsdt", "metad"),
  links = NULL,
  ...
)
```

Arguments

response	A character vector of K column names containing response counts per rating category, ordered from "definitely noise" to "definitely signal".
stimulus	The name of the variable coding the stimulus type. Must be coded as 0 (noise/new) and 1 (signal/old).
dist	The distribution assumed for the latent evidence, given here by its cumulative distribution function. One of: • "normal" (default): Gaussian SDT, $\Phi(x)$ • "gumbel_min": smallest-extreme-value SDT, $1 - \exp(-\exp(x))$ (complementary log-log) • "gumbel_max": largest-extreme-value SDT, $\exp(-\exp(-x))$ (log-log, as in `evd::pgumbel`) • "logistic": logistic SDT, $1/(1 + \exp(-x))$
threshold_type	Character. Threshold parameterization: • "parsimonious" (default): 2 parameters (criterion + spacing). Thresholds follow logit-spaced canonical positions (Selker et al., 2019). • "equidistant": 2 parameters (criterion + spacing). Thresholds are equally spaced, exp(spacing) apart. • "log_distance": K-2 parameters. `delta<i>` is the log width of the interval between thresholds i and i + 1, so the intervals are free and the ordering is guaranteed (Paulewicz & Blaut, 2022).

- "log_ratio": K-2 parameters (Paulewicz & Blaut, 2022; the odd-K form is bmm's). One interval is the spread, `exp(delta)`: for even K the interval just above `criterion`, for odd K the middle category. The first interval on the other side (even K) or on each side (odd K) is a ratio times the spread, and every further interval a ratio times the first interval on its side. The deltas are therefore not exchangeable; `model$parameters` names the role of each.
- "softmax": K-2 parameters. A shared spacing parameter sets the mean interval width, exp(spacing), while K-3 delta parameters share the total width out over the intervals through a softmax (each delta is the log ratio of its interval to the last one).

`version` Character. The latent-process version of the rating model. One of `"standard"` (default, a single familiarity SDT process), `"dpsdt"` (dual-process SDT with recollection parameters `Ro`/`Rn`), or `"metad"` (meta-d' with a metacognitive efficiency parameter `logmratio`, the log M-ratio). All three use the same confidence-rating response interface. See Details.

`links` A named list of link functions for the parameters, e.g. `links = list(d = "log")`. Only `d`, `criterion` and `bias` can be set, to `"identity"`, `"log"`, `"softplus"`, `"logit"` or `"probit"`. `sdratio` and the threshold parameters keep their identity links, because the model reads each of them through `exp()` and fixes `sdratio` at 0 for equal variance. The same holds for `Ro` and `Rn`, read through `inv_logit()` and fixed off by default, and for `logmratio`, read through `exp()`.

`...` used internally for testing, ignore it

Details

Three versions share the confidence-rating response interface, selected with `version`:

Version: `standard` **(default):**

- **Domain:** Perception & Recognition Memory
- **Task:** Signal/Noise or Old/New Recognition
- **Name:** Signal Detection Theory (Confidence Rating)
- **Citation:**
  - Green, D. M., & Swets, J. A. (1966). Signal detection theory and psychophysics. Wiley.
- **Version:** standard
- **Requirements:**
  Provide pre-aggregated data with the following columns:
- Response counts: one column per rating category (K columns)
- Stimulus type (stimulus): 0 = noise, 1 = signal Categories should be ordered: 1 = 'definitely noise' to K = 'definitely signal'
- **Parameters:**
  - d: Sensitivity: d' under equal variance (the default). When sdratio is estimated, d is d_a, the distance between the signal and noise distributions in units of their root-mean-square SD
  - `criterion`: Decision criterion, measured from the mean of the noise distribution in noise-SD units: the middle threshold (the old/new boundary) for an even number of categories, the centre of the middle category for an odd number

  - bias: Alternative to criterion: the same middle threshold measured from the midpoint between the noise and signal means (0 = neutral, positive = conservative). Use one of the two
  - spacing: Threshold spacing: controls distance between adjacent thresholds (exp(spacing) ensures positive spacing)
  - sdratio: Log SD ratio: the log of the signal-to-noise standard deviation ratio, so exp(sdratio) is the ratio itself and 0 means equal variance
- **Fixed parameters:**
  - sdratio = 0
- **Default parameter links:**
  - d = identity; criterion = identity; bias = identity; spacing = identity; sdratio = identity
- **Default priors:**
  - d:
    * main: normal(1, 1)
    * effects: normal(0, 0.5)
    * sd: exponential(1)
  - criterion:
    * main: normal(0.5, 1.5)
    * effects: normal(0, 0.5)
    * sd: exponential(1)
  - bias:
    * main: normal(0, 1.5)
    * effects: normal(0, 0.5)
    * sd: exponential(2)
  - spacing:
    * main: normal(0, 0.5)
    * effects: normal(0, 0.3)
    * sd: exponential(2)
  - sdratio:
    * main: normal(0, 0.3)
    * effects: normal(0, 0.3)
    * sd: exponential(2)

Version: dpsdt**:**

Dual-process SDT (Yonelinas, 1994): a familiarity SDT process plus an all-or-none recollection process that loads the most-confident category. Ro is recollection of old items (loads the most-confident "signal" category), the only recollection term of the classic model; Rn is recall-to-reject of new items (loads the most-confident "noise" category), the two-sided extension that Yonelinas (2024) applies to visual working memory. inv_logit(Ro)/inv_logit(Rn) are the recollection probabilities. Both are fixed off by default (recovering standard); add Ro ~ 1 for the one-sided model and Ro ~ 1, Rn ~ 1 for the two-sided model. d describes the familiarity distributions only – the observed ROC is a mixture of familiarity and recollection, so it is not the discriminability of that mixture. summary() reports Ro and Rn on the logit scale; latent_sdt() returns them as probabilities (attribute extra). Two cautions. Freeing sdratio alongside Ro is weakly identified

from a single ROC: in simulation the two are correlated at about -0.8 in the posterior and `Ro` is pulled down while `exp(sdratio)` is pulled above 1; keep the familiarity process equal-variance unless the design separates them. And a recollection probability near zero is reported as the tail of its prior: on the probability scale the interval cannot include 0, so the test of "no recollection" is a comparison with the fit that leaves the parameter fixed off, not the interval.

- **Domain:** Perception & Recognition Memory
- **Task:** Signal/Noise or Old/New Recognition
- **Name:** Signal Detection Theory (Confidence Rating)
- **Citation:**
  - Yonelinas, A. P. (1994). Receiver-operating characteristics in recognition memory: Evidence for a dual-process model. Journal of Experimental Psychology: Learning, Memory, and Cognition, 20(6), 1341-1354. https://doi.org/10.1037/0278-7393.20.6.1341
  - Yonelinas, A. P. (2024). The role of recollection and familiarity in visual working memory: A mixture of threshold and signal detection processes. Psychological Review, 131(2), 321-348. https://doi.org/10.1037/rev0000432
- **Version:** dpsdt
- **Requirements:**
  Provide pre-aggregated data with the following columns:
- Response counts: one column per rating category (K columns)
- Stimulus type (stimulus): 0 = noise, 1 = signal Categories should be ordered: 1 = 'definitely noise' to K = 'definitely signal'
- **Parameters:**
  - `d`: Sensitivity: d' under equal variance (the default). When sdratio is estimated, d is d_a, the distance between the signal and noise distributions in units of their root-mean-square SD
  - `criterion`: Decision criterion, measured from the mean of the noise distribution in noise-SD units: the middle threshold (the old/new boundary) for an even number of categories, the centre of the middle category for an odd number
  - `bias`: Alternative to criterion: the same middle threshold measured from the midpoint between the noise and signal means (0 = neutral, positive = conservative). Use one of the two
  - `spacing`: Threshold spacing: controls distance between adjacent thresholds (exp(spacing) ensures positive spacing)
  - `sdratio`: Log SD ratio: the log of the signal-to-noise standard deviation ratio, so exp(sdratio) is the ratio itself and 0 means equal variance
  - `Ro`: Recollection of old items: probability inv_logit(Ro) that an old item is recollected as old, loading the most-confident 'signal' category
  - `Rn`: Recollection of new items: probability inv_logit(Rn) that a new item is recall-rejected, loading the most-confident 'noise' category
- **Fixed parameters:**
  - `sdratio` = 0
  - `Ro` = -100
  - `Rn` = -100
- **Default parameter links:**

  - d = identity; criterion = identity; bias = identity; spacing = identity; sdratio = identity; Ro = identity; Rn = identity
- **Default priors:**
  - d:
    * main: normal(1, 1)
    * effects: normal(0, 0.5)
    * sd: exponential(1)
  - criterion:
    * main: normal(0.5, 1.5)
    * effects: normal(0, 0.5)
    * sd: exponential(1)
  - bias:
    * main: normal(0, 1.5)
    * effects: normal(0, 0.5)
    * sd: exponential(2)
  - spacing:
    * main: normal(0, 0.5)
    * effects: normal(0, 0.3)
    * sd: exponential(2)
  - sdratio:
    * main: normal(0, 0.3)
    * effects: normal(0, 0.3)
    * sd: exponential(2)
  - Ro:
    * main: normal(0, 1)
    * effects: normal(0, 0.5)
    * sd: exponential(1)
  - Rn:
    * main: normal(0, 1)
    * effects: normal(0, 0.5)
    * sd: exponential(1)

Version: `metad`**:**

Meta-d' (Maniscalco & Lau, 2012): a type-2 metacognitive sensitivity governs confidence-threshold placement, with the total "old"/"new" response rates held to what type-1 d implies. Rather than estimating meta-d' directly, the model estimates `logmratio`, the log M-ratio $\log(\text{meta-d}'/d')$, and recovers meta-d' as `exp(logmratio) * d`. The M-ratio is the field-standard measure of metacognitive efficiency (Maniscalco & Lau, 2012; Fleming, 2017): estimating it on the log scale keeps meta-d' positive, regularizes it toward d, and anchors the ideal point (meta-d' = d, perfect metacognition) at `logmratio = 0`, which recovers `standard`. Type-1 and type-2 sensitivity share one scale (d', or d_a when `sdratio` is estimated), so the M-ratio is unaffected by `sdratio`. The type-1 boundary is the middle threshold (`criterion` or `bias`), so this version needs an even number of rating categories: with an odd number the middle category straddles the boundary (see "Where `criterion` and `bias` sit"). Every threshold keeps its distance from the midpoint between

the means when d is replaced by meta-d', as the HMeta-d toolbox's model equations do (Fleming, 2017, Appendix). Whether the formula estimates `criterion` or `bias` changes how the type-1 boundary is parameterized, not this constraint. Maximum-likelihood meta-d' instead constrains meta-c' = c' (Maniscalco & Lau, 2014), so its estimates differ from bmm's when the criterion is far from the midpoint. Extract the M-ratio posterior with `mratio()`.

- **Domain:** Perception & Recognition Memory
- **Task:** Signal/Noise or Old/New Recognition
- **Name:** Signal Detection Theory (Confidence Rating)
- **Citation:**
  - Maniscalco, B., & Lau, H. (2012). A signal detection theoretic approach for estimating metacognitive sensitivity from confidence ratings. Consciousness and Cognition, 21(1), 422-430. https://doi.org/10.1016/j.concog.2011.09.021
- **Version:** metad
- **Requirements:**
  Provide pre-aggregated data with the following columns:
- Response counts: one column per rating category (K columns)
- Stimulus type (stimulus): 0 = noise, 1 = signal Categories should be ordered: 1 = 'definitely noise' to K = 'definitely signal'
- **Parameters:**
  - d: Sensitivity: d' under equal variance (the default). When sdratio is estimated, d is d_a, the distance between the signal and noise distributions in units of their root-mean-square SD
  - `criterion`: Decision criterion, measured from the mean of the noise distribution in noise-SD units: the middle threshold (the old/new boundary) for an even number of categories, the centre of the middle category for an odd number
  - `bias`: Alternative to criterion: the same middle threshold measured from the midpoint between the noise and signal means (0 = neutral, positive = conservative). Use one of the two
  - `spacing`: Threshold spacing: controls distance between adjacent thresholds (exp(spacing) ensures positive spacing)
  - `sdratio`: Log SD ratio: the log of the signal-to-noise standard deviation ratio, so exp(sdratio) is the ratio itself and 0 means equal variance
  - `logmratio`: Log M-ratio, log(meta-d/d): metacognitive efficiency. 0 is ideal metacognition (meta-d = d), negative is inefficiency, positive is hyper-efficiency. meta-d is recovered as exp(logmratio) * d, on the same scale as d
- **Fixed parameters:**
  - `sdratio` = 0
- **Default parameter links:**
  - d = identity; criterion = identity; bias = identity; spacing = identity; sdratio = identity; logmratio = identity
- **Default priors:**
  - d:
    - `main`: normal(1, 1)
    - `effects`: normal(0, 0.5)

    - sd: exponential(1)
  - criterion:
    - main: normal(0.5, 1.5)
    - effects: normal(0, 0.5)
    - sd: exponential(1)
  - bias:
    - main: normal(0, 1.5)
    - effects: normal(0, 0.5)
    - sd: exponential(2)
  - spacing:
    - main: normal(0, 0.5)
    - effects: normal(0, 0.3)
    - sd: exponential(2)
  - sdratio:
    - main: normal(0, 0.3)
    - effects: normal(0, 0.3)
    - sd: exponential(2)
  - logmratio:
    - main: normal(0, 0.5)
    - effects: normal(0, 0.3)
    - sd: exponential(2)

  By default, the model assumes equal variance (sdratio fixed to 0). To estimate unequal variance, add `sdratio ~ 1` (or `sdratio ~ predictors`) to the formula.

Value

An object of class `bmmodel`

Sensitivity is on the same scale as `sdt_yn()`

`d` is d' whenever `sdratio` stays fixed at 0, which is every fit that does not give `sdratio` a formula. With `sdratio` estimated, `d` is the balanced index d_a that `sdt_yn()` reports: the separation between the signal and noise distributions divided by the root-mean-square of their SDs. Unlike the noise-standardized d', it remains comparable across conditions that differ in `sdratio`; see the sensitivity section of `sdt_yn()` for the reasoning, for how far the two indices lie apart, and for the caveat that under the Gumbel distributions d_a is not the AUC-equivalent index once `sdratio` is estimated.

The criterion and the confidence thresholds are **not** rescaled. They stay on the noise-standardized axis, so under unequal variance they and `d` are in different units.

The `dpsdt` and `metad` versions inherit the same convention: `d` there describes the familiarity (type-1) distributions, and meta-d' is scaled the same way, which leaves the M-ratio invariant to `sdratio`.

Where `criterion` and `bias` sit

Every `threshold_type` places the criterion the same way. With an even number of categories K it is the middle threshold, the boundary between the K/2 "noise" categories and the K/2 "signal" categories. With an odd number there is no such boundary – the middle category straddles it – so the criterion is the centre of that category, and the two thresholds around it sit half an interval below and above. A shift in the criterion therefore moves the whole threshold set, and the threshold parameters (`spacing`, `delta`) describe its shape around that point.

As in `sdt_yn()`, the formula estimates that point either as `criterion`, measured from the mean of the noise distribution, or as `bias`, measured from the midpoint between the noise and signal means (`bias = criterion - d * sqrt((1 + exp(sdratio)^2) / 2) / 2`); a formula that names neither estimates `criterion ~ 1`. The two differ in what a linear formula holds constant when `d` or `sdratio` varies; the section "Choosing between `criterion` and `bias`" of `sdt_yn()` says which to choose. Threshold locations reported by `sdt_thresholds()` and `latent_sdt()`, and the `thresholds` taken by `rsdt_rating()` and its siblings, are positions measured from the noise mean.

Identifying `sdratio`

Unlike `sdt_yn()`, a rating design identifies `sdratio` from a single condition: the K - 1 thresholds give K - 1 operating points per condition, which trace the ROC; under `dist = "normal"` its z-transform is a line with slope `1 / exp(sdratio)`. `sdratio ~ 1` on a one-condition dataset is identified from K = 3 categories, where it uses every degree of freedom (the fit is saturated), and leaves the zROC testable from K = 4; the identification caveats on the `sdt_yn()` page do not carry over.

References

Yonelinas, A. P. (1994). Receiver-operating characteristics in recognition memory: Evidence for a dual-process model. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *20*(6), 1341–1354.

Yonelinas, A. P. (2024). The role of recollection and familiarity in visual working memory: A mixture of threshold and signal detection processes. *Psychological Review*, *131*(2), 321–348. doi:10.1037/rev0000432

Maniscalco, B., & Lau, H. (2012). A signal detection theoretic approach for estimating metacognitive sensitivity from confidence ratings. *Consciousness and Cognition*, *21*(1), 422–430.

Maniscalco, B., & Lau, H. (2014). Signal detection theory analysis of type 1 and type 2 data: meta-d', response-specific meta-d', and the unequal variance SDT model. In S. M. Fleming & C. D. Frith (Eds.), *The cognitive neuroscience of metacognition* (pp. 25–66). Springer.

Fleming, S. M. (2017). HMeta-d: hierarchical Bayesian estimation of metacognitive efficiency from confidence ratings. *Neuroscience of Consciousness*, *2017*(1), nix007. doi:10.1093/nc/nix007

Green, D. M., & Swets, J. A. (1966). *Signal detection theory and psychophysics*. Wiley.

Selker, R., van den Bergh, D., Criss, A. H., & Wagenmakers, E.-J. (2019). Parsimonious estimation of signal detection models from confidence ratings. *Behavior Research Methods*, *51*(5), 1953–1967. doi:10.3758/s13428019012313

Paulewicz, B., & Blaut, A. (2022). The general causal cumulative model of ordinal response. *PsyArXiv*. doi:10.31234/osf.io/e7a3x

Examples

```
## Not run:
# EV-SDT rating model
dat <- expand.grid(id = 1:20, stimulus = c(0L, 1L))
dat <- cbind(dat, rsdt_rating(nrow(dat), 200, dat$stimulus,
                              d = 1.5, thresholds = c(0.25, 0.75, 1.25)))

model <- sdt_rating(
  response = c("r1", "r2", "r3", "r4"),
  stimulus = "stimulus"
)

fit <- bmm(
  formula = bmf(d ~ 1, criterion ~ 1, spacing ~ 1),
  data = dat,
  model = model,
  cores = 4,
  backend = "cmdstanr"
)

# UV-SDT: add sdratio to the formula
fit_uv <- bmm(
  formula = bmf(d ~ 1, criterion ~ 1, spacing ~ 1, sdratio ~ 1),
  data = dat,
  model = model,
  cores = 4,
  backend = "cmdstanr"
)

# Dual-process SDT: free recollection of old items (one-sided) or both
# old and new items (two-sided) via the formula
model_dp <- sdt_rating(c("r1", "r2", "r3", "r4"), "stimulus", version = "dpsdt")
fit_dp <- bmm(bmf(d ~ 1, criterion ~ 1, spacing ~ 1, Ro ~ 1, Rn ~ 1),
              data = dat, model = model_dp, backend = "cmdstanr")

# Meta-d': estimate metacognitive efficiency (log M-ratio)
model_md <- sdt_rating(c("r1", "r2", "r3", "r4"), "stimulus", version = "metad")
fit_md <- bmm(bmf(d ~ 1, criterion ~ 1, spacing ~ 1, logmratio ~ 1),
              data = dat, model = model_md, backend = "cmdstanr")
mratio(fit_md) # posterior M-ratio (meta-d'/d')

## End(Not run)
```

---

sdt_rating_dist *Distribution functions for Confidence Rating SDT*

Description

Density and random generation for confidence rating signal detection theory models. The response is a vector of counts across K ordered rating categories (multinomial likelihood).

Usage

```
dsdt_rating(
  counts,
  stimulus,
  d,
  thresholds,
  sdratio = 1,
  dist = c("normal", "gumbel_min", "gumbel_max", "logistic"),
  log = FALSE
)

rsdt_rating(
  n,
  n_trials,
  stimulus,
  d,
  thresholds,
  sdratio = 1,
  dist = c("normal", "gumbel_min", "gumbel_max", "logistic")
)
```

Arguments

counts	Integer matrix with one row per observation and one column per rating category, ordered from "definitely noise" (1) to "definitely signal" (K), or a vector for a single observation.
stimulus	Integer vector (0/1). Stimulus type: 0 = noise, 1 = signal.
d	Numeric vector. Sensitivity: d' when `sdratio` is 1, and otherwise the balanced index d_a (see `sdt_yn()`). The separation between the distributions in noise units is `d * sqrt((1 + sdratio^2) / 2)`.
thresholds	Numeric vector of length K-1 with the ordered decision thresholds, or an n-by-(K-1) matrix with one row per observation. The thresholds are positions on the noise-standardized axis, measured from the mean of the noise distribution (the signal mean sits at `d * sqrt((1 + sdratio^2) / 2)`), and are not rescaled by `sdratio`.
sdratio	Numeric vector. Ratio of signal to noise standard deviations (default 1, i.e., equal variance). Must be positive. Note that this is the natural scale: the `sdratio` parameter of `sdt_rating()` is sampled on the log scale, so it corresponds to `log(sdratio)` here.
dist	Character. Noise distribution: "normal" (default), "logistic", "gumbel_min", or "gumbel_max".
log	Logical. If `TRUE`, returns log-density (default `FALSE`).
n	Integer. Number of observations to generate. `n_trials`, `stimulus`, `thresholds`, and the model parameters are recycled to this length.
n_trials	Integer vector. Number of trials per observation.

Value

`dsdt_rating` returns the (log-)density (multinomial probability). `rsdt_rating` returns an integer matrix with one row per observation and one rating-count column per category (`r1` ... `rK`).

References

Green, D. M., & Swets, J. A. (1966). *Signal detection theory and psychophysics*. Wiley.

Selker, R., van den Bergh, D., Criss, A. H., & Wagenmakers, E.-J. (2019). Parsimonious estimation of signal detection models from confidence ratings. *Behavior Research Methods*, *51*(5), 1953–1967. doi:10.3758/s13428019012313

Examples

```
# Density for a single observation (K=4)
dsdt_rating(counts = c(5, 15, 25, 55), stimulus = 1,
            d = 1.5, thresholds = c(0.25, 0.75, 1.25))
# Generate rating data (K=4) for 10 subjects and both stimulus types
dat <- expand.grid(id = 1:10, stimulus = c(0L, 1L))
dat <- cbind(dat, rsdt_rating(nrow(dat), 100, dat$stimulus,
                              d = 1.5, thresholds = c(0.25, 0.75, 1.25)))
head(dat)
```

`sdt_sensitivity` *Sensitivity on the noise, signal, or root-mean-square scale*

Description

Re-expresses the posterior sensitivity of a fitted SDT model against a different reference standard deviation. As long as `sdratio` is fixed at its default, `d` is the familiar d' and all three scales below are the same number. Once `sdratio` is estimated, `d` is d_a, which measures the separation of the two evidence distributions in units of their root-mean-square SD, and the same separation can also be read against the noise SD (d_N, the classical d') or against the signal SD (d_S). This function returns any of the three as posterior draws, so contrasts and intervals can be computed on whichever scale a literature reports.

Usage

```
sdt_sensitivity(
  fit,
  measure = c("da", "dn", "ds"),
  conditions = NULL,
  probs = c(0.025, 0.975),
  ...
)
```

Arguments

fit	A bmmfit object returned by bmm() from an SDT model.
measure	Character vector naming the scales to return: "da" (root-mean-square SD, the estimated parameter), "dn" (noise SD), and/or "ds" (signal SD). Defaults to all three.
conditions	Optional data frame of predictor values at which to evaluate the model. Column names must match predictor variables used in the formula. If NULL (default), unique predictor combinations are derived from the data.
probs	Numeric vector of length 2. Lower and upper quantiles for the credible interval (default c(0.025, 0.975)).
...	Additional arguments passed to brms::posterior_linpred(), such as draw_ids to use a subset of the posterior draws. ndraws is refused: each parameter is drawn by its own call, so random subsets would not match across parameters.

Details

The model places the noise distribution at $-\delta/2$ with SD $\sigma_N = 1$ and the signal distribution at $+\delta/2$ with SD $\sigma_S = \exp(\text{sdratio})$, so δ is the separation in noise-SD units. Dividing that separation by each reference SD gives

$$d_N = \delta/\sigma_N, \quad d_S = \delta/\sigma_S, \quad d_a = \delta/\sqrt{(\sigma_N^2 + \sigma_S^2)/2}.$$

The estimated parameter is d_a, hence $\delta = d_a\sqrt{(1+\sigma_S^2)/2}$ and

$$d_N = d_a\sqrt{(1+\sigma_S^2)/2}, \qquad d_S = d_a\sqrt{(1+\sigma_S^2)/2}\,/\,\sigma_S.$$

All three coincide when sdratio is 0 (equal variance), which is the case for sdt_mafc() and for any fit that leaves sdratio at its default of 0. The conversion is applied draw by draw, so the returned intervals propagate the joint posterior uncertainty in d and sdratio rather than combining point estimates.

Only d_a is invariant to which distribution is treated as the reference. d_N and d_S are not comparable across conditions that differ in sdratio: two conditions with identical discriminability can show a large, confidently estimated d_N difference. Prefer d_a for contrasts, and use d_N/d_S for comparison with published values. When sdratio is estimated but constant across the conditions being compared, all three differ by one common factor per draw and give the same contrasts up to scale.

The criterion and rating thresholds are not converted: they stay on the noise-SD axis whichever sensitivity scale you report. Under the Gumbel distributions with sdratio estimated, none of the three is the AUC-equivalent index; use auc_sdt() to compare discriminability on the probability scale.

Value

A data frame of class "bmm_sdt_sensitivity" with columns measure, value, .draw, and any condition columns. The object carries a summary attribute (measure, mean, lower, upper, plus condition columns), and probs, model_class, dist and conditions attributes.

See Also

auc_sdt(), roc_sdt(), latent_sdt()

Examples

```
## Not run:
# Three base-rate conditions shift the criterion, which identifies sdratio
dat <- expand.grid(id = 1:20, stimulus = c(0L, 1L),
                   condition = c("liberal", "neutral", "strict"))
dat$n_trials <- 100L
# simulated as bias, measured from the midpoint between the means (0 = boundary at the midpoint)
criteria <- c(liberal = -0.5, neutral = 0, strict = 0.5)
dat$n_old <- rsdt_yn(nrow(dat), dat$n_trials, dat$stimulus, d = 1.5,
                     bias = criteria[as.character(dat$condition)],
                     sdratio = 1.3)

fit <- bmm(
  formula = bmf(d ~ 1, criterion ~ 0 + condition, sdratio ~ 1),
  data = dat,
  model = sdt_yn(response = "n_old", stimulus = "stimulus",
                 n_trials = "n_trials"),
  cores = 4,
  backend = "cmdstanr"
)

# d_a (the fitted d), d_N and d_S, converted draw by draw
sdt_sensitivity(fit)
sdt_sensitivity(fit, measure = "dn", draw_ids = 1:500)

## End(Not run)
```

`sdt_thresholds` *Latent decision thresholds from a fitted rating SDT model*

Description

Extracts the $K - 1$ latent decision thresholds of a rating signal detection model fit with bmm(). Thresholds are not posterior parameters: they are reconstructed per draw from the `criterion` and the parameterization parameters (`spacing` and/or `delta*`), whose mapping to ordered thresholds depends on the model's `threshold_type`. This function returns those draws on the latent decision-variable scale together with a posterior summary, so threshold estimates are accessible without knowing the parameterization. Which threshold separates "noise" from "signal" responses depends on whether the number of categories is even or odd; see the section "Where `criterion` sits" in sdt_rating().

Usage

```
sdt_thresholds(fit, conditions = NULL, probs = c(0.025, 0.975), ...)
```

Arguments

fit	A bmmfit object returned by bmm() from an SDT model.
conditions	Optional data frame of predictor values at which to evaluate the model. Column names must match predictor variables used in the formula. If NULL (default), unique predictor combinations are derived from the data.
probs	Numeric vector of length 2. Lower and upper quantiles for the credible interval (default c(0.025, 0.975)).
...	Additional arguments passed to brms::posterior_linpred(), such as draw_ids to use a subset of the posterior draws. ndraws is refused: each parameter is drawn by its own call, so random subsets would not match across parameters.

Value

A data frame of class "bmm_sdt_thresholds" with columns marker (threshold label t1, t2, ...), position (latent location), .draw, and any condition columns. The object carries a summary attribute (marker, position posterior mean, lower, upper, plus condition columns). The position/lower/upper naming matches the lines attribute of latent_sdt(), which visualises the same quantities. The object also carries probs, model_class, dist and conditions.

See Also

latent_sdt(), roc_sdt()

Examples

```
## Not run:
dat <- expand.grid(id = 1:20, stimulus = c(0L, 1L))
dat <- cbind(dat, rsdt_rating(nrow(dat), 200, dat$stimulus, d = 1.5,
                              thresholds = c(0.25, 0.75, 1.25), sdratio = 1.3))

fit <- bmm(
  formula = bmf(d ~ 1, criterion ~ 1, spacing ~ 1, sdratio ~ 1),
  data = dat,
  model = sdt_rating(response = c("r1", "r2", "r3", "r4"),
                     stimulus = "stimulus"),
  cores = 4,
  backend = "cmdstanr"
)

sdt_thresholds(fit)

## End(Not run)
```

sdt_yn *Yes/No Signal Detection Theory Model*

Description

Estimates sensitivity (`d`, which is d' under equal variance and d_a once `sdratio` is estimated) and the decision criterion from yes/no detection or old/new recognition counts. The criterion is estimated either as `criterion`, measured from the mean of the noise distribution, or as `bias`, measured from the midpoint between the noise and signal means; the formula names the one to estimate.

Usage

```
sdt_yn(
  response,
  stimulus,
  n_trials,
  dist = c("normal", "gumbel_min", "gumbel_max", "logistic"),
  links = NULL,
  ...
)
```

Arguments

`response`	The name of the variable in the dataset containing the count of "old"/"signal" responses for each cell.
`stimulus`	The name of the variable in the dataset coding the stimulus type: 0 (noise/new) and 1 (signal/old). Logical, and factor or character columns holding "0"/"1", are coerced automatically; anything else (e.g. "noise"/"signal" labels) must be recoded by hand, since bmm cannot guess which level is the signal. `dsdt_yn()` and `rsdt_yn()` take the same column but are stricter, accepting only numeric or logical input.
`n_trials`	The name of the variable in the dataset containing the total number of trials for each cell. It may differ from cell to cell. Trial-level data also works: keep one row per trial, with the column named by `response` holding 0 or 1 and the column named by `n_trials` holding 1s.
`dist`	The noise distribution assumed for the latent evidence variable, given here by its cumulative distribution function. One of: • "normal" (default): Gaussian SDT, $\Phi(x)$ • "gumbel_min": smallest-extreme-value SDT, $1 - \exp(-\exp(x))$ (complementary log-log) • "gumbel_max": largest-extreme-value SDT, $\exp(-\exp(-x))$ (log-log, as in `evd::pgumbel`) • "logistic": logistic SDT, $1/(1 + \exp(-x))$

`links`	A named list of link functions for the parameters, one entry per parameter you want to change, e.g. `links = list(d = "log")`. Only `d`, `criterion` and `bias` can be set. `sdratio` keeps its log link, because the model's default of an equal-variance SD ratio is stored as the 0 that the log link maps to 1; read on any other link that same 0 is a different ratio, and on an identity link it is a ratio of zero that makes the signal trials' evidence infinite.
...	used internally for testing, ignore it

Details

- **Domain:** Perception & Recognition Memory
- **Task:** Yes/No Detection or Old/New Recognition
- **Name:** Signal Detection Theory (Yes/No)
- **Citation:**
  - Green, D. M., & Swets, J. A. (1966). Signal detection theory and psychophysics. Wiley.
- **Requirements:**
  Provide pre-aggregated data with the following columns:
- Response counts (response): number of 'old'/'signal' responses
- Stimulus type (stimulus): 0 = noise, 1 = signal
- Number of trials (n_trials): total trials per cell
- **Parameters:**
  - `d`: d_a sensitivity (= d' when sdratio is fixed): the distance between the signal and noise distributions in units of their root-mean-square SD
  - `criterion`: Decision criterion, measured from the mean of the noise distribution in noise-SD units; the false-alarm rate depends on it alone
  - `bias`: Alternative to criterion: the decision criterion measured from the midpoint between the noise and signal means (0 = neutral, positive = conservative). Use one of the two
  - `sdratio`: SD ratio signal/noise, log link (0 = equal SDs): summary() prints the log of this ratio, not the ratio itself, so exp() a posterior value to read it
- **Fixed parameters:**
  - `sdratio` = 0
- **Default parameter links:**
  - d = identity; criterion = identity; bias = identity; sdratio = log
- **Default priors:**
  - `d`:
    - `main`: normal(1, 1)
    - `effects`: normal(0, 0.5)
    - `sd`: exponential(1)
  - `criterion`:
    - `main`: normal(0.5, 1.5)
    - `effects`: normal(0, 0.5)

    * sd: exponential(1)
- bias:
    * main: normal(0, 1.5)
    * effects: normal(0, 0.5)
    * sd: exponential(2)
- sdratio:
    * main: normal(0, 0.3)
    * effects: normal(0, 0.3)
    * sd: exponential(2)

Value

An object of class bmmodel

Which sensitivity measure d is

d is the familiar d' whenever sdratio keeps its default of 0 (an SD ratio of 1, equal variance), which is every fit that does not give sdratio a formula. The rest of this section only matters once you estimate sdratio.

When the signal and noise distributions have different widths, their separation only becomes dimensionless after choosing an SD to divide by. bmm then reports d_a, the separation divided by the root-mean-square of the two SDs:

$$d_a = \sqrt{2}\,\delta/\sqrt{1+r^2},$$

where δ is the separation in noise-SD units and r is the SD ratio, exp(sdratio). This weights the two distributions equally, and it is the measure Simpson and Fitter (1973), Macmillan and Creelman (2005), and Mickes et al. (2007) recommend under unequal variance. The classical noise-standardized index is $d_N = \delta = d_a\sqrt{(1+r^2)/2}$, so a published d' from an unequal-variance analysis is larger than d when $r > 1$: by 13% at $r = 1.25$ and by 33% at $r = 1.6$. sdt_sensitivity() returns d_N and the signal-standardized d_S from a fitted model, and summary() notes when d is d_a.

d is d_a rather than d_N because only d_a is comparable across conditions or subjects that differ in sdratio: two conditions that are equally discriminable can show a large, confidently estimated difference in d_N. When sdratio is estimated but constant across the conditions you compare, the two indices differ by one common factor and give the same contrasts up to scale.

Units of the other parameters. criterion and bias are *not* rescaled: both are in noise-SD units, so under unequal variance d (in root-mean-square SD units) and the criterion are on different scales, and a ratio such as bias / d mixes them.

Extreme-value distributions. For dist = "normal", d_a is also the AUC-equivalent index, $d_a = \sqrt{2}\,\Phi^{-1}(\mathrm{AUC})$, so it carries the same information as 2AFC accuracy. For "gumbel_min" and "gumbel_max" that identity holds only under equal variance. With sdratio estimated, d_a keeps its balanced geometry but drifts away from the AUC-equivalent index as the SD ratio moves away from 1 (for "gumbel_min" at $\delta = 1.5$ and $r = 2$, d_a is 31% larger), so compare such fits on the AUC rather than on d.

Because d is a short name, a column called d in your data that is also used as a predictor will collide with this parameter; bmm() warns when that happens.

Choosing between `criterion` and `bias`

Both parameters place the same decision boundary on the noise-standardized evidence axis, and they differ only in the point they measure it from:

- `criterion` is the distance from the mean of the noise distribution. The false-alarm rate depends on it alone: $FA = 1 - F(criterion)$.
- `bias` is the distance from the midpoint between the noise and signal means: 0 is the neutral point, positive values a conservative bias toward "new"/"noise". Under equal variance it is the index usually written c.

They are related by $bias = criterion - \delta/2$, where $\delta = d\sqrt{(1 + r^2)/2}$ is the separation in noise-SD units. Write one of them in the formula; a formula that names neither estimates `criterion ~ 1`, and one that names both is refused. `sdt_criterion()` and `sdt_bias()` compute the two from a single hit and false-alarm rate.

The choice matters whenever `d` or `sdratio` differs between rows that share a criterion term, because a linear formula holds constant the parameter you write. `criterion ~ 1` with `d ~ 1 + strength` says the observer keeps one boundary while targets get stronger; `bias ~ 1` with the same `d` formula says the boundary moves up by half of every change in separation. When the criterion and `d` share condition-wise (factor) predictors and `sdratio` does not vary within those cells, or share predictors while `d` keeps its identity link and `sdratio` is the same in every row, the two give the same likelihood and their coefficients differ by $\delta/2$. Otherwise they are different models. Even with the same likelihood, the posteriors differ through the default priors: `criterion` takes `normal(0.5, 1.5)` for its intercept, or for each cell under `~ 0 + condition`, and `exponential(1)` for its random-effect SDs, while `bias` takes `normal(0, 1.5)` and `exponential(2)`.

Use `criterion` when the lures, and the observer's boundary, do not depend on the manipulation of `d`: the classic recognition design in which weak and strong targets are tested against one shared set of lures under the same criteria. Its lure rows have no strength level, and with `criterion` they need none, because the false-alarm rate does not involve `d`. Use `bias` when you want to hold, or compare, the observer's position relative to the two distributions, e.g. a bias contrast between groups that differ in sensitivity.

`bias` is measured from the midpoint, which is the point where the two densities are equal – the optimal boundary for equal base rates and payoffs – only for the symmetric distributions (`"normal"`, `"logistic"`) under equal variance. Under unequal variance, or for the Gumbel distributions, `bias = 0` is the midpoint, not the optimum.

Identifying `sdratio`

`sdratio` needs a design that supplies more than one operating point, and whether it gets one depends on which criterion parameter the formula holds constant.

With `criterion`, the boundary must move among rows that share the same `d`: give `criterion` a predictor that shifts it while `d` stays put, such as a base-rate, payoff, or confidence manipulation (`d ~ 1`, `criterion ~ 0 + condition`; see broeder_schuetz_2009_e3). A criterion that is the same in every row is fixed by the false-alarm rate, and each hit rate then fixes only $(\delta - criterion)/r$, so `sdratio` trades against `d`. A change in `sdratio` is matched by shifting the separation of every cell by one common amount and rescaling it by one common factor, and whenever the formula for `d` can make that change, `sdratio` returns its prior. A factor can, under the identity and the log link alike (`d ~ 0 + condition` or `d ~ 1 + condition`), because it gives every cell a value of its

own. An intercept with a numeric predictor can under the identity link (`d ~ 1 + strength`), because the intercept shifts and the slope rescales. `sdratio` is identified only when the formula constrains `d` across cells beyond that. A numeric predictor without an intercept (`d ~ 0 + strength`) holds `d` proportional to `strength`. It identifies `sdratio` when `strength` takes at least two nonzero values and the criterion lies clearly away from the noise mean (for `"normal"`, a false-alarm rate clearly away from .5); at a criterion of 0 it does not. The estimate then rests on the assumption that `d` is zero where `strength` is zero, not on a second operating point in the data. A numeric predictor under a log link on `d` barely identifies `sdratio`, and in practice `sdratio` returns its prior. `bmm()` warns for every criterion that is the same in every row, so it also warns for `d ~ 0 + strength`. Giving `d` and `criterion` the same condition-wise predictors (`d ~ 0 + condition`, `criterion ~ 0 + condition`) fails too, because each condition then has two rates for two parameters of its own, and no warning catches it. A random effect such as `criterion ~ 1 + (1 | id)` moves the boundary between subjects and does not trigger the warning, but it informs `sdratio` only weakly, through the assumption that the subject effects on `d` and on the criterion are independent.

With `bias`, any predictor that moves the operating point supplies what is missing, including a sensitivity manipulation such as `d ~ 0 + condition` with `bias ~ 1`: holding the bias constant moves the criterion with the separation, and that movement traces the ROC. The identification then rests on that assumption about the observer. The design in which every parameter is intercept-only with no random effects cannot work: a single (hit, false-alarm) pair is two numbers for three unknowns, and `bmm()` warns for it. Neither can `bias` and `d` with the same condition-wise predictors, for the reason given above, and that design raises no warning.

Sampling converges and `Rhat` is fine in every unidentified case, so the warnings are the only signal. They count formula terms, not how far the terms move the operating point, so a predictor with a small effect carries little information without a warning. Leaving `sdratio` at its default is always identified.

Reading `sdratio` and carrying it to `dsdt_yn()`/`rsdt_yn()`

As in every `bmm` model, the parameters the model *estimates* are on their link scale, while the distribution functions take their arguments on the *natural* scale. `sdratio` has a log link, so `summary()` reports $\log r$ whereas `dsdt_yn()` and `rsdt_yn()` expect the ratio r itself (their default is 1, equal variance). Exponentiate before carrying a posterior value across:

```
r <- exp(as_draws_matrix(fit)[, "b_sdratio_Intercept"])
```

A posterior mean of `sdratio = 0.22` is a ratio of `exp(0.22) = 1.25`. Passing `0.22` straight to `rsdt_yn()` instead asks for a signal distribution 4.5 times *narrower* than the noise — a legal value that raises no error, and the one mistake worth checking for in a posterior predictive check written by hand. `d`, `criterion` and `bias` have identity links, so they carry across unchanged.

The **zROC slope** reported in the recognition-memory literature is the reciprocal of that ratio, `1 / exp(sdratio)`, so a `sdratio` posterior mean of 0.375 is a zROC slope of 0.69.

The same log link applies going the other way: a constant you supply yourself, whether as `bmf(sdratio = )` or through a hand-written `brms::set_prior(..., dpar = "sdratio")`, is read on it too. `bmf(sdratio = 1)` does not fix a ratio of 1 — it fixes `exp(1) = 2.72`, and `bmm()` raises no warning; a fixed ratio of 1.25 needs `bmf(sdratio = log(1.25))`. Both `sdratio` defaults `default_prior()` reports — `normal(0, 0.3)` on the intercept and `exponential(2)` on the random-effect SDs — are on that same scale, unannotated.

Terms used on this page

- **noise-standardized axis** — latent evidence expressed in units of the noise distribution's SD, with its origin at the noise mean. `criterion` is a position on this axis; `bias` is a distance on it, measured from the midpoint between the means.
- δ (**separation**) — the distance between the signal and noise means, on that axis.
- d_N — the classical d': δ itself, i.e. the separation in noise-SD units.
- d_a — the separation divided by the root-mean-square of the two SDs, which is what `d` reports. $d_a = d_N$ under equal variance.
- **operating point** — one (false-alarm rate, hit rate) pair, i.e. one point of an ROC curve. One condition gives one operating point.
- **AUC** — the area under that ROC curve, equivalently the probability that a random signal trial yields more evidence than a random noise trial. Obtain it from the posterior with `pnorm(d / sqrt(2))` for `dist = "normal"`.
- `main` / `effects` / `sd` — the keys of the default priors shown in the model description above: `main` is the prior on the intercept, `effects` the prior on regression coefficients, and `sd` the prior on the standard deviations of the parameter's random effects.

References

Green, D. M., & Swets, J. A. (1966). *Signal detection theory and psychophysics*. Wiley.

DeCarlo, L. T. (1998). Signal detection theory and generalized linear models. *Psychological Methods*, *3*(2), 186–205. doi:10.1037/1082989X.3.2.186

Simpson, A. J., & Fitter, M. J. (1973). What is the best index of detectability? *Psychological Bulletin*, *80*(6), 481–488. doi:10.1037/h0035203

Macmillan, N. A., & Creelman, C. D. (2005). *Detection theory: A user's guide* (2nd ed.). Erlbaum.

Mickes, L., Wixted, J. T., & Wais, P. E. (2007). A direct test of the unequal-variance signal detection model of recognition memory. *Psychonomic Bulletin & Review*, *14*(5), 858–865. doi:10.3758/BF03194112

See Also

`sdt_d()`, `sdt_criterion()` and `sdt_bias()` compute the `d`, `criterion` and `bias` of this model in closed form from a single pair of observed hit and false-alarm rates, without fitting: use them for a quick check of a fitted value, and this model when you need a hierarchical or condition-wise estimate, or unequal variance.

Examples

```
## Not run:
dat <- expand.grid(id = 1:20, stimulus = c(0L, 1L))
dat$n_trials <- 100L
dat$n_old <- rsdt_yn(nrow(dat), dat$n_trials, dat$stimulus,
                     d = 1.5, criterion = 0.9)

model <- sdt_yn(
  response = "n_old",
```

```
  stimulus = "stimulus",
  n_trials = "n_trials"
)

fit <- bmm(
  formula = bmf(d ~ 1, criterion ~ 1),
  data = dat,
  model = model,
  cores = 4,
  backend = "cmdstanr"
)

# Sensitivity and bias per participant
fit_re <- bmm(
  formula = bmf(d ~ 1 + (1 | id), criterion ~ 1 + (1 | id)),
  data = dat,
  model = model,
  cores = 4,
  backend = "cmdstanr"
)

# Unequal-variance yes/no SDT. sdratio needs more than one operating point:
# on the single-condition `dat` above it would not be identified.
# `model` already names the columns this dataset uses.
fit_uv <- bmm(
  formula = bmf(d ~ 1, criterion ~ 0 + condition, sdratio ~ 1),
  data = broeder_schuetz_2009_e3,
  model = model,
  cores = 4,
  backend = "cmdstanr"
)

# The same observer, with the boundary measured from the midpoint
fit_bias <- bmm(
  formula = bmf(d ~ 1 + (1 | id), bias ~ 1 + (1 | id)),
  data = dat,
  model = model,
  cores = 4,
  backend = "cmdstanr"
)

# Weak and strong targets against one shared set of lures, one criterion per
# subject: `strong` is 1 for strong targets and 0 for weak targets and lures
# (the lure rows' value does not matter, their false-alarm rate depends on
# the criterion alone)
lure_dat <- expand.grid(id = 1:20, cell = c("lure", "weak", "strong"))
lure_dat$stimulus <- as.integer(lure_dat$cell != "lure")
lure_dat$strong <- as.integer(lure_dat$cell == "strong")
lure_dat$n_trials <- 100L
lure_dat$n_old <- rsdt_yn(nrow(lure_dat), lure_dat$n_trials,
                          lure_dat$stimulus, d = 1 + 0.8 * lure_dat$strong,
                          criterion = 0.7)
fit_strength <- bmm(
```

```
  formula = bmf(d ~ 1 + strong + (1 | id), criterion ~ 1 + (1 | id)),
  data = lure_dat,
  model = model,
  cores = 4,
  backend = "cmdstanr"
)

# Simulating from a fitted unequal-variance model: the model reports
# sdratio on its log link, rsdt_yn() takes the ratio itself.
sdratio_posterior <- 0.375
rsdt_yn(2, 100L, c(0L, 1L),
        d = 1.3, criterion = 0.75, sdratio = exp(sdratio_posterior))

## End(Not run)
```

---

sdt_yn_dist *Distribution functions for Yes/No SDT*

Description

Density and random generation for the yes/no signal detection theory model, where the response is the number of "old"/"signal" responses out of a fixed number of trials (a binomial likelihood).

Usage

```
dsdt_yn(
  n_old,
  n_trials,
  stimulus,
  d,
  criterion = NULL,
  bias = NULL,
  sdratio = 1,
  dist = c("normal", "gumbel_min", "gumbel_max", "logistic"),
  log = FALSE
)

rsdt_yn(
  n,
  n_trials,
  stimulus,
  d,
  criterion = NULL,
  bias = NULL,
  sdratio = 1,
  dist = c("normal", "gumbel_min", "gumbel_max", "logistic")
)
```

Arguments

n_old	Integer vector. Number of "old"/"signal" responses.
n_trials	Integer vector. Total number of trials per cell.
stimulus	Numeric or logical vector (0/1). Stimulus type: 0 = noise, 1 = signal.
d	Numeric. Sensitivity: d' when sdratio is 1, and otherwise the balanced index d_a, the separation between the two distributions divided by the root-mean-square of their SDs (see sdt_yn()). The separation in noise units is d * sqrt((1 + sdratio^2) / 2).
criterion	Numeric. Location of the decision criterion, measured from the mean of the noise distribution in units of its SD. Give either criterion or bias, not both.
bias	Numeric. Location of the decision criterion measured from the midpoint between the noise and signal means instead, in noise-SD units: criterion - d * sqrt((1 + sdratio^2) / 2) / 2. 0 is the neutral point, positive values a bias toward "new"/"noise".
sdratio	Numeric. Ratio of signal to noise standard deviations (default 1, i.e., equal variance). Must be positive, and is on the **natural** scale — see the section below before reusing a fitted value.
dist	Character. The distribution assumed for the latent evidence, given here by its cumulative distribution function: • "normal" (default): Gaussian, $\Phi(x)$ • "gumbel_min": smallest extreme value, $1 - \exp(-\exp(x))$ (the complementary log-log distribution) • "gumbel_max": largest extreme value, $\exp(-\exp(-x))$ (the log-log distribution, as in evd::pgumbel) • "logistic": $1/(1 + \exp(-x))$
log	Logical. If TRUE, returns log-density (default FALSE).
n	Integer. Number of observations to generate. n_trials, stimulus, and the model parameters are recycled to this length.

Value

dsdt_yn returns the (log-)density (binomial probability). rsdt_yn returns an integer vector with the number of "old"/"signal" responses per observation.

Parameter scales

As everywhere in bmm, these functions take their arguments on the **natural** scale, while the parameters sdt_yn() *estimates* are on their link scale. d and criterion have identity links and carry across unchanged, but sdratio has a log link: a fitted sdratio of 0.375 is a ratio of exp(0.375) = 1.455, and passing 0.375 here instead asks for a signal distribution 2.7 times narrower than the noise. That is a legal value and raises no error, so exponentiate first.

References

Green, D. M., & Swets, J. A. (1966). *Signal detection theory and psychophysics*. Wiley.

Examples

```
# Density of yes/no SDT data
dsdt_yn(n_old = 80, n_trials = 100, stimulus = 1,
        d = 1.5, criterion = 0.9)

# The same observer, with the criterion given relative to the midpoint
dsdt_yn(n_old = 80, n_trials = 100, stimulus = 1,
        d = 1.5, bias = 0.15)

# Vectorized over observations
dsdt_yn(n_old = c(30, 80), n_trials = c(100, 100),
        stimulus = c(0, 1), d = 1.5, criterion = 0.9,
        log = TRUE)

# Unequal variance from a fitted model: sdt_yn() reports sdratio on its log
# link, so exponentiate before passing it here
dsdt_yn(n_old = 80, n_trials = 100, stimulus = 1,
        d = 1.5, criterion = 0.9, sdratio = exp(0.375))
# Generate yes/no SDT data for a design
dat <- expand.grid(id = 1:20, stimulus = c(0L, 1L))
dat$n_trials <- 100L
dat$n_old <- rsdt_yn(nrow(dat), dat$n_trials, dat$stimulus,
                     d = 1.5, criterion = 0.9)
head(dat)
```

SDTdist *Utility functions for Signal Detection Theory*

Description

Compute sensitivity, criterion and bias from hit and false alarm rates for different SDT distribution families. A single (hit, false alarm) pair cannot identify the signal-to-noise SD ratio, so all three are the equal-variance values; see `sdt_yn()` for the unequal-variance model.

Usage

```
sdt_d(
  hit_rate,
  fa_rate,
  dist = c("normal", "gumbel_min", "gumbel_max", "logistic")
)

sdt_criterion(
  hit_rate,
  fa_rate,
  dist = c("normal", "gumbel_min", "gumbel_max", "logistic")
)
```

```
sdt_bias(
  hit_rate,
  fa_rate,
  dist = c("normal", "gumbel_min", "gumbel_max", "logistic")
)
```

Arguments

`hit_rate`	Numeric. Proportion of hits (P("old" | signal)).
`fa_rate`	Numeric. Proportion of false alarms (P("old" | noise)).
`dist`	Character. The distribution assumed for the latent evidence, given here by its cumulative distribution function: • "normal" (default): Gaussian, $\Phi(x)$ • "gumbel_min": smallest extreme value, $1 - \exp(-\exp(x))$ (the complementary log-log distribution) • "gumbel_max": largest extreme value, $\exp(-\exp(-x))$ (the log-log distribution, as in `evd::pgumbel`) • "logistic": $1/(1 + \exp(-x))$

Value

`sdt_d` returns the distance between the signal and noise distributions on the latent evidence axis, obtained by inverting the decision rule "respond old when the evidence exceeds the criterion": $Q(1 - FA) - Q(1 - H)$, where Q is the quantile function of `dist`. For `dist = "normal"` this reduces to the familiar $d' = \Phi^{-1}(H) - \Phi^{-1}(FA)$. Because one operating point implies equal variance, this matches the `d` parameter of `sdt_yn()` whenever `sdratio` is at its default.

`sdt_criterion` returns the location of the decision criterion measured from the mean of the noise distribution, in noise-SD units, as the `criterion` parameter of `sdt_yn()` is: $Q(1-FA)$. It depends on the false-alarm rate alone. For `dist = "normal"` this is $-\Phi^{-1}(FA)$.

`sdt_bias` returns the criterion measured from the midpoint between the noise and signal means instead, as the `bias` parameter of `sdt_yn()` is: $(Q(1-FA)+Q(1-H))/2$, which is `sdt_criterion() - sdt_d() / 2`. For `dist = "normal"` this is the familiar $c = -(\Phi^{-1}(H) + \Phi^{-1}(FA))/2$. Positive values are a conservative bias, toward "new"/"noise" responses.

References

Green, D. M., & Swets, J. A. (1966). *Signal detection theory and psychophysics*. Wiley.

See Also

`sdt_yn()`, whose `d`, `criterion` and `bias` parameters these functions compute in closed form from observed rates: `sdt_d()`, `sdt_criterion()` and `sdt_bias()` return the same quantities as the parameters of the same name whenever `sdratio` is at its default.

Examples

```
# Compute d from hit and false alarm rates (Gaussian SDT)
sdt_d(hit_rate = 0.8, fa_rate = 0.2, dist = "normal")

# The extreme-value analogue
sdt_d(hit_rate = 0.75, fa_rate = 0.25, dist = "gumbel_min")
# Compute criterion from hit and false alarm rates
sdt_criterion(hit_rate = 0.8, fa_rate = 0.2, dist = "normal")
# Compute the bias relative to the midpoint
sdt_bias(hit_rate = 0.8, fa_rate = 0.2, dist = "normal")
```

---

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 `bmm_models()` to see the list of supported models, and `?modelname` (for example `?imm`) for 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

`bmm_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 `bmm_models()` to see the list of supported models, and `?modelname` (for example `?imm`) for 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

`bmm_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)
```

---

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,
  file = NULL,
  file_compress = TRUE,
  ...
)
```

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. Rows of fit$data are accepted as they are, which is how brms::kfold() and brms::reloo() pass them.
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.
file	Either NULL or a character string. If a string, the updated model is saved via saveRDS in a file named after the string, as in bmm(). update() never writes to the file the original fit was read from: pass file explicitly to save the updated fit.
file_compress	Logical or a character string, specifying one of the compression algorithms supported by saveRDS when saving the updated model object.
...	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. If file is given, it names the newly written file. If not, the file field is carried over from the original fit, so the updated object still points at the file it came from – but that file is *not* rewritten: it still holds the fit as it was before this update. Pass file to save the updated fit.

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

∗ **bmmodel**
cswald, 28
ddm, 33
ezdm, 41
imm, 53
m3, 64
mixture2p, 71
mixture3p, 74
mpt, 79
mpt_from_eqn, 87
mpt_from_string, 90
mpt_tree, 94
plot.mpt_tree, 109
sdm, 124
sdt_cdp, 128
sdt_mafc, 137
sdt_ranking, 143
sdt_rating, 148
sdt_yn, 162
∗ **dataset**
broeder_schuetz_2009_e3, 21
data_color_judgement_task, 33
meyer_grant_jakob_2025, 70
oberauer_lewandowsky_2019_e1, 104
oberauer_lin_2017, 105
zhang_luck_2008, 182
∗ **deprecated**
bmm, 9
imm, 53
sdm, 124
∗ **developer**
apply_links, 7
native_transform, 102
∗ **distribution**
cswald_dist, 30
ddm_dist, 36
ezdm_dist, 44
IMMdist, 58
m3dist, 68
mixture2p_dist, 73
mixture3p_dist, 77
mptdist, 96
rejection_sampling, 119
SDMdist, 126
sdt_cdp_dist, 133
sdt_dpsdt_dist, 135
sdt_mafc_dist, 140
sdt_metad_dist, 141
sdt_ranking_dist, 146
sdt_rating_dist, 156
sdt_yn_dist, 169
SDTdist, 171
∗ **extract_info**
bmm_data_check, 13
default_prior.bmmformula, 37
extract_parameter_dimensions, 39
extract_stan_blocks, 40
fit_info, 50
model_citation, 78
native_parameters, 99
parameter_info, 105
pp_check_vars, 114
prior_info, 118
stancode.bmmformula, 174
standata.bmmformula, 175
∗ **transform**
adjust_ezdm_accuracy, 5
aggregate_sdt_cdp_data, 6
c_parametrizations, 22
calc_error_relative_to_nontargets, 23
circle_transform, 23
construct_m3_act_funs, 27
ezdm_summary_stats, 47
flag_contaminant_rts, 51
k2sd, 61
mpt_data_reshape, 84
restructure.bmmfit, 120

softmax, 173
validate_fast_guesses, 179
wrap, 181

adjust_ezdm_accuracy, 5
aggregate_sdt_cdp_data, 6
aggregate_sdt_cdp_data(), *129*, *131*
apply_links, 7
auc_sdt, 8
auc_sdt(), *107*, *109*, *124*, *159*, *160*

bmf *(*bmmformula*)*, 19
bmm, 9, *106*
bmm(), *6*, *8*, *10*, *13*, *14*, *17–19*, *24*, *37*, *38*, *49*, *62*, *78*, *84–86*, *89*, *92*, *97*, *98*, *100*, *113*, *115*, *116*, *118*, *122*, *123*, *131*, *159–161*, *178*
bmm-package, 4
bmm_data_check, 13
bmm_data_check(), *19*, *85*, *86*, *89*
bmm_models, 15
bmm_models(), *10*, *13*, *14*, *37*, *38*, *174*, *175*
bmm_options, 16
bmm_options(), *11*
bmm_setup, 18
bmmformula, 19
bmmformula(), *13*, *79*, *89*, *92*, *178*
brms::brm(), *11*, *13*
brms::brmsformula(), *20*
brms::conditional_effects(), *24–26*
brms::default_prior(), *37*, *38*
brms::fitted.brmsfit(), *112*
brms::kfold(), *178*
brms::posterior_epred(), *83*, *101*, *112*, *116*
brms::posterior_linpred(), *8*, *62*, *98*, *100*, *103*, *123*, *159*, *161*
brms::posterior_predict(), *83*, *111*, *113*, *114*, *116*
brms::pp_check(), *112–114*
brms::prepare_predictions(), *114*
brms::reloo(), *178*
brms::set_prior(), *10*, *37*, *174*
brms::stancode(), *174*
brms::standata(), *175*
brms::update.brmsfit(), *177*, *178*
broeder_schuetz_2009_e3, 21, *63*, *123*, *165*

c_bessel2sqrtexp *(*c_parametrizations*)*, 22
c_parametrizations, 22
c_sqrtexp2bessel *(*c_parametrizations*)*, 22
calc_error_relative_to_nontargets, 23
check_data(), *13*, *14*
check_formula(), *13*, *14*
circle_transform, 23
citation(), *79*
conditional_effects *(*conditional_effects.bmmfit*)*, 24
conditional_effects.bmmfit, 24
conditional_effects.bmmfit(), *102*
construct_m3_act_funs, 27
cswald, 28
cswald_dist, 30

data_check_findings(), *13*, *14*
data_color_judgement_task, 33
dcswald *(*cswald_dist*)*, 30
dcswald(), *30*
dddm *(*ddm_dist*)*, 36
ddm, 33
ddm(), *28*, *114*
ddm_dist, 36
default_prior *(*default_prior.bmmformula*)*, 37
default_prior(), *10*, *13*, *81–83*, *119*, *174*
default_prior.bmmformula, 37
deg2rad *(*circle_transform*)*, 23
dezdm *(*ezdm_dist*)*, 44
dimm *(*IMMdist*)*, 58
dm3 *(*m3dist*)*, 68
dm3(), *97*
dmixture2p *(*mixture2p_dist*)*, 73
dmixture3p *(*mixture3p_dist*)*, 77
dmpt *(*mptdist*)*, 96
dplyr::group_by(), *49*
dplyr::reframe(), *49*
dsdm *(*SDMdist*)*, 126
dsdt_cdp *(*sdt_cdp_dist*)*, 133
dsdt_cdp(), *131*
dsdt_dpsdt *(*sdt_dpsdt_dist*)*, 135
dsdt_mafc *(*sdt_mafc_dist*)*, 140
dsdt_metad *(*sdt_metad_dist*)*, 141
dsdt_ranking *(*sdt_ranking_dist*)*, 146
dsdt_rating *(*sdt_rating_dist*)*, 156
dsdt_yn *(*sdt_yn_dist*)*, 169
dsdt_yn(), *162*, *166*

emm_basis.bmmfit *(emmeans-bmmfit)*, 38
emmeans-bmmfit, 38
emmeans::emm_basis(), *39*
emmeans::emmeans(), *38, 39*
emmeans::recover_data(), *39*
extract_parameter_dimensions, 39
extract_stan_blocks, 40
ezdm, 41
ezdm(), *28*, *50*, *113, 114*
ezdm_dist, 44
ezdm_summary_stats, 47
ezdm_summary_stats(), *5*, *52*

fit_info, 50
fit_model *(bmm)*, 9
fitted.bmmfit *(posterior_epred.bmmfit)*, 111
flag_contaminant_rts, 51
flag_contaminant_rts(), *50*, *180, 181*

imm, 53
IMMabc *(imm)*, 53
IMMbsc *(imm)*, 53
IMMdist, 58
IMMfull *(imm)*, 53

k2sd, 61

latent_sdt, 61
latent_sdt(), *99*, *107*, *150*, *155*, *160, 161*

m3, 64
m3(), *25*, *115*, *117*
m3dist, 68
merge(), *85*
meyer_grant_jakob_2025, 70
mixture2p, 71
mixture2p(), *18*
mixture2p_dist, 73
mixture3p, 74
mixture3p(), *25*
mixture3p_dist, 77
model_citation, 78
mpt, 79
mpt(), *84–89*, *91, 92*, *94*, *96, 97*, *109, 110*, *113*, *115–117*
mpt_data_reshape, 84
mpt_data_reshape(), *81*, *89*, *92*
mpt_from_eqn, 87
mpt_from_eqn(), *85, 86*, *91, 92*
mpt_from_string, 90
mpt_from_string(), *85–87*, *89*
mpt_tree, 94
mpt_tree(), *79*, *81*, *85*, *88, 89*, *92*, *110*
mptdist, 96
mratio, 98
mratio(), *153*

native_parameters, 99
native_parameters(), *82, 83*, *102*, *104*, *111*, *112*
native_transform, 102
native_transform(), *102*

oberauer_lewandowsky_2019_e1, *67*, 104
oberauer_lin_2017, 105
options, *11*

parameter_info, 105
parameter_info(), *81*, *101, 102*, *119*
pcswald *(cswald_dist)*, 30
pimm *(IMMdist)*, 58
plot(), *26*
plot.bmm_sdt_auc, 106
plot.bmm_sdt_auc(), *9*
plot.bmm_sdt_latent, 107
plot.bmm_sdt_latent(), *63*
plot.bmm_sdt_roc, 108
plot.bmm_sdt_roc(), *121*, *124*
plot.mpt *(plot.mpt_tree)*, 109
plot.mpt_tree, 109
pmixture2p *(mixture2p_dist)*, 73
pmixture3p *(mixture3p_dist)*, 77
posterior_epred.bmmfit, 111
pp_check *(pp_check.bmmfit)*, 112
pp_check.bmmfit, 112
pp_check.bmmfit(), *114*, *115*, *117*
pp_check_vars, 114
pp_check_vars(), *113, 114*
ppp, 115
ppp(), *83*
prior_info, 118
prior_info(), *81*, *105*
psdm *(SDMdist)*, 126

qcswald *(cswald_dist)*, 30
qimm *(IMMdist)*, 58
qmixture2p *(mixture2p_dist)*, 73

qmixture3p (mixture3p_dist), 77
qsdm (SDMdist), 126

rad2deg (circle_transform), 23
rcswald (cswald_dist), 30
rcswald(), *30*
rddm (ddm_dist), 36
readLines(), *87*, *91*
recover_data.bmmfit (emmeans-bmmfit), 38
rejection_sampling, 119
restructure (restructure.bmmfit), 120
restructure.bmmfit, 120
rezdm (ezdm_dist), 44
rimm (IMMdist), 58
rm3 (m3dist), 68
rmixture2p (mixture2p_dist), 73
rmixture3p (mixture3p_dist), 77
rmpt (mptdist), 96
roc_observed, 121
roc_observed(), *108, 109*, *124*
roc_sdt, 122
roc_sdt(), *9*, *63*, *107–109*, *121*, *160, 161*
rsdm (SDMdist), 126
rsdt_cdp (sdt_cdp_dist), 133
rsdt_cdp(), *129*, *131*
rsdt_dpsdt (sdt_dpsdt_dist), 135
rsdt_mafc (sdt_mafc_dist), 140
rsdt_metad (sdt_metad_dist), 141
rsdt_ranking (sdt_ranking_dist), 146
rsdt_rating (sdt_rating_dist), 156
rsdt_rating(), *155*
rsdt_yn (sdt_yn_dist), 169
rsdt_yn(), *162*, *166*
rtdists::pdiffusion(), *30*, *32*
rtdists::qdiffusion(), *30*, *32*
rtdists::rdiffusion(), *30*, *32*

saveRDS, *11*, *178*
sdm, 124
sdm(), *25*
SDMdist, 126
sdmSimple (sdm), 124
sdt_bias (SDTdist), 171
sdt_bias(), *165*, *167*
sdt_cdp, 128
sdt_cdp(), *6*, *115*, *133–135*
sdt_cdp_dist, 133
sdt_criterion (SDTdist), 171
sdt_criterion(), *165*, *167*
sdt_d (SDTdist), 171
sdt_d(), *167*
sdt_dpsdt_dist, 135
sdt_mafc, 137
sdt_mafc(), *63*, *107*, *113*, *115*, *122*, *159*
sdt_mafc_dist, 140
sdt_metad_dist, 141
sdt_ranking, 143
sdt_ranking(), *63*, *70*, *107*, *113*, *115*, *117*, *122*, *147*
sdt_ranking_dist, 146
sdt_rating, 148
sdt_rating(), *61*, *63*, *98, 99*, *107*, *113*, *115*, *121, 122*, *129*, *131–133*, *135, 136*, *141, 142*, *157*, *160*
sdt_rating_dist, 156
sdt_sensitivity, 158
sdt_sensitivity(), *98*, *164*
sdt_thresholds, 160
sdt_thresholds(), *155*
sdt_yn, 162
sdt_yn(), *21*, *61*, *63*, *107*, *113*, *115*, *121, 122*, *132*, *136*, *139, 140*, *142*, *145*, *147*, *154, 155*, *157*, *170–172*
sdt_yn_dist, 169
SDTdist, 171
softmax, 173
softmaxinv (softmax), 173
stancode (stancode.bmmformula), 174
stancode(), *11*, *13*
stancode.bmmformula, 174
standata (standata.bmmformula), 175
standata(), *11*, *13*
standata.bmmformula, 175
stats::D(), *82*
stats::dmultinom(), *97*
summary.bmmfit, 176
summary.bmmfit(), *100*
summary.brmsfit, *177*

update.bmmfit, 177

validate_fast_guesses, 179
validate_fast_guesses(), *52*

wrap, 181

zhang_luck_2008, 182