Package ‘EMC2’

January 20, 2025

Title Bayesian Hierarchical Analysis of Cognitive Models of Choice

Version 2.1.0

Description Fit Bayesian (hierarchical) cognitive models using a linear modeling language interface using particle metropolis Markov chain Monte Carlo sampling with Gibbs steps. The diffusion decision model (DDM), linear ballistic accumulator model (LBA), racing diffusion model (RDM), and the lognormal race model (LNR) are supported. Additionally, users can specify their own likelihood function and/or choose for non-hierarchical estimation, as well as for a diagonal, blocked or full multivariate normal group-level distribution to test individual differences. Prior specification is facilitated through methods that visualize the (implied) prior. A wide range of plotting functions assist in assessing model convergence and posterior inference. Models can be easily evaluated using functions that plot posterior predictions or using relative model comparison metrics such as information criteria or Bayes factors. References: Stevenson et al. (2024) <doi:10.31234/osf.io/2e4dq>.

License GPL (>= 3)

URL `https://ampl-psych.github.io/EMC2/`, `https://github.com/ampl-psych/EMC2`

BugReports `https://github.com/ampl-psych/EMC2/issues`

Encoding UTF-8

RoxygenNote 7.3.2

VignetteBuilder knitr

Suggests testthat (>= 3.0.0), vdiffr, knitr, rmarkdown

Config/testthat/edition 3

Imports abind, coda, corpcor, graphics, grDevices, magic, MASS, matrixcalc, methods, msm, mvtnorm, parallel, stats, Matrix, Rcpp, Brobdingnag, corrplot, colorspace, psych, utils, lpSolve, WienR

LinkingTo Rcpp

Depends R (>= 3.5.0)

LazyData true

Config/testthat/parallel true

NeedsCompilation yes

Author Niek Stevenson [aut, cre] (<`https://orcid.org/0000-0003-3206-7544`>),
Michelle Donzallaz [aut],
Andrew Heathcote [aut],
Steven Miletić [ctb],
Raphael Hartmann [ctb],
Karl C. Klauer [ctb],
Steven G. Johnson [ctb],
Jean M. Linhart [ctb],
Brian Gough [ctb],
Gerard Jungman [ctb],
Rudolf Schuerer [ctb],
Przemyslaw Sliwa [ctb],
Jason H. Stover [ctb]

Maintainer Niek Stevenson <`niek.stevenson@gmail.com`>

Repository CRAN

Date/Publication 2024-10-14 20:10:06 UTC

Contents

`chain_n` *chain_n()*

Description

Returns a matrix with the number of samples per chain for each stage that is present in the emc object (i.e., `preburn`, `burn`, `adapt`, `sample`). The number of rows of the matrix reflects the number of chains and the number of columns the number of sampling stages.

Usage

```
chain_n(emc)
```

Arguments

`emc`	A list, the output of `fit()`.

Value

A matrix

Examples

```
chain_n(samples_LNR)
```

`check.emc` *Convergence checks for an emc object*

Description

Runs a series of convergence checks, prints statistics to the console, and makes traceplots of the worst converged parameter per selection.

Usage

```
## S3 method for class 'emc'
check(
  emc,
  selection = c("mu", "sigma2", "alpha"),
  digits = 3,
  plot_worst = TRUE,
  ...
)

check(emc, ...)
```

Arguments

`emc`	An emc object
`selection`	A Character vector. Indicates which parameter types to check (e.g., `alpha`, `mu`, `sigma2`, `correlation`).
`digits`	Integer. How many digits to round the ESS and Rhat to in the plots
`plot_worst`	Boolean. If `TRUE` also plots the chain plots for the worst parameter
`...`	Optional arguments that can be passed to `get_pars` or `plot.default` (see `par()`)

Details

Note that the `Rhat` is calculated by doubling the number of chains by first splitting chains into first and second half, so it also a test of stationarity.

Efficiency of sampling is indicated by the effective sample size (ESS) (from the `coda` R package). Full range of possible samples manipulations described in `get_pars`.

Value

a list with the statistics for the worst converged parameter per selection

Examples

```
check(samples_LNR)
```

`compare` *Information criteria and marginal likelihoods*

Description

Returns the BPIC/DIC or marginal deviance (-2*marginal likelihood) for a list of samples objects.

Usage

```
compare(
  sList,
  stage = "sample",
  filter = NULL,
  use_best_fit = TRUE,
  BayesFactor = TRUE,
  cores_for_props = 4,
  cores_per_prop = 1,
  print_summary = TRUE,
  digits = 0,
  digits_p = 3,
  ...
)
```

Arguments

`sList`	List of samples objects
`stage`	A string. Specifies which stage the samples are to be taken from `"preburn"`, `"burn"`, `"adapt"`, or `"sample"`
`filter`	An integer or vector. If it's an integer, iterations up until the value set by `filter` will be excluded. If a vector is supplied, only the iterations in the vector will be considered.
`use_best_fit`	Boolean, defaults to `TRUE`, uses the minimal or mean likelihood (whichever is better) in the calculation, otherwise always uses the mean likelihood.
`BayesFactor`	Boolean, defaults to `TRUE`. Include marginal likelihoods as estimated using WARP-III bridge sampling. Usually takes a minute per model added to calculate
`cores_for_props`	Integer, how many cores to use for the Bayes factor calculation, here 4 is the default for the 4 different proposal densities to evaluate, only 1, 2 and 4 are sensible.
`cores_per_prop`	Integer, how many cores to use for the Bayes factor calculation if you have more than 4 cores available. Cores used will be cores_for_props * cores_per_prop. Best to prioritize cores_for_props being 4 or 2
`print_summary`	Boolean (default `TRUE`), print table of results
`digits`	Integer, significant digits in printed table for information criteria
`digits_p`	Integer, significant digits in printed table for model weights
`...`	Additional, optional arguments

Value

Matrix of effective number of parameters, mean deviance, deviance of mean, DIC, BPIC, Marginal Deviance (if `BayesFactor=TRUE`) and associated weights.

Examples

```
## Not run:
# Define a list of two (or more different models)
# Here the full model is an emc object with the hypothesized effect
# The null model is an emc object without the hypothesized effect
design_full <- design(data = forstmann,model=DDM,
                           formula =list(v~0+S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
                           constants=c(s=log(1)))
# Now without a ~ E
design_null <- design(data = forstmann,model=DDM,
                           formula =list(v~0+S,a~1, t0~1, s~1, Z~1, sv~1, SZ~1),
                           constants=c(s=log(1)))

full_model <- make_emc(forstmann, design_full)
full_model <- fit(full_model)

null_model <- make_emc(forstmann, design_null)
null_model <- fit(null_model)
sList <- list(full_model, null_model)
# By default emc uses 4 cores to parallelize marginal likelihood estimation across proposals
# So cores_per_prop = 3 results in 12 cores used.
compare(sList, cores_per_prop = 3)

## End(Not run)
```

`compare_subject` *Information criteria for each participant*

Description

Returns the BPIC/DIC based model weights for each participant in a list of samples objects

Usage

```
compare_subject(
  sList,
  stage = "sample",
  filter = 0,
  use_best_fit = TRUE,
  print_summary = TRUE,
  digits = 3
)
```

Arguments

`sList`	List of samples objects
`stage`	A string. Specifies which stage the samples are to be taken from `"preburn"`, `"burn"`, `"adapt"`, or `"sample"`
`filter`	An integer or vector. If it's an integer, iterations up until the value set by `filter` will be excluded. If a vector is supplied, only the iterations in the vector will be considered.
`use_best_fit`	Boolean, defaults to `TRUE`, use minimal likelihood or mean likelihood (whichever is better) in the calculation, otherwise always uses the mean likelihood.
`print_summary`	Boolean (defaults to `TRUE`) print table of results
`digits`	Integer, significant digits in printed table

Value

List of matrices for each subject of effective number of parameters, mean deviance, deviance of mean, DIC, BPIC and associated weights.

Examples

```
## Not run:
# Define a list of two (or more different models)
# Here the full model is an emc object with the hypothesized effect
# The null model is an emc object without the hypothesized effect
design_full <- design(data = forstmann,model=DDM,
                           formula =list(v~0+S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
                           constants=c(s=log(1)))
# Now without a ~ E
design_null <- design(data = forstmann,model=DDM,
                           formula =list(v~0+S,a~1, t0~1, s~1, Z~1, sv~1, SZ~1),
                           constants=c(s=log(1)))

full_model <- make_emc(forstmann, design_full)
full_model <- fit(full_model, cores_for_chains = 1)

null_model <- make_emc(forstmann, design_null, cores_for_chains = 1)
null_model <- fit(null_model)
sList <- list(full_model, null_model)
compare_subject(sList)
# prints a set of weights for each model for the different participants
# And returns the DIC and BPIC for each participant for each model.

## End(Not run)
```

`contr.anova` *Anova style contrast matrix*

Description

Similar to `contr.helmert`, but then scaled to estimate differences between conditions. Use in `design()`.

Usage

```
contr.anova(n)
```

Arguments

`n`	An integer. The number of items for which to create the contrast

Value

A contrast matrix.

Examples

```
{
design_DDMaE <- design(data = forstmann,model=DDM, contrasts = list(E = contr.anova),
formula =list(v~S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
constants=c(s=log(1)))
}
```

`contr.bayes` *Contrast to enforce equal prior variance on each level*

Description

Typical contrasts impose different levels of marginal prior variance for the different levels. This contrast can be used to ensure that each level has equal marginal priors (Rouder, Morey, Speckman, & Province; 2012).

Usage

```
contr.bayes(n)
```

Arguments

`n`	An integer. The number of items for which to create the contrast

Value

A contrast matrix.

Examples

```
{
design_DDMaE <- design(data = forstmann,model=DDM, contrasts = list(E = contr.bayes),
formula =list(v~S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
constants=c(s=log(1)))
}
```

`contr.decreasing` *Contrast to enforce decreasing estimates*

Description

Each level will be estimated as a reduction from the previous level

Usage

```
contr.decreasing(n)
```

Arguments

`n`	an integer. The number of items for which to create the contrast.

Value

a contrast matrix.

Examples

```
{
design_DDMaE <- design(data = forstmann,model=DDM, contrasts = list(E = contr.decreasing),
formula =list(v~S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
constants=c(s=log(1)))
}
```

`contr.increasing` *Contrast to enforce increasing estimates*

Description

Each level will be estimated additively from the previous level

Usage

```
contr.increasing(n)
```

Arguments

`n`	an integer. The number of items for which to create the contrast.

Value

a contrast matrix.

Examples

```
{
design_DDMaE <- design(data = forstmann,model=DDM, contrasts = list(E = contr.increasing),
formula =list(v~S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
constants=c(s=log(1)))
}
```

`credible.emc` *Posterior credible interval tests*

Description

Modeled after `t.test`, returns the credible interval of the parameter or test and what proportion of the posterior distribution (or the difference in posterior distributions in case of a two sample test) overlaps with mu. For a one sample test provide `x` and for two sample also provide `y`. Note that for comparisons within one model, we recommend using `hypothesis()` if the priors were well chosen.

Usage

```
## S3 method for class 'emc'
credible(
  x,
  x_name = NULL,
  x_fun = NULL,
  x_fun_name = "fun",
  selection = "mu",
```

```
  y = NULL,
  y_name = NULL,
  y_fun = NULL,
  y_fun_name = "fun",
  x_subject = NULL,
  y_subject = NULL,
  mu = 0,
  alternative = c("less", "greater")[1],
  probs = c(0.025, 0.5, 0.975),
  digits = 2,
  p_digits = 3,
  print_table = TRUE,
  ...
)

credible(x, ...)
```

Arguments

`x`	An emc object
`x_name`	A character string. Name of the parameter to be tested for `x`
`x_fun`	Function applied to the MCMC chains to create variable to be tested.
`x_fun_name`	Name to give to quantity calculated by `x_fun`
`selection`	A character string designating parameter type (e.g. `alpha` or `covariance`)
`y`	A second emc object
`y_name`	A character string. Name of the parameter to be tested for `y`
`y_fun`	Function applied to the MCMC chains to create variable to be tested.
`y_fun_name`	Name to give to quantity calculated by `y_fun`
`x_subject`	Integer or name selecting a subject
`y_subject`	Integer or name selecting a subject
`mu`	Numeric. `NULL` value for single sample test if `y` is not supplied (default 0)
`alternative`	`less` or `greater` determining direction of test probability
`probs`	Vector defining quantiles to return.
`digits`	Integer, significant digits for estimates in printed results
`p_digits`	Integer, significant digits for probability in printed results
`print_table`	Boolean (defaults to `TRUE`) for printing results table
`...`	Additional optional arguments that can be passed to `get_pars`

Value

Invisible results table with no rounding.

Examples

```
## Not run:
# Run a credible interval test (Bayesian ''t-test'')
# Here the full model is an emc object with the hypothesized effect
design_full <- design(data = forstmann,model=DDM,
                           formula =list(v~0+S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
                           constants=c(s=log(1)))

full_model <- make_emc(forstmann, design_full)
full_model <- fit(full_model)
credible(full_model, x_name = "v")
# We can also compare between two sets of emc objects
# Now without a ~ E
design_null <- design(data = forstmann,model=DDM,
                           formula =list(v~0+S,a~1, t0~1, s~1, Z~1, sv~1, SZ~1),
                           constants=c(s=log(1)))

null_model <- make_emc(forstmann, design_null)
null_model <- fit(null_model)
credible(x = null_model, x_name = "a", y = full_model, y_name = "a")

# Or provide custom functions
credible(x = full_model, x_fun = function(d) d["a_Eaccuracy"] - d["a_Eneutral"])

## End(Not run)
```

`DDM` *The Diffusion Decision Model*

Description

Model file to estimate the Diffusion Decision Model (DDM) in EMC2.

Usage

```
DDM()
```

Details

Model files are almost exclusively used in `design()`.

Default values are used for all parameters that are not explicitly listed in the `formula` argument of `design()`.They can also be accessed with `DDM()$p_types`.

Parameter	**Transform**	**Natural scale**	**Default**	**Mapping**	**Interpretation**
v	-	[-Inf, Inf]	1		Mean evidence-accumulation
a	log	[0, Inf]	log(1)		Boundary separation
t0	log	[0, Inf]	log(0)		Non-decision time
s	log	[0, Inf]	log(1)		Within-trial standard deviatio

Z	probit	[0, 1]	qnorm(0.5)	*z* = *Z* x *a*	Relative start point (bias)
SZ	probit	[0, 1]	qnorm(0)	*sz* = 2 x *SZ* x min(*a* x *Z*, *a* x (1-*Z*))	Relative between-trial variati
sv	log	[0, Inf]	log(0)		Between-trial standard deviat
st0	log	[0, Inf]	log(0)		Between-trial variation (rang

`a`, `t0`, `sv`, `st0`, `s` are sampled on the log scale because these parameters are strictly positive, `Z`, `SZ` and `DP` are sampled on the probit scale because they should be strictly between 0 and 1.

`Z` is estimated as the ratio of bias to one boundary where 0.5 means no bias. `DP` comprises the difference in non-decision time for each response option.

Conventionally, `sv` is fixed to 1 to satisfy scaling constraints.

See Ratcliff, R., & McKoon, G. (2008). The diffusion decision model: theory and data for two-choice decision tasks. *Neural computation, 20*(4), 873-922. doi:10.1162/neco.2008.12-06-420.

Value

A model list with all the necessary functions for EMC2 to sample

Examples

```
design_DDMaE <- design(data = forstmann,model=DDM,
                           formula =list(v~0+S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
                           constants=c(s=log(1)))
# For all parameters that are not defined in the formula, default values are assumed
# (see Table above).
```

---

design *Specify a design and model*

Description

This function combines information regarding the data, type of model, and the model specification.

Usage

```
design(
  formula = NULL,
  factors = NULL,
  Rlevels = NULL,
  model,
  data = NULL,
  contrasts = NULL,
  matchfun = NULL,
  constants = NULL,
  covariates = NULL,
```

```
  functions = NULL,
  report_p_vector = TRUE,
  custom_p_vector = NULL,
  ...
)
```

Arguments

`formula`	A list. Contains the design formulae in the format `list(y ~ x, a ~ z)`.
`factors`	A named list containing all the factor variables that span the design cells and that should be taken into account by the model. The name `subjects` must be used to indicate the participant factor variable, also in the data. Example: `list(subjects=levels(dat$subjects), condition=levels(dat$condition))`
`Rlevels`	A character vector. Contains the response factor levels. Example: `c("right", "left")`
`model`	A function, specifies the model type. Choose from the drift diffusion model (`DDM()`, `DDMt0natural()`), the log-normal race model (`LNR()`), the linear ballistic model (`LBA()`), the racing diffusion model (`RDM()`, `RDMt0natural()`), or define your own model functions.
`data`	A data frame. `data` can be used to automatically detect `factors`, `Rlevels` and `covariates` in a dataset. The variable `R` needs to be a factor variable indicating the response variable. Any numeric column except `trials` and `rt` are treated as covariates, and all remaining factor variables are internally used in `factors`.
`contrasts`	Optional. A named list specifying a design matrix. Example for supplying a customized design matrix: `list(lM = matrix(c(-1/2,1/2),ncol=1,dimnames=list(NULL,"diff"))))`
`matchfun`	A function. Only needed for race models. Specifies whether a response was correct or not. Example: `function(d)d$S==d$lR` where lR refers to the latent response factor.
`constants`	A named vector that sets constants. Any parameter in `sampled_p_vector` can be set constant.
`covariates`	Names of numeric covariates.
`functions`	List of functions to create new factors based on those in the factors argument. These new factors can then be used in `formula`.
`report_p_vector`	Boolean. If TRUE (default), it returns the vector of parameters to be estimated.
`custom_p_vector`	A character vector. If specified, a custom likelihood function can be supplied.
`...`	Additional, optional arguments

Value

A design list.

Examples

```
# load example dataset
dat <- forstmann

# create a function that takes the latent response (lR) factor (d) and returns a logical
# defining the correct response for each stimulus. Here the match is simply
# such that the S factor equals the latent response factor
matchfun <- function(d)d$S==d$lR

# When working with lM and lR, it can be useful to design  an
# "average and difference" contrast matrix. For binary responses, it has a
# simple canonical form
ADmat <- matrix(c(-1/2,1/2),ncol=1,dimnames=list(NULL,"diff"))

# Create a design for a linear ballistic accumulator model (LBA) that allows
# thresholds to be a function of E and lR. The final result is a 9 parameter model.
design_LBABE <- design(data = dat,model=LBA,matchfun=matchfun,
                            formula=list(v~lM,sv~lM,B~E+lR,A~1,t0~1),
                            contrasts=list(v=list(lM=ADmat)),
                            constants=c(sv=log(1)))
```

`ess_summary.emc` *Effective sample size*

Description

Returns the effective sample size (ESS) of the selected parameter type. Full range of possible samples manipulations described in `get_pars`.

Usage

```
## S3 method for class 'emc'
ess_summary(
  emc,
  selection = "mu",
  stat = "min",
  stat_only = FALSE,
  digits = 1,
  ...
)

ess_summary(emc, ...)
```

Arguments

`emc`	An emc object
`selection`	A Character vector. Indicates which parameter types to check (e.g., `alpha`, `mu`, `sigma2`, `correlation`).

`stat`	A string. Should correspond to a function that can be applied to a vector, which will be performed on the vector/rows or columns of the matrix of the parameters
`stat_only`	Boolean. If `TRUE` will only return the result of the applied stat function, otherwise returns both the stat result and the result of the function on all parameters.
`digits`	Integer. How many digits to round the output to
`...`	Optional additional arguments that can be passed to `get_pars`

Value

A matrix or vector of ESS values for the selected parameter type.

Examples

```
ess_summary(samples_LNR, selection = "alpha")
```

`fit.emc` *Model estimation in EMC2*

Description

General purpose function to estimate models specified in EMC2.

Usage

```
## S3 method for class 'emc'
fit(
  emc,
  stage = NULL,
  iter = 1000,
  stop_criteria = NULL,
  report_time = TRUE,
  p_accept = 0.8,
  step_size = 100,
  verbose = TRUE,
  verboseProgress = FALSE,
  fileName = NULL,
  particles = NULL,
  particle_factor = 50,
  cores_per_chain = 1,
  cores_for_chains = length(emc),
  max_tries = 20,
  n_blocks = 1,
  ...
)

fit(emc, ...)
```

Arguments

`emc`	An emc object created with `make_emc`, or a path to where the emc object is stored.
`stage`	A string. Indicates which stage to start the run from, either `preburn`, `burn`, `adapt` or `sample`. If unspecified, it will run the subsequent stage (if there is one).
`iter`	An integer. Indicates how many iterations to run in the sampling stage.
`stop_criteria`	A list. Defines the stopping criteria and for which types of parameters these should hold. See the details and examples section.
`report_time`	Boolean. If `TRUE`, the time taken to run the MCMC chains till completion of the `stop_criteria` will be printed.
`p_accept`	A double. The target acceptance probability of the MCMC process. This fine-tunes the width of the search space to obtain the desired acceptance probability. Defaults to .8
`step_size`	An integer. After each step, the stopping requirements as specified by `stop_criteria` are checked and proposal distributions are updated. Defaults to 100.
`verbose`	Logical. Whether to print messages between each step with the current status regarding the `stop_criteria`.
`verboseProgress`	Logical. Whether to print a progress bar within each step or not. Will print one progress bar for each chain and only if `cores_for_chains = 1`.
`fileName`	A string. If specified, will auto-save emc object at this location on every iteration.
`particles`	An integer. How many particles to use, default is `NULL` and `particle_factor` is used instead. If specified, `particle_factor` is overwritten.
`particle_factor`	An integer. `particle_factor` multiplied by the square root of the number of sampled parameters determines the number of particles used.
`cores_per_chain`	An integer. How many cores to use per chain. Parallelizes across participant calculations. Only available on Linux or Mac OS. For Windows, only parallelization across chains (`cores_for_chains`) is available.
`cores_for_chains`	An integer. How many cores to use across chains. Defaults to the number of chains. The total number of cores used is equal to `cores_per_chain` * `cores_for_chains`.
`max_tries`	An integer. How many times should it try to meet the finish conditions as specified by `stop_criteria`? Defaults to 20. `max_tries` is ignored if the required number of iterations has not been reached yet.
`n_blocks`	An integer. Number of blocks. Will block the parameter chains such that they are updated in blocks. This can be helpful in extremely tough models with a large number of parameters.
`...`	Additional optional arguments

Details

`stop_criteria` is either a list of lists with names of the stages, or a single list in which case its assumed to be for the sample `stage` (see examples). The potential stop criteria to be set are:

`selection` (character vector): For which parameters the `stop_criteria` should hold

`mean_gd` (numeric): The mean Gelman-Rubin diagnostic across all parameters in the selection

`max_gd` (numeric): The max Gelman-Rubin diagnostic across all parameters in the selection

`min_unique` (integer): The minimum number of unique samples in the MCMC chains across all parameters in the selection

`min_es` (integer): The minimum number of effective samples across all parameters in the selection

`omit_mpsrf` (Boolean): Whether to include the multivariate point-scale reduction factor in the Gelman-Rubin diagnostic. Default is `FALSE`.

`iter` (integer): The number of MCMC samples to collect.

The estimation is performed using particle-metropolis within-Gibbs sampling. For sampling details see:

Gunawan, D., Hawkins, G. E., Tran, M.-N., Kohn, R., & Brown, S. (2020). New estimation approaches for the hierarchical linear ballistic accumulator model. *Journal of Mathematical Psychology* ,96, 102368. doi.org/10.1016/j.jmp.2020.102368

Stevenson, N., Donzallaz, M. C., Innes, R. J., Forstmann, B., Matzke, D., & Heathcote, A. (2024). EMC2: An R Package for cognitive models of choice. doi.org/10.31234/osf.io/2e4dq

Value

An emc object

Examples

```
## Not run:
# First define a design
design_DDMaE <- design(data = forstmann,model=DDM,
                           formula =list(v~0+S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
                           constants=c(s=log(1)))
# Then make the emc object, we've omitted a prior here for brevity so default priors will be used.
emc_forstmann <- make_emc(forstmann, design)

# With the emc object we can start sampling by simply calling fit
emc_forstmann <- fit(emc_forstmann, fileName = "intermediate_save_location.RData")

# For particularly hard models it pays off to increase the ``particle_factor``
# and, although to a lesser extent, lower ``p_accept``.
emc_forstmann <- fit(emc_forstmann, particle_factor = 100, p_accept = .6)

# Example of how to use the stop_criteria:
emc_forstmann <- fit(emc_forstmann, stop_criteria = list(mean_gd = 1.1, max_gd = 1.5,
            selection = c('alpha', 'sigma2'), omit_mpsrf = TRUE, min_es = 1000))
# In this case the stop_criteria are set for the sample stage, which will be
# run until the mean_gd < 1.1, the max_gd < 1.5 (omitting the multivariate psrf)
# and the effective sample size > 1000,
```

```
# for both the individual-subject parameters ("alpha")
# and the group-level variance parameters.

# For the unspecified stages in the ``stop_criteria`` the default values
# are assumed which are found in Stevenson et al. 2024 <doi.org/10.31234/osf.io/2e4dq>

# Alternatively, you can also specify the stop_criteria for specific stages by creating a
# nested list
emc_forstmann <- fit(emc_forstmann, stop_criteria = list("burn" = list(mean_gd = 1.1, max_gd = 1.5,
            selection = c('alpha')), "adapt" = list(min_unique = 100)))

## End(Not run)
```

`forstmann` *Forstmann et al.'s data*

Description

A dataset containing the speed or accuracy manipulation for a Random Dot Motion experiment.

Usage

```
forstmann
```

Format

A data frame with 15818 rows and 5 variables:

E Factor with 3 levels for Speed, Accuracy and Neutral

R Factor with 2 levels for Left and Right responses

S Factor with 2 levels for Left and Right trials

rt reaction time for each trial as a double

subjects integer ID for each subject

Details

Details on the dataset can be found in the following paper:

Striatum and pre-SMA facilitate decision-making under time pressure

Birte U. Forstmann, Gilles Dutilh, Scott Brown, Jane Neumann, D. Yves von Cramon, K. Richard Ridderinkhof, Eric-Jan Wagenmakers.

Proceedings of the National Academy of Sciences Nov 2008, 105 (45) 17538-17542; DOI: 10.1073/pnas.0805903105

Source

https://www.pnas.org/doi/10.1073/pnas.0805903105

`gd_summary.emc` *Gelman-Rubin statistic*

Description

Returns the Gelman-Rubin diagnostics (otherwise known as the R-hat) of the selected parameter type; i.e. the ratio of between to within MCMC chain variance.

Usage

```
## S3 method for class 'emc'
gd_summary(
  emc,
  selection = "mu",
  omit_mpsrf = TRUE,
  stat = "max",
  stat_only = FALSE,
  digits = 3,
  ...
)

gd_summary(emc, ...)
```

Arguments

`emc`	An emc object
`selection`	A Character vector. Indicates which parameter types to check (e.g., `alpha`, `mu`, `sigma2`, `correlation`).
`omit_mpsrf`	Boolean. If `TRUE` also returns the multivariate point scale reduction factor (see `?coda::gelman.diag`).
`stat`	A string. Should correspond to a function that can be applied to a vector, which will be performed on the vector/rows or columns of the matrix of the parameters
`stat_only`	Boolean. If `TRUE` will only return the result of the applied stat function, otherwise returns both the stat result and the result of the function on all parameters.
`digits`	Integer. How many digits to round the output to
`...`	Optional additional arguments that can be passed to `get_pars`

Details

See: Gelman, A and Rubin, DB (1992) Inference from iterative simulation using multiple sequences, *Statistical Science*, 7, 457-511.

Full range of possible samples manipulations described in `get_pars`.

Value

A matrix or vector of R-hat values for the selected parameter type.

Examples

```
gd_summary(samples_LNR, selection = "correlation", stat = "mean", flatten = TRUE)
```

`get_BayesFactor` *Bayes Factors*

Description

returns the Bayes Factor for two models

Usage

```
get_BayesFactor(MLL1, MLL2)
```

Arguments

`MLL1`	Numeric. Marginal likelihood of model 1. Obtained with `run_bridge_sampling()`
`MLL2`	Numeric. Marginal likelihood of model 2. Obtained with `run_bridge_sampling()`

Value

The BayesFactor for model 1 over model 2

Examples

```
## Not run:
# First get the marginal likelihood for two_models
# Here the full model is an emc object with the hypothesized effect
# The null model is an emc object without the hypothesized effect
MLL_full <- run_bridge_sampling(full_model, cores_per_prop = 3)
MLL_null <- run_bridge_sampling(null_model, cores_per_prop = 3)
# Now we can calculate their Bayes factor
get_BayesFactor(MLL_full, MLL_null)

## End(Not run)
```

`get_data.emc` *Get data*

Description

Extracts data from an emc object

Usage

```
## S3 method for class 'emc'
get_data(emc)

get_data(emc)
```

Arguments

`emc`	an emc object

Details

emc adds columns and rows to a dataframe in order to facilitate efficient likelihood calculations. This function will return the data as provided originally.

Value

A dataframe of the original data

Examples

```
get_data(samples_LNR)
```

`get_pars` *Filter/manipulate parameters from emc object*

Description

Underlying function used in most plotting and object handling functions in EMC2. Can for example be used to filter/thin a parameter type (i.e, group-level means `mu`) and convert to an mcmc.list.

Usage

```
get_pars(
  emc,
  selection = "mu",
  stage = "sample",
  thin = 1,
  filter = 0,
  map = FALSE,
  add_recalculated = FALSE,
  length.out = NULL,
  by_subject = FALSE,
  return_mcmc = TRUE,
  merge_chains = FALSE,
  subject = NULL,
  flatten = FALSE,
  remove_dup = FALSE,
  remove_constants = TRUE,
  use_par = NULL,
  type = NULL,
  true_pars = NULL,
  chain = NULL,
  covariates = NULL
)
```

Arguments

`emc`	an emc object.
`selection`	A Character string. Indicates which parameter type to select (e.g., `alpha`, `mu`, `sigma2`, `correlation`).
`stage`	A character string. Indicates from which sampling stage(s) to take the samples from (i.e. `preburn`, `burn`, `adapt`, `sample`)
`thin`	An integer. By how much to thin the chains
`filter`	Integer or numeric vector. If an integer is supplied, iterations up until that integer are removed. If a vector is supplied, the iterations within the range are kept.
`map`	Boolean. If `TRUE` parameters will be mapped back to the cells of the experimental design using the design matrices. Otherwise the sampled parameters are returned. Only works for `selection = mu` or `selection = alpha`.
`add_recalculated`	Boolean. If `TRUE` will also add recalculated parameters, such as b in the LBA (b = B + A; see `?LBA`), or z in the DDM z = Z*A (see `?DDM`) only works when `map = TRUE`
`length.out`	Integer. Alternatively to thinning, you can also select a desired length of the MCMC chains, which will be thinned appropriately.
`by_subject`	Boolean. If `TRUE` for selections that include subject parameters (e.g. `alpha`), plot/stats are organized by subject, otherwise by parameter.

`return_mcmc`	Boolean. If `TRUE` returns an mcmc.list object, otherwise a matrix/array with the parameter type.
`merge_chains`	Boolean. If `TRUE` returns parameter type merged across chains.
`subject`	Integer (vector) or character (vector). If an integer will select the 'x'th subject(s), if a character it should match subject names in the data which will be selected.
`flatten`	Boolean. If `FALSE` for 3-dimensional samples (e.g., correlations: n-pars x n-pars x iterations). organizes by the dimension containing parameter names, otherwise collapses names across the first and second dimension. Does not apply for `selection = "alpha"`
`remove_dup`	Boolean. If `TRUE` removes duplicate values from the samples. Automatically set to `TRUE` if `flatten = TRUE`
`remove_constants`	Boolean. If `TRUE` removes constant values from the samples (e.g. 0s in the covariance matrix).
`use_par`	Character (vector). If specified, only these parameters are returned. Should match the parameter names (i.e. these are collapsed when `flatten = TRUE` and use_par should also be collapsed names).
`type`	Character indicating the group-level model selected. Only necessary if sampler isn't specified.
`true_pars`	Set of `true_parameters` can be specified to apply flatten or use_par on a set of true parameters
`chain`	Integer. Which of the chain(s) to return
`covariates`	Only needed with `plot_prior` and covariates in the design

Value

An mcmc.list object of the selected parameter types with the specified manipulations

Examples

```
# E.g. get the group-level mean parameters mapped back to the design
get_pars(samples_LNR, stage = "sample", map = TRUE, selection = "mu")

# Or return the flattened correlation, with 10 iterations per chain
get_pars(samples_LNR, stage = "sample", selection = "correlation", flatten = TRUE, length.out = 10)
```

`hypothesis.emc` *Within-model hypothesis testing*

Description

Approximates the Bayes factor for parameter effects using the savage-dickey ratio.

Usage

```
## S3 method for class 'emc'
hypothesis(
  emc,
  parameter = NULL,
  H0 = 0,
  fun = NULL,
  selection = "mu",
  do_plot = TRUE,
  use_prior_lim = TRUE,
  N = 10000,
  prior_plot_args = list(),
  ...
)

hypothesis(emc, ...)
```

Arguments

`emc`	An emc object
`parameter`	A string. A parameter which you want to compare to H0. Will not be used if a FUN is specified.
`H0`	An integer. The H0 value which you want to compare to
`fun`	A function. Specifies an operation to be performed on the sampled or mapped parameters.
`selection`	A Character string. Indicates which parameter type to use (e.g., `alpha`, `mu`, `sigma2`, `correlation`).
`do_plot`	Boolean. If `FALSE` will omit the prior-posterior plot and only return the savage-dickey ratio.
`use_prior_lim`	Boolean. If `TRUE` will use xlimits based on prior density, otherwise based on posterior density.
`N`	Integer. How many prior samples to draw
`prior_plot_args`	A list. Optional additional arguments to be passed to plot.default for the plotting of the prior density (see `par()`)
`...`	Optional arguments that can be passed to `get_pars`, `density`, or `plot.default` (see `par()`)

Details

Note this is different to the computation of the marginal deviance in `compare` since it only considers the group level effect and not the whole model (i.e. subject-level parameters). For details see: Wagenmakers, Lodewyckx, Kuriyal, & Grasman (2010).

Value

The Bayes factor for the hypothesis against H0.

Examples

```
# Here the emc object has an effect parameter (e.g. m),
# that maps onto a certain hypothesis.
# The hypothesis here is that m is different from zero.
# We can test whether there's a group-level effect on m:
hypothesis(samples_LNR, parameter = "m")
# Alternatively we can also test whether two parameters differ from each other
mdiff <- function(p)diff(p[c("m","m_lMd")])
hypothesis(samples_LNR,fun=mdiff)
```

`init_chains` *Initialize chains*

Description

Adds a set of start points to each chain. These start points are sampled from a user-defined multivariate normal across subjects.

Usage

```
init_chains(
  emc,
  start_mu = NULL,
  start_var = NULL,
  particles = 1000,
  cores_per_chain = 1,
  cores_for_chains = length(emc)
)
```

Arguments

`emc`	An emc object made by `make_emc()`
`start_mu`	A vector. Mean of multivariate normal used in proposal distribution
`start_var`	A matrix. Variance covariance matrix of multivariate normal used in proposal distribution. Smaller values will lead to less deviation around the mean.
`particles`	An integer. Number of starting values
`cores_per_chain`	An integer. How many cores to use per chain. Parallelizes across participant calculations.
`cores_for_chains`	An integer. How many cores to use to parallelize across chains. Default is the number of chains.

Value

An emc object

Examples

```
## Not run:
# Make a design and an emc object
design_DDMaE <- design(data = forstmann,model=DDM,
                           formula =list(v~0+S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
                           constants=c(s=log(1)))

DDMaE <- make_emc(forstmann, design_DDMaE)
# set up our mean starting points (same used across subjects).
mu <- c(v_Sleft=-2,v_Sright=2,a=log(1),a_Eneutral=log(1.5),a_Eaccuracy=log(2),
       t0=log(.2),Z=qnorm(.5),sv=log(.5),SZ=qnorm(.5))
# Small variances to simulate start points from a tight range
var <- diag(0.05, length(mu))
# Initialize chains, 4 cores per chain, and parallelizing across our 3 chains as well
# so 4*3 cores used.
DDMaE <- init_chains(DDMaE, start_mu = p_vector, start_var = var, cores_per_chain = 4)
# Afterwards we can just use fit
DDMaE <- fit(DDMaE, cores_per_chain = 4)

## End(Not run)
```

`LBA` *The Linear Ballistic Accumulator model*

Description

Model file to estimate the Linear Ballistic Accumulator (LBA) in EMC2.

Usage

`LBA()`

Details

Model files are almost exclusively used in `design()`.

Default values are used for all parameters that are not explicitly listed in the `formula` argument of `design()`.They can also be accessed with `LBA()$p_types`.

Parameter	**Transform**	**Natural scale**	**Default**	**Mapping**	**Interpretation**
v	-	[-Inf, Inf]	1		Mean evidence-accumulation rate
A	log	[0, Inf]	log(0)		Between-trial variation (range) in start point
B	log	[0, Inf]	log(1)	*b* = *B*+*A*	Distance from *A* to *b* (response threshold)
t0	log	[0, Inf]	log(0)		Non-decision time
sv	log	[0, Inf]	log(1)		Between-trial variation in evidence-accumulation rate

All parameters are estimated on the log scale, except for the drift rate which is estimated on the real line.

Conventionally, `sv` is fixed to 1 to satisfy scaling constraints.

The $b = B + A$ parameterization ensures that the response threshold is always higher than the between trial variation in start point of the drift rate.

Because the LBA is a race model, it has one accumulator per response option. EMC2 automatically constructs a factor representing the accumulators `lR` (i.e., the latent response) with level names taken from the `R` column in the data.

The `lR` factor is mainly used to allow for response bias, analogous to `Z` in the DDM. For example, in the LBA, response thresholds are determined by the B parameters, so `B~lR` allows for different thresholds for the accumulator corresponding to left and right stimuli (e.g., a bias to respond left occurs if the left threshold is less than the right threshold). For race models, the `design()` argument `matchfun` can be provided, a function that takes the `lR` factor (defined in the augmented data (d) in the following function) and returns a logical defining the correct response. In the example below, the match is simply such that the `S` factor equals the latent response factor: `matchfun=function(d)d$S==d$lR`. Then `matchfun` is used to automatically create a latent match (`lM`) factor with levels `FALSE` (i.e., the stimulus does not match the accumulator) and `TRUE` (i.e., the stimulus does match the accumulator). This is added internally and can also be used in model formula, typically for parameters related to the rate of accumulation.

Brown, S. D., & Heathcote, A. (2008). The simplest complete model of choice response time: Linear ballistic accumulation. *Cognitive Psychology, 57*(3), 153-178. https://doi.org/10.1016/j.cogpsych.2007.12.002

Value

A model list with all the necessary functions for EMC2 to sample

Examples

```
# When working with lM it is useful to design  an "average and difference"
# contrast matrix, which for binary responses has a simple canonical from:
ADmat <- matrix(c(-1/2,1/2),ncol=1,dimnames=list(NULL,"d"))
# We also define a match function for lM
matchfun=function(d)d$S==d$lR
# We now construct our design, with v ~ lM and the contrast for lM the ADmat.
design_LBABE <- design(data = forstmann,model=LBA,matchfun=matchfun,
                       formula=list(v~lM,sv~lM,B~E+lR,A~1,t0~1),
                       contrasts=list(v=list(lM=ADmat)),constants=c(sv=log(1)))
# For all parameters that are not defined in the formula, default values are assumed
# (see Table above).
```

`LNR` *The Log-Normal Race Model*

Description

Model file to estimate the Log-Normal Race Model (LNR) in EMC2.

Usage

```
LNR()
```

Details

Model files are almost exclusively used in `design()`.

Default values are used for all parameters that are not explicitly listed in the `formula` argument of `design()`.They can also be accessed with `LNR()$p_types`.

Parameter	Transform	Natural scale	Default	Mapping	Interpretation
m	-	[-Inf, Inf]	1		Scale parameter
s	log	[0, Inf]	log(1)		Shape parameter
t0	log	[0, Inf]	log(0)		Non-decision time

Because the LNR is a race model, it has one accumulator per response option. EMC2 automatically constructs a factor representing the accumulators `lR` (i.e., the latent response) with level names taken from the `R` column in the data.

In `design()`, `matchfun` can be used to automatically create a latent match (`lM`) factor with levels `FALSE` (i.e., the stimulus does not match the accumulator) and `TRUE` (i.e., the stimulus does match the accumulator). This is added internally and can also be used in the model formula, typically for parameters related to the rate of accumulation (see the example below).

Rouder, J. N., Province, J. M., Morey, R. D., Gomez, P., & Heathcote, A. (2015). The lognormal race: A cognitive-process model of choice and latency with desirable psychometric properties. *Psychometrika, 80*, 491-513. https://doi.org/10.1007/s11336-013-9396-3

Value

A model list with all the necessary functions for EMC2 to sample

Examples

```
# When working with lM it is useful to design  an "average and difference"
# contrast matrix, which for binary responses has a simple canonical from:
ADmat <- matrix(c(-1/2,1/2),ncol=1,dimnames=list(NULL,"d"))
# We also define a match function for lM
matchfun=function(d)d$S==d$lR
# We now construct our design, with v ~ lM and the contrast for lM the ADmat.
design_LNRmE <- design(data = forstmann,model=LNR,matchfun=matchfun,
                       formula=list(m~lM + E,s~1,t0~1),
                       contrasts=list(m=list(lM=ADmat)))
# For all parameters that are not defined in the formula, default values are assumed
# (see Table above).
```

`make_data` *Simulate data*

Description

Simulates data based on a model design and a parameter vector (`p_vector`) by one of two methods:

1. Creating a fully crossed and balanced design specified by the design, with number of trials per cell specified by the `n_trials` argument
2. Using the design of a data frame supplied, which allows creation of unbalanced and other irregular designs, and replacing previous data with simulated data

Usage

```
make_data(
  parameters,
  design = NULL,
  n_trials = NULL,
  data = NULL,
  expand = 1,
  mapped_p = FALSE,
  hyper = FALSE,
  ...
)
```

Arguments

`parameters`	parameter vector used to simulate data. Can also be a matrix with one row per subject (with corresponding row names) or an emc object with sampled parameters (in which case posterior medians of `alpha` are used to simulate data)
`design`	Design list created by `design()`
`n_trials`	Integer. If `data` is not supplied, number of trials to create per design cell
`data`	Data frame. If supplied, the factors are taken from the data. Determines the number of trials per level of the design factors and can thus allow for unbalanced designs
`expand`	Integer. Replicates the `data` (if supplied) expand times to increase number of trials per cell.
`mapped_p`	If `TRUE` instead returns a data frame with one row per design cell and columns for each parameter specifying how they are mapped to the design cells.
`hyper`	If `TRUE` the supplied parameters must be a set of samples, from which the group-level will be used to generate subject level parameters. See also `make_random_effects` to generate subject-level parameters from a hyper distribution.
`...`	Additional optional arguments

Details

To create data for multiple subjects see `?make_random_effects()`.

Value

A data frame with simulated data

Examples

```
# First create a design
design_DDMaE <- design(factors = list(S = c("left", "right"),
                                           E = c("SPD", "ACC"),
                                           subjects = 1:30),
                            Rlevels = c("left", "right"), model = DDM,
                            formula =list(v~0+S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
                            constants=c(s=log(1)))
# Then create a p_vector:
parameters <- c(v_Sleft=-2,v_Sright=2,a=log(1),a_EACC=log(2), t0=log(.2),
              Z=qnorm(.5),sv=log(.5),SZ=qnorm(.5))

# Now we can simulate data
data <- make_data(parameters, design_DDMaE, n_trials = 30)

# We can also simulate data based on a specific dataset
design_DDMaE <- design(data = forstmann,model=DDM,
                            formula =list(v~0+S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
                            constants=c(s=log(1)))
parameters <- c(v_Sleft=-2,v_Sright=2,a=log(1),a_Eneutral=log(1.5),a_Eaccuracy=log(2),
              t0=log(.2),Z=qnorm(.5),sv=log(.5),SZ=qnorm(.5))

data <- make_data(parameters, design_DDMaE, data = forstmann)
```

`make_emc` *Make an emc object*

Description

Creates an emc object by combining the data, prior, and model specification into a emc object that is needed in `fit()`.

Usage

```
make_emc(
  data,
  design,
  model = NULL,
  type = "standard",
  n_chains = 3,
  compress = TRUE,
  rt_resolution = 0.02,
  prior_list = NULL,
  grouped_pars = NULL,
  par_groups = NULL,
  ...
)
```

Arguments

`data`	A data frame, or a list of data frames. Needs to have the variable `subjects` as participant identifier.
`design`	A list with a pre-specified design, the output of `design()`.
`model`	A model list. If none is supplied, the model specified in `design()` is used.
`type`	A string indicating whether to run a `standard` group-level, `blocked`, `diagonal`, `factor`, or `single` (i.e., non-hierarchical) model.
`n_chains`	An integer. Specifies the number of mcmc chains to be run (has to be more than 1 to compute `rhat`).
`compress`	A Boolean, if `TRUE` (i.e., the default), the data is compressed to speed up likelihood calculations.
`rt_resolution`	A double. Used for compression, response times will be binned based on this resolution.
`prior_list`	A named list containing the prior. Default prior created if `NULL`. For the default priors, see `?get_prior_{type}`.
`grouped_pars`	An integer vector. Parameters on this location of the vector of parameters are treated as constant across subjects
`par_groups`	A vector. Only to be specified with type `blocked`, e.g., `c(1,1,1,2,2)` means the covariances of the first three and of the last two parameters are estimated as two separate blocks.
`...`	Additional, optional arguments.

Value

An uninitialized emc object

Examples

```
dat <- forstmann

# function that takes the lR factor (named diff in the following function) and
# returns a logical defining the correct response for each stimulus. In this
# case the match is simply such that the S factor equals the latent response factor.
matchfun <- function(d)d$S==d$lR

# design an "average and difference" contrast matrix
ADmat <- matrix(c(-1/2,1/2),ncol=1,dimnames=list(NULL,"diff"))

# specify design
design_LBABE <- design(data = dat,model=LBA,matchfun=matchfun,
formula=list(v~lM,sv~lM,B~E+lR,A~1,t0~1),
contrasts=list(v=list(lM=ADmat)),constants=c(sv=log(1)))

# specify priors
pmean <- c(v=1,v_lMdiff=1,sv_lMTRUE=log(.5), B=log(.5),B_Eneutral=log(1.5),
           B_Eaccuracy=log(2),B_lRright=0, A=log(0.25),t0=log(.2))
psd <- c(v=1,v_lMdiff=0.5,sv_lMTRUE=.5,
```

```
         B=0.3,B_Eneutral=0.3,B_Eaccuracy=0.3,B_lRright=0.3,A=0.4,t0=.5)
prior_LBABE <- prior(design_LBABE, type = 'standard',pmean=pmean,psd=psd)

# create emc object
LBABE <- make_emc(dat,design_LBABE,type="standard",  prior=prior_LBABE)
```

`make_random_effects` *Make random effects*

Description

Simulates subject-level parameters in the format required by `make_data()`.

Usage

```
make_random_effects(
  design,
  group_means,
  n_subj = NULL,
  variance_proportion = 0.2,
  covariances = NULL
)
```

Arguments

`design`	A design list. The design as specified by `design()`
`group_means`	A numeric vector. The group level means for each parameter, in the same order as `sampled_p_vector(design)`
`n_subj`	An integer. The number of subjects to generate parameters for. If `NULL` will be inferred from design
`variance_proportion`	A double. Optional. If `covariances` are not specified, the variances will be created by multiplying the means by this number. The covariances will be 0.
`covariances`	A covariance matrix. Optional. Specify the intended covariance matrix.

Value

A matrix of subject-level parameters.

Examples

```
# First create a design
design_DDMaE <- design(data = forstmann,model=DDM,
                            formula =list(v~0+S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
                            constants=c(s=log(1)))
# Then create a group-level means vector:
```

```
group_means =c(v_Sleft=-2,v_Sright=2,a=log(1),a_Eneutral=log(1.5),a_Eaccuracy=log(2),
               t0=log(.2),Z=qnorm(.5),sv=log(.5),SZ=qnorm(.5))
# Now we can create subject-level parameters
subj_pars <- make_random_effects(design_DDMaE, group_means, n_subj = 5)

# We can also define a covariance matrix to simulate from
subj_pars <- make_random_effects(design_DDMaE, group_means, n_subj = 5,
             covariances = diag(.1, length(group_means)))

# The subject level parameters can be used to generate data
make_data(subj_pars, design_DDMaE, n_trials = 10)
```

`mapped_par` *Parameter mapping back to the design factors*

Description

Maps a parameter vector that corresponds to sampled parameters of the cognitive model back to the experimental design. The parameter vector can be created using `sampled_p_vector()`. The returned matrix shows whether/how parameters differ across the experimental factors.

Usage

```
mapped_par(
  p_vector,
  design,
  model = NULL,
  digits = 3,
  remove_subjects = TRUE,
  covariates = NULL,
  ...
)
```

Arguments

`p_vector`	A parameter vector. Must be in the form of `sampled_p_vector(design)`
`design`	A design list. Created by `design`
`model`	Optional model type (if not already specified in `design`)
`digits`	Integer. Will round the output parameter values to this many decimals
`remove_subjects`	Boolean. Whether to include subjects as a factor in the design
`covariates`	Covariates specified in the design can be included here.
`...`	optional arguments

Value

Matrix with a column for each factor in the design and for each model parameter type (`p_type`).

Examples

```
# First define a design:
design_DDMaE <- design(data = forstmann,model=DDM,
                           formula =list(v~0+S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
                           constants=c(s=log(1)))
# Then create a p_vector:
p_vector=c(v_Sleft=-2,v_Sright=2,a=log(1),a_Eneutral=log(1.5),a_Eaccuracy=log(2),
          t0=log(.2),Z=qnorm(.5),sv=log(.5),SZ=qnorm(.5))
# This will map the parameters of the p_vector back to the design
mapped_par(p_vector,design_DDMaE)
```

`merge_chains` *Merge samples*

Description

Merges samples from all chains as one unlisted object.

Usage

```
merge_chains(emc)
```

Arguments

`emc`	An emc object, commonly the output of `fit()`

Details

Note that all sampling stages are included in the merged output, including iterations from the `preburn`, `burn`, and `adapt` stages. `merge_chains(emc)$samples$stage` shows the corresponding sampling stages.

Value

An unlisted emc object with all chains merged

`pairs_posterior` *Plot within-chain correlations*

Description

Plots within-chain parameter correlations (upper triangle) and corresponding scatterplots (lower triangle) to visualize parameter sloppiness.

Usage

```
pairs_posterior(
  emc,
  selection = "alpha",
  scale_subjects = TRUE,
  do_plot = TRUE,
  N = 500,
  ...
)
```

Arguments

`emc`	An emc object
`selection`	A Character string. Indicates which parameter type to plot (`alpha`, `mu`, `variance`, `covariance`, `correlation`).
`scale_subjects`	Boolean. To standardize each participant with `selection = "alpha"`, by subtracting the mean and divding by the standard deviation. This ensures the plot has every participant on the same scale.
`do_plot`	Boolean. Whether to plot the pairs plot, if `FALSE`, only the correlations are returned.
`N`	Integer for maximum number of iterations used (defaults to 500). If number of samples in stage or selection exceeds N, a random subset will be taken of size N
`...`	Optional arguments that can be passed to `get_pars`

Details

If `selection = alpha` the parameter chains are concatenated across participants, (after standardizing if `scale_subjects = TRUE`) and then correlated.

Value

Invisibly returns a matrix with the correlations between the parameters.

Examples

```
# Plot the sloppiness for the individual-level subjects
pairs_posterior(samples_LNR, selection = "alpha")

# We can also choose group-level parameters and subsets of the parameter space
pairs_posterior(samples_LNR, use_par = c("m", "t0"), selection = "sigma2")
```

`parameters.emc` *Returns a parameter type from an emc object as a data frame.*

Description

Returns a parameter type from an emc object as a data frame.

Usage

```
## S3 method for class 'emc'
parameters(emc, selection = "mu", N = NULL, resample = FALSE, ...)

parameters(emc, ...)
```

Arguments

`emc`	An emc object
`selection`	String designating parameter type (e.g. mu, sigma2, correlation, alpha)
`N`	Integer. How many samples to take from the posterior. If `NULL` will return the full posterior
`resample`	Boolean. If `TRUE` will sample N samples from the posterior with replacement
`...`	Optional arguments that can be passed to `get_pars`

Value

A data frame with one row for each sample (with a subjects column if selection = "alpha")

`plot.emc` *Plot function for emc objects*

Description

Makes trace plots for model parameters.

Usage

```
## S3 method for class 'emc'
plot(
  x,
  stage = "sample",
  selection = c("mu", "sigma2", "alpha"),
  layout = NA,
  ...
)
```

Arguments

`x`	An object of class `emc`
`stage`	A character string indicating the sampling stage to be summarized. Can be `preburn`, `burn`, `adapt`, or `sample`.
`selection`	A character vector indicating the parameter group(s). Defaults to `mu`, `sigma2`, and `alpha`.
`layout`	A vector indicating which layout to use as in par(mfrow = layout). If NA, will automatically generate an appropriate layout.
`...`	Optional arguments that can be passed to `get_pars` or `plot.default` (see `par()`)

Value

A trace/acf plot of the selected MCMC chains

Examples

```
plot(samples_LNR)
# Or trace autocorrelation for the second subject:
plot(samples_LNR, subject = 2, selection = "alpha")

# Can also plot the trace of for example the group-level correlation:
plot(samples_LNR, selection = "correlation", col = c("green", "purple", "orange"), lwd = 2)
```

`plot_defective_density` *Plot defective densities for each subject and cell*

Description

Plots panels that contain a set of densities for each response option in the data. These densities are defective; their areas are relative to the respective response proportion. Across all responses, the area sums to 1.

Usage

```
plot_defective_density(
  data,
  subject = NULL,
  factors = NULL,
  layout = NA,
  correct_fun = NULL,
  rt_pos = "top",
  accuracy = "topright",
  ...
)
```

Arguments

`data`	A data frame. The experimental data in EMC2 format with at least `subject` (i.e., the subject factor), `R` (i.e., the response factor) and `rt` (i.e., response time) variable. Additional factor variables of the design are optional.
`subject`	An integer or character string selecting a subject from the data. If specified, only that subject is plotted. Per default (i.e., `NULL`), all subjects are plotted.
`factors`	A character vector of the factor names in the design to aggregate across Defaults to all (i.e., `NULL`).
`layout`	A vector indicating which layout to use as in par(mfrow = layout). If NA, will automatically generate an appropriate layout.
`correct_fun`	If specified, the accuracy for each subject is calculated, using the supplied function and an accuracy vector for each subject is returned invisibly.
`rt_pos`	legend function position character string for the mean response time (defaults to `top`)
`accuracy`	legend function position character string for accuracy (defaults to `topright`)
`...`	Optional arguments that can be passed to `get_pars`, `density`, or `plot.default` (see `par()`)

Value

If `correct_fun` is specified, a subject accuracy vector is returned invisibly

Examples

```
# First for each subject and the factor combination in the design:
plot_defective_density(forstmann)
# Now collapsing across subjects:
plot_defective_density(forstmann, factors = c("S", "E"))
# If the data is response coded, it generally makes sense to include the "S" factor
# because EMC2 will plot the "R" factor automatically. This way, choice accuracy can
# be examined
# Each subject's accuracy can be returned using a custom function:
print(plot_defective_density(forstmann, correct_fun = function(d) d$R == d$S))
```

`plot_fit` *Posterior predictive checks*

Description

Plot (defective) cumulative density functions of the observed data and data from the posterior predictive distribution: the probability of a response, p(R) as a function of response time for the experimental data and posterior predictive simulations.

Usage

```
plot_fit(
  data,
  pp,
  subject = NULL,
  factors = NULL,
  functions = NULL,
  stat = NULL,
  stat_name = "",
  adjust = 1,
  quants = c(0.025, 0.5, 0.975),
  do_plot = TRUE,
  xlim = NULL,
  ylim = NULL,
  layout = NULL,
  mfcol = FALSE,
  probs = c(1:99)/100,
  data_lwd = 2,
  fit_lwd = 1,
  q_points = c(0.1, 0.3, 0.5, 0.7, 0.9),
  qp_cex = 1,
  pqp_cex = 0.5,
  lpos = "right",
  main = ""
)
```

Arguments

`data`	A data frame. The experimental data in EMC2 format with at least `subject` (i.e., the subject factor), `R` (i.e., the response factor) and `rt` (i.e., response time) variable. Additional factor variables of the design are optional.
`pp`	A data frame. Posterior predictives created by `predict()`
`subject`	Integer or string selecting a subject from the data. If specified only that subject is plotted. `NULL` (i.e., the default), will plot all subjects.
`factors`	Character vector of factors in data to display separately. If NULL (default) use names of all columns in data except "trials","R", and "rt". Omitted factors are aggregated over. If NA treats entire data set as a single cell. Must be NA or NULL when using stat argument.
`functions`	A named list of functions that create new factors which can then be used by the factors and stat arguments.
`stat`	A function that takes the data/the posterior predictives and returns a single value. For the posterior predictives it will use a single value per replicate, which are then plotted as a density.
`stat_name`	A string naming what the stat argument calculates, used in labeling the x-axis of the plot.
`adjust`	Numeric. Density function bandwidth adjust parameter. See “?density‘
`quants`	A vector. Quantiles of the posterior predictives to return when stat argument is supplied.
`do_plot`	Boolean. Set to `FALSE` to only return the quantiles and omit the plots.
`xlim`	A numeric vector. x-axis plot limit.
`ylim`	A numeric vector. y-axis plot limit.
`layout`	A vector specifying the layout as in `par(mfrow = layout)`. If `NA` or `NULL` uses current plot window layout.
`mfcol`	Boolean. If `TRUE` uses par(mfrow = layout), otherwise uses par(mfcol = layout)
`probs`	Vector of probabilities at which to calculate cumulative density function
`data_lwd`	Integer. Line width for data
`fit_lwd`	Integer. Line width for posterior predictives
`q_points`	Vector. Quantile points to plot
`qp_cex`	Numeric. Cex for data quantile points
`pqp_cex`	Numeric. Cex for predicted quantile points
`lpos`	Character. Legend position, see `?legend()`.
`main`	Character. Pasted before the plot title, especially useful when specifying a stat argument.

Details

The data is plotted in black. Large grey points show the average quantiles across the posterior predictives. The small grey points represent the predicted quantile of an individual replicate, providing a representation of uncertainty in the model predictions.

If the stat argument is supplied (which calculates a statistic based on the data), the posterior predictives are plotted as a density over the different replicates. A vertical line is plotted at the value of that statistic for the experimental data.

If more than one subject is included, the data and fits are aggregated across subjects by default.

Also see `?plot_defective_density()` for more details.

Value

If stat argument is provided, a vector of observed values and predicted quantiles is returned

Examples

```
# First generate posterior predictives based on an emc object run with run_emc
pp <- predict(samples_LNR, n_cores = 1, n_post = 10)
# Then visualize the model fit
plot_fit(forstmann, pp, factors = c("S", "E"), layout = c(2,3))

# Specific statistics on the posterior predictives can also be specified
# This function calculates the difference in rt between two S levels.
# It takes the data (or the posterior predictives) as an argument
drt <- function(data) diff(tapply(data$rt,data[,c("S")],mean))
plot_fit(forstmann, pp, stat=drt,stat_name="Rt difference",
         main=("Left vs Right"))
```

`plot_pars` *Plots density for parameters*

Description

Plots the posterior and prior density for selected parameters of a model. Full range of samples manipulations described in `get_pars`.

Usage

```
plot_pars(
  emc,
  layout = NA,
  selection = "mu",
  show_chains = FALSE,
  plot_prior = TRUE,
  N = 10000,
  use_prior_lim = !all_subjects,
```

```
  lpos = "topright",
  true_pars = NULL,
  all_subjects = FALSE,
  prior_plot_args = list(),
  true_plot_args = list(),
  ...
)
```

Arguments

`emc`	An emc object
`layout`	A vector indicating which layout to use as in par(mfrow = layout). If NA, will automatically generate an appropriate layout.
`selection`	A Character string. Indicates which parameter type to use (e.g., `alpha`, `mu`, `sigma2`, `correlation`).
`show_chains`	Boolean (defaults to `FALSE`) plots a separate density for each chain.
`plot_prior`	Boolean. If `TRUE` will overlay prior density in the plot (default in red)
`N`	Integer. How many prior samples to draw
`use_prior_lim`	Boolean. If `TRUE` will use xlimits based on prior density, otherwise based on posterior density.
`lpos`	Character. Where to plot the contraction statistic.
`true_pars`	A vector or emc object. Can be used to visualize recovery. If a vector will plot a vertical line for each parameter at the appropriate place. If an emc object will plot the densities of the object as well, assumed to be the data-generating posteriors.
`all_subjects`	Boolean. Will plot the densities of all (selected) subjects overlaid with the group-level distribution
`prior_plot_args`	A list. Optional additional arguments to be passed to plot.default for the plotting of the prior density (see `par()`)
`true_plot_args`	A list. Optional additional arguments to be passed to plot.default for the plotting of the true parameters (see `par()`)
`...`	Optional arguments that can be passed to `get_pars`, `density`, or `plot.default` (see `par()`)

Value

An invisible return of the contraction statistics for the selected parameter type

Examples

```
# Full range of possibilities described in get_pars
plot_pars(samples_LNR)
# Or plot all subjects
plot_pars(samples_LNR, all_subjects = TRUE, col = 'purple')
# Or plot recovery
```

```
true_emc <- samples_LNR # This would normally be the data-generating samples
plot_pars(samples_LNR, true_pars = true_emc, true_plot_args = list(col = 'blue'), adjust = 2)
```

`plot_prior` *Title*

Description

Title

Usage

```
plot_prior(
  prior,
  design,
  selection = "mu",
  do_plot = TRUE,
  covariates = NULL,
  layout = NA,
  N = 50000,
  ...
)
```

Arguments

`prior`	A prior list created with `prior`
`design`	A design list created with `design`
`selection`	A Character string. Indicates which parameter type to use (e.g., `alpha`, `mu`, `sigma2`, `correlation`).
`do_plot`	Boolean. If `FALSE` will only return prior samples and omit plotting.
`covariates`	dataframe/functions as specified by the design
`layout`	A vector indicating which layout to use as in par(mfrow = layout). If NA, will automatically generate an appropriate layout.
`N`	Integer. How many prior samples to draw
`...`	Optional arguments that can be passed to get_pars, histogram, plot.default (see par()), or arguments required for the types of models e.g. n_factors for type = "factor"

Value

An mcmc.list object with prior samples of the selected type

Examples

```
# First define a design for the model
design_DDMaE <- design(data = forstmann,model=DDM,
                           formula =list(v~0+S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
                           constants=c(s=log(1)))
# Then set up a prior using make_prior
p_vector=c(v_Sleft=-2,v_Sright=2,a=log(1),a_Eneutral=log(1.5),a_Eaccuracy=log(2),
          t0=log(.2),Z=qnorm(.5),sv=log(.5),SZ=qnorm(.5))
psd <- c(v_Sleft=1,v_Sright=1,a=.3,a_Eneutral=.3,a_Eaccuracy=.3,
          t0=.4,Z=1,sv=.4,SZ=1)
# Here we left the variance prior at default
prior_DDMaE <- prior(design_DDMaE,mu_mean=p_vector,mu_sd=psd)
# Now we can plot all sorts of (implied) priors
plot_prior(prior_DDMaE, design_DDMaE, selection = "mu", N = 1e3)
plot_prior(prior_DDMaE, design_DDMaE, selection = "mu", mapped = FALSE, N=1e3)
# We can also plot the implied prior on the participant level effects.
plot_prior(prior_DDMaE, design_DDMaE, selection = "alpha", col = "green", N = 1e3)
```

`plot_relations` *Plot relations*

Description

An adjusted version of the `corrplot` package function `corrplot()` tailored to `EMC2` and the plotting of estimated correlations.

Usage

```
plot_relations(
  emc = NULL,
  stage = "sample",
  plot_cred = TRUE,
  plot_means = TRUE,
  only_cred = FALSE,
  nice_names = NULL,
  ...
)
```

Arguments

`emc`	An EMC2 object, commonly the output of `run_emc()`.
`stage`	Character. The stage from which to take the samples, defaults to the sampling stage `sample`.
`plot_cred`	Boolean. Whether to plot the 95 percent credible intervals or not
`plot_means`	Boolean. Whether to plot the means or not
`only_cred`	Boolean. Whether to only plot credible values

`nice_names`	Character string. Alternative names to give the parameters
`...`	Optional additional arguments

Value

No return value, creates a plot of group-level relations

Examples

```
# For a given set of hierarchical model samples we can make a
# correlation matrix plot.
plot_relations(samples_LNR, only_cred = TRUE, plot_cred = TRUE)
# We can also only plot the correlations where the credible interval does not include zero
plot_relations(samples_LNR, plot_means = TRUE, only_cred = TRUE)
```

`plot_sbc_ecdf` *Plot the ECDF difference in SBC ranks*

Description

Plots the difference in observed cumulative rank statistics and the expected cumulative distribution of a uniform distribution. The blue shaded areas indicate the 95% credible interval.

Usage

```
plot_sbc_ecdf(ranks, layout = NA)
```

Arguments

`ranks`	A list of named dataframes of the rank statistic
`layout`	Optional. A numeric vector specifying the layout using `par(mfrow = layout)`

Value

No returns

`plot_sbc_hist` *Plot the histogram of the observed rank statistics of SBC*

Description

Note that this plot is dependent on the number of bins, and a more general visualization is to use `plot_sbc_ecdf`

Usage

```
plot_sbc_hist(ranks, bins = 10, layout = NA)
```

Arguments

`ranks`	A list of named dataframes of the rank statistic
`bins`	An integer specifying the number of bins to use when plotting the histogram
`layout`	Optional. A numeric vector specifying the layout using `par(mfrow = layout)`

Value

No returns

`posterior_summary.emc` *Posterior quantiles*

Description

Returns the quantiles of the selected parameter type. Full range of possible samples manipulations described in `get_pars`.

Usage

```
## S3 method for class 'emc'
posterior_summary(
  emc,
  selection = "mu",
  probs = c(0.025, 0.5, 0.975),
  digits = 3,
  ...
)

posterior_summary(emc, ...)
```

Arguments

`emc`	An emc object
`selection`	A Character vector. Indicates which parameter types to check (e.g., `alpha`, `mu`, `sigma2`, `correlation`).
`probs`	A vector. Indicates which quantiles to return from the posterior.
`digits`	Integer. How many digits to round the output to
`...`	Optional additional arguments that can be passed to `get_pars`

Value

A list of posterior quantiles for each parameter group in the selected parameter type.

Examples

```
posterior_summary(samples_LNR)
```

`predict.emc` *Generate posterior predictives*

Description

Simulate `n_post` data sets using the posterior parameter estimates

Usage

```
## S3 method for class 'emc'
predict(
  object,
  hyper = FALSE,
  n_post = 100,
  n_cores = 1,
  stat = c("random", "mean", "median")[1],
  ...
)
```

Arguments

`object`	An emc object from which posterior predictives should be generated
`hyper`	Boolean. Defaults to `FALSE`. If `TRUE`, simulates from the group-level (`hyper`) parameters instead of the subject-level parameters.
`n_post`	Integer. Number of generated datasets
`n_cores`	Integer. Number of cores across which there should be parallellized
`stat`	Character. Can be `mean`, `median` or `random` (i.e., the default). Will take either random samples from the chain(s) or use the mean or median of the parameter estimates.
`...`	Optional additional arguments passed to `get_pars` or `make_data`

Value

A list of simulated data sets of length `n_post`

Examples

```
# based on an emc object ran by fit() we can generate posterior predictives
predict(samples_LNR, n_cores = 1, n_post = 10)
```

`prior` *Prior specification*

Description

Specify priors for the chosen model. These values are entered manually by default but can be recycled from another prior (given in the `update` argument).

Usage

```
prior(
  design,
  type = "standard",
  update = NULL,
  do_ask = NULL,
  fill_default = TRUE,
  ...
)
```

Arguments

`design`	Design list for which a prior is constructed, typically the output of `design()`
`type`	Character. What type of group-level model you plan on using i.e. `diagonal`
`update`	Prior list from which to copy values
`do_ask`	Character. For which parameter types or hyperparameters to ask for prior specification, i.e. `Sigma`, `mu` or `loadings` for factor models, but `theta_mu_mean` or `A` also works.
`fill_default`	Boolean, If `TRUE` will fill all non-specified parameters, and parameters outside of `do_ask`, to default values
`...`	Either values to prefill, i.e. `theta_mu_mean = c(1:6)`, or additional arguments such as `n_factors = 2`

Details

Where a value is not supplied, the user is prompted to enter numeric values (or functions that evaluate to numbers).

To get the prior help use `prior_help(type)`. With `type` e.g. 'diagonal'.

Value

A prior list object

Examples

```
# First define a design for the model
design_DDMaE <- design(data = forstmann,model=DDM,
                           formula =list(v~0+S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
                           constants=c(s=log(1)))
# Then set up a prior using prior
p_vector=c(v_Sleft=-2,v_Sright=2,a=log(1),a_Eneutral=log(1.5),a_Eaccuracy=log(2),
                     t0=log(.2),Z=qnorm(.5),sv=log(.5),SZ=qnorm(.5))
psd <- c(v_Sleft=1,v_Sright=1,a=.3,a_Eneutral=.3,a_Eaccuracy=.3,
                     t0=.4,Z=1,sv=.4,SZ=1)
# Here we left the variance prior at default
prior_DDMaE <- prior(design_DDMaE,mu_mean=p_vector,mu_sd=psd)
# Also add a group-level variance prior:
pscale <- c(v_Sleft=.6,v_Sright=.6,a=.3,a_Eneutral=.3,a_Eaccuracy=.3,
                             t0=.2,Z=.5,sv=.4,SZ=.3)
df <- .4
prior_DDMaE <- prior(design_DDMaE,mu_mean=p_vector,mu_sd=psd, A = pscale, df = df)
# If we specify a new design
design_DDMat0E <- design(data = forstmann,model=DDM,
                           formula =list(v~0+S,a~E, t0~E, s~1, Z~1, sv~1, SZ~1),
                           constants=c(s=log(1)))
# We can easily update the prior
prior_DDMat0E <- prior(design_DDMat0E, update = prior_DDMaE)
```

`profile_plot` *Likelihood profile plots*

Description

Creates likelihood profile plots from a design and the experimental data by varying one model parameter while holding all others constant.

Usage

```
profile_plot(
  data,
  design,
  p_vector,
  range = 0.5,
  layout = NA,
  p_min = NULL,
  p_max = NULL,
  use_par = NULL,
  n_point = 100,
```

```
  n_cores = 1,
  round = 3,
  true_plot_args = list(),
  ...
)
```

Arguments

`data`	A dataframe. Experimental data used, needed for the design mapping
`design`	A design list. Created using `design`.
`p_vector`	Named vector of parameter values (typically created with `sampled_p_vector(design)`)
`range`	Numeric. The max and min will be p_vector + range/2 and p_vector - range/2, unless specified in p_min or p_max.
`layout`	A vector indicating which layout to use as in par(mfrow = layout). If NA, will automatically generate an appropriate layout.
`p_min`	Named vector. If specified will instead use these values for minimum range of the selected parameters.
`p_max`	Named vector. If specified will instead use these values for maximum range of the selected parameters.
`use_par`	Character vector. If specified will only plot the profiles for the specified parameters.
`n_point`	Integer. Number of evenly spaced points at which to calculate likelihood
`n_cores`	Number of likelihood points evenly spaced between the minimum and maximum likelihood range.
`round`	Integer. To how many digits will the output be rounded.
`true_plot_args`	A list. Optional additional arguments that can be passed to plot.default for the plotting of the true vertical line.
`...`	Optional additional arguments that can be passed to plot.default.

Value

Vector with highest likelihood point, input and mismatch between true and highest point

Examples

```
# First create a design
design_DDMaE <- design(data = forstmann,model=DDM,
                       formula =list(v~0+S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
                       constants=c(s=log(1)))
# Then create a p_vector:
p_vector=c(v_Sleft=-2,v_Sright=2,a=log(.95),a_Eneutral=log(1.5),a_Eaccuracy=log(2),
          t0=log(.25),Z=qnorm(.5),sv=log(.5),SZ=qnorm(.5))
# Make a profile plot for some parameters. Specifying a custom range for t0.
profile_plot(p_vector = p_vector, p_min = c(t0 = -1.35),
             p_max = c(t0 = -1.45), use_par = c("a", "t0", "SZ"),
             data = forstmann, design = design_DDMaE, n_point = 10)
```

`RDM` *The Racing Diffusion Model*

Description

Model file to estimate the Racing Diffusion Model (RDM), also known as the Racing Wald Model.

Usage

```
RDM()
```

Details

Model files are almost exclusively used in `design()`.

Default values are used for all parameters that are not explicitly listed in the `formula` argument of `design()`.They can also be accessed with `RDM()$p_types`.

Parameter	**Transform**	**Natural scale**	**Default**	**Mapping**	**Interpretation**
v	log	[0, Inf]	log(1)		Evidence-accumulation rate (drift rate)
A	log	[0, Inf]	log(0)		Between-trial variation (range) in start point
B	log	[0, Inf]	log(1)	$b = B + A$	Distance from A to b (response threshold)
$t0$	log	[0, Inf]	log(0)		Non-decision time
sv	log	[0, Inf]	log(1)		Within-trial standard deviation of drift rate

All parameters are estimated on the log scale.

The parameterization $b = B + A$ ensures that the response threshold is always higher than the between trial variation in start point.

Conventionally, `s` is fixed to 1 to satisfy scaling constraints.

Because the RDM is a race model, it has one accumulator per response option. EMC2 automatically constructs a factor representing the accumulators `lR` (i.e., the latent response) with level names taken from the `R` column in the data.

The `lR` factor is mainly used to allow for response bias, analogous to Z in the DDM. For example, in the RDM, response thresholds are determined by the B parameters, so `B~lR` allows for different thresholds for the accumulator corresponding to "left" and "right" stimuli, for example, (e.g., a bias to respond left occurs if the left threshold is less than the right threshold).

For race models in general, the argument `matchfun` can be provided in `design()`. One needs to supply a function that takes the `lR` factor (defined in the augmented data (d) in the following function) and returns a logical defining the correct response. In the example below, this is simply whether the `S` factor equals the latent response factor: `matchfun=function(d)d$S==d$lR`. Using `matchfun` a latent match factor (`lM`) with levels `FALSE` (i.e., the stimulus does not match the accumulator) and `TRUE` (i.e., the stimulus does match the accumulator). This is added internally and can also be used in model formula, typically for parameters related to the rate of accumulation.

Tillman, G., Van Zandt, T., & Logan, G. D. (2020). Sequential sampling models without random between-trial variability: The racing diffusion model of speeded decision making. *Psychonomic Bulletin & Review, 27*(5), 911-936. https://doi.org/10.3758/s13423-020-01719-6

Value

A list defining the cognitive model

Examples

```
# When working with lM it is useful to design  an "average and difference"
# contrast matrix, which for binary responses has a simple canonical from:
ADmat <- matrix(c(-1/2,1/2),ncol=1,dimnames=list(NULL,"d"))
# We also define a match function for lM
matchfun=function(d)d$S==d$lR
# We now construct our design, with v ~ lM and the contrast for lM the ADmat.
design_RDMBE <- design(data = forstmann,model=RDM,matchfun=matchfun,
                      formula=list(v~lM,s~lM,B~E+lR,A~1,t0~1),
                      contrasts=list(v=list(lM=ADmat)),constants=c(s=log(1)))
# For all parameters that are not defined in the formula, default values are assumed
# (see Table above).
```

`recovery.emc` *Recovery plots*

Description

Plots recovery of data generating parameters/samples. Full range of samples manipulations described in `get_pars`

Usage

```
## S3 method for class 'emc'
recovery(
  emc,
  true_pars,
  selection = "mu",
  layout = NA,
  do_CI = TRUE,
  correlation = "pearson",
  stat = "rmse",
  digits = 3,
  CI = 0.95,
  ci_plot_args = list(),
  ...
)

recovery(emc, ...)
```

Arguments

`emc`	An emc object
`true_pars`	A vector of data-generating parameters or an emc object with data-generating samples
`selection`	A Character vector. Indicates which parameter types to plot (e.g., `alpha`, `mu`, `sigma2`, `correlation`).
`layout`	A vector indicating which layout to use as in par(mfrow = layout). If NA, will automatically generate an appropriate layout.
`do_CI`	Boolean. If `TRUE` will also include bars representing the credible intervals
`correlation`	Character. Which correlation to include in the plot. Options are either `pearson` or `spearman`
`stat`	Character. Which statistic to include in the plot. Options are either `rmse` or `coverage`
`digits`	Integer. How many digits to round the statistic and correlation in the plot to
`CI`	Numeric. The size of the credible intervals. Default is .95 (95%).
`ci_plot_args`	A list. Optional additional arguments to be passed to plot.default for the plotting of the credible intervals (see `par()`)
`...`	Optional arguments that can be passed to `get_pars` or `plot.default` (see `par()`)

Value

Invisible list with RMSE, coverage, and Pearson and Spearman correlations.

Examples

```
# Make up some values that resemble posterior samples
# Normally this would be true values that were used to simulate the data
# Make up some values that resemble posterior samples
# Normally this would be true values that were used to simulate the data
pmat <- matrix(rnorm(12, mean = c(-1, -.6, -.4, -1.5), sd = .01), ncol = 4, byrow = TRUE)
# Conventionally this would be created before one makes data with true values
recovery(samples_LNR, pmat, correlation = "pearson", stat = "rmse", selection = "alpha")
# Similarly we can plot recovery of other parameters with a set of true samples
true_samples <- samples_LNR # Normally this would be data-generating samples
recovery(samples_LNR, true_samples, correlation = "pearson", stat = "rmse",
         selection = "correlation", cex = 1.5,
         ci_plot_args = list(lty = 3, length = .2, lwd = 2, col = "brown"))
```

`run_bridge_sampling` *Estimating Marginal likelihoods using WARP-III bridge sampling*

Description

Uses bridge sampling that matches a proposal distribution to the first three moments of the posterior distribution to get an accurate estimate of the marginal likelihood. The marginal likelihood can be used for computing Bayes factors and posterior model probabilities.

Usage

```
run_bridge_sampling(
  emc,
  stage = "sample",
  filter = NULL,
  repetitions = 1,
  cores_for_props = 4,
  cores_per_prop = 1,
  both_splits = TRUE,
  ...
)
```

Arguments

`emc`	An emc object with a set of converged samples
`stage`	A character indicating which stage to use, defaults to `sample`
`filter`	An integer or vector. If integer, it will exclude up until that integer. If vector it will include everything in that range.
`repetitions`	An integer. How many times to repeat the bridge sampling scheme. Can help get an estimate of stability of the estimate.
`cores_for_props`	Integer. Warp-III evaluates the posterior over 4 different proposal densities. If you have the CPU, 4 cores will do this in parallel, 2 is also already helpful.
`cores_per_prop`	Integer. Per density we can also parallelize across subjects. Eventual cores will be `cores_for_props` * `cores_per_prop`. For efficiency users should prioritize cores_for_props being 4.
`both_splits`	Boolean. Bridge sampling uses a proposal density and a target density. We can estimate the stability of our samples and therefore MLL estimate, by running 2 bridge sampling iterations The first one uses the first half of the samples as the proposal and the second half as the target, the second run uses the opposite. If this is is set to `FALSE`, it will only run bridge sampling once and it will instead do an odd-even iterations split to get a more reasonable estimate for just one run.
`...`	Additional, optional more in-depth hyperparameters

Details

If not enough posterior samples were collected using `fit()`, bridge sampling can be unstable. It is recommended to run `run_bridge_sampling()` several times with the `repetitions` argument and to examine how stable the results are.

It can be difficult to converge bridge sampling for exceptionally large models, because of a large number of subjects (> 100) and/or cognitive model parameters.

For a practical introduction:

Gronau, Q. F., Heathcote, A., & Matzke, D. (2020). Computing Bayes factors for evidence-accumulation models using Warp-III bridge sampling. *Behavior research methods*, 52(2), 918-937. doi.org/10.3758/s13428-019-01290-6

For mathematical background:

Meng, X.-L., & Wong, W. H. (1996). Simulating ratios of normalizing constants via a simple identity: A theoretical exploration. *Statistica Sinica*, 6, 831-860. http://www3.stat.sinica.edu.tw/statistica/j6n4/j6n43/j6n43.htm

Meng, X.-L., & Schilling, S. (2002). Warp bridge sampling. *Journal of Computational and Graphical Statistics*, 11(3), 552-586. doi.org/10.1198/106186002457

Value

A vector of length repetitions which contains the marginal log likelihood estimates per repetition

Examples

```
## Not run:
# After `fit` has converged on a specific model
# We can take those samples and calculate the marginal log-likelihood for them
MLL <- run_bridge_sampling(list(samples_LNR), cores_per_prop = 2)
# This will run on 2*4 cores (since 4 is the default for ``cores_for_props``)

## End(Not run)
```

`run_emc` *Custom function for more controlled model estimation*

Description

Although typically users will rely on `fit`, this function can be used for more fine-tuned specification of estimation needs. The function will throw an error if a stage is skipped, the stages have to be run in order ("preburn", "burn", "adapt", "sample"). More details can be found in the `fit` help files (`?fit`).

Usage

```
run_emc(
  emc,
  stage,
  stop_criteria,
  p_accept = 0.8,
  step_size = 100,
  verbose = FALSE,
  verboseProgress = FALSE,
  fileName = NULL,
  particles = NULL,
  particle_factor = 50,
  cores_per_chain = 1,
  cores_for_chains = length(emc),
  max_tries = 20,
  n_blocks = 1
)
```

Arguments

`emc`	An emc object
`stage`	A string. Indicates which stage is to be run, either `preburn`, `burn`, `adapt` or `sample`
`stop_criteria`	A list. Defines the stopping criteria and for which types of parameters these should hold. See `?fit`.
`p_accept`	A double. The target acceptance probability of the MCMC process. This fine-tunes the width of the search space to obtain the desired acceptance probability. Defaults to .8
`step_size`	An integer. After each step, the stopping requirements as specified by `stop_criteria` are checked and proposal distributions are updated. Defaults to 100.
`verbose`	Logical. Whether to print messages between each step with the current status regarding the stop_criteria.
`verboseProgress`	Logical. Whether to print a progress bar within each step or not. Will print one progress bar for each chain and only if cores_for_chains = 1.
`fileName`	A string. If specified will autosave emc at this location on every iteration.
`particles`	An integer. How many particles to use, default is `NULL` and `particle_factor` is used instead. If specified will override `particle_factor`.
`particle_factor`	An integer. `particle_factor` multiplied by the square root of the number of sampled parameters determines the number of particles used.
`cores_per_chain`	An integer. How many cores to use per chain. Parallelizes across participant calculations. Only available on Linux or Mac OS. For Windows, only parallelization across chains (`cores_for_chains`) is available.
`cores_for_chains`	An integer. How many cores to use across chains. Defaults to the number of chains. the total number of cores used is equal to `cores_per_chain` * `cores_for_chains`.
`max_tries`	An integer. How many times should it try to meet the finish conditions as specified by stop_criteria? Defaults to 20. max_tries is ignored if the required number of iterations has not been reached yet.
`n_blocks`	An integer. Number of blocks. Will block the parameter chains such that they are updated in blocks. This can be helpful in extremely tough models with a large number of parameters.

Value

An emc object

Examples

```
## Not run:
# First define a design
design_DDMaE <- design(data = forstmann,model=DDM,
```

```
                         formula =list(v~0+S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
                         constants=c(s=log(1)))
# Then make the emc, we've omitted a prior here for brevity so default priors will be used.
emc <- make_emc(forstmann, design)

# Now for example we can specify that we only want to run the "preburn" phase
# for MCMC 200 iterations
emc <- run_emc(emc, stage = "preburn", stop_criteria = list(iter = 200))

## End(Not run)
```

`run_sbc` *Simulation-based calibration*

Description

Runs SBC for an EMC2 model and associated design. Returns normalized rank (between 0 and 1) and prior samples. For hierarchical models the group-level mean and the (implied) group-level (co-)variance are returned. For non-hierarchical models only the subject-level parameters rank is returned.

Usage

```
run_sbc(
  design_in,
  prior_in,
  replicates = 250,
  trials = 100,
  n_subjects = 30,
  plot_data = FALSE,
  verbose = TRUE,
  fileName = NULL,
  ...
)
```

Arguments

`design_in`	An emc design list. The design of the model to be used in SBC
`prior_in`	An emc prior list. The prior for the design to be used in SBC
`replicates`	Integer. The number of samples to draw from the prior
`trials`	Integer. The number of trials of the simulated data (per subject)
`n_subjects`	Integer. Only used for hierarchical models. The number of subjects to be used in data generation of each replicate
`plot_data`	Boolean. Whether to plot the data simulated (aggregated across subjects)
`verbose`	Verbose. Whether to print progress related messages
`fileName`	Character. Highly recommended, saves temporary results to the fileName
`...`	A list of optional additional arguments that can be passed to `fit` and `make_emc`

Value

The ranks and prior samples. For hierarchical models also the prior-generated subject-level parameters.

`sampled_p_vector` *Get model parameters from a design*

Description

Makes a vector with zeroes, with names and length corresponding to the model parameters of the design.

Usage

```
sampled_p_vector(
  design,
  model = NULL,
  doMap = TRUE,
  add_da = FALSE,
  all_cells_dm = FALSE
)
```

Arguments

`design`	a list of the design made with `design()`.
`model`	a model list. Defaults to the model specified in the design list.
`doMap`	logical. If `TRUE` will also include an attribute `map` with the design matrices that perform the mapping back to the design
`add_da`	Boolean. Whether to include the relevant data columns in the map attribute
`all_cells_dm`	Boolean. Whether to include all levels of a factor in the mapping attribute, even when one is dropped in the design

Value

Named vector.

Examples

```
# First define a design
design_DDMaE <- design(data = forstmann,model=DDM,
                           formula =list(v~0+S,a~E, t0~1, s~1, Z~1, sv~1, SZ~1),
                           constants=c(s=log(1)))
# Then for this design get which cognitive model parameters are sampled:
sampled_p_vector(design_DDMaE)
```

`samples_LNR` *An emc object of an LNR model of the Forstmann dataset using the first three subjects*

Description

An emc object with a limited number of samples and subjects of the Forstmann dataset. The object is a nested list of lenght three, each list containing the MCMC samples of the respective chain. The MCMC samples are stored in the samples element.

Usage

```
samples_LNR
```

Format

An emc object. An emc object is a list with a specific structure and elements, as outlined below.

data A list of dataframes, one for each subject included

par_names A character vector containing the model parameter names

n_pars The number of parameters in the model

n_subjects The number of unique subject ID's in the data

subjects A vector containing the unique subject ID's

prior A list that holds the prior for `theta_mu` (the model parameters). Contains the mean (`theta_mu_mean`), covariance matrix (`theta_mu_var`), degrees of freedom (`v`), and scale (`A`) and inverse covariance matrix (`theta_mu_invar`)

ll_func The log likelihood function used by pmwg for model estimation

samples A list with defined structure containing the samples, see the Samples Element section for more detail

grouped Which parameters are grouped across subjects, in this case none

sampler_nuis A sampler list for nuisance parameters (in this case there are none), similarly structured to the overall samples list of one of the MCMC chains.

Samples Element

The samples element of a emc object contains the different types of samples estimated by EMC2. These include the three main types of samples `theta_mu`, `theta_var` and `alpha` as well as a number of other items which are detailed here.

theta_mu samples used for estimating the model parameters (group level), an array of size (n_pars x n_samples)

theta_var samples used for estimating the parameter covariance matrix, an array of size (n_pars x n_pars x n_samples)

alpha samples used for estimating the subject random effects, an array of size (n_pars x n_subjects x n_samples)

stage A vector containing what PMwG stage each sample was drawn in

subj_ll The winning particles log-likelihood for each subject and sample

a_half Mixing weights used during the Gibbs step when creating a new sample for the covariance matrix

last_theta_var_inv The inverse of the last samples covariance matrix

idx The index of the last sample drawn

epsilon The scaling parameter of the proposal distributions for each subject array of size (n_subjects x n_samples)

origin From which propoosal distribution the accepted samples in the MCMC chain came, an array of size (n_subjects x n_samples)

Source

https://www.pnas.org/doi/10.1073/pnas.0805903105

`subset.emc` *Shorten an emc object*

Description

Shorten an emc object

Usage

```
## S3 method for class 'emc'
subset(
  x,
  stage = "sample",
  filter = NULL,
  thin = 1,
  keep_stages = FALSE,
  length.out = NULL,
  ...
)
```

Arguments

`x`	an emc object
`stage`	A character string. Indicates from which sampling stage(s) to take the samples from (i.e. `preburn`, `burn`, `adapt`, `sample`)
`filter`	Integer or numeric vector. If an integer is supplied, iterations up until that integer are removed. If a vector is supplied, the iterations within the range are kept.
`thin`	An integer. By how much to thin the chains
`keep_stages`	Boolean. If `TRUE`, will not remove samples from unselected stages.
`length.out`	Integer. Alternatively to thinning, you can also select a desired length of the MCMC chains, which will be thinned appropriately.
`...`	additional optional arguments

Value

A shortened emc object

Examples

```
subset(samples_LNR, length.out = 10)
```

`summary.emc` *Summary statistics for emc objects*

Description

Computes quantiles, `Rhat` and `ESS` for selected model parameters.

Usage

```
## S3 method for class 'emc'
summary(
  object,
  selection = c("mu", "sigma2", "alpha"),
  probs = c(0.025, 0.5, 0.975),
  digits = 3,
  ...
)
```

Arguments

`object`	An object of class `emc`
`selection`	A character string indicating the parameter type Defaults to `mu`, `sigma2`, and `alpha`. See below for more information.
`probs`	The quantiles to be computed. Defaults to the the 2.5%, 50% and 97.5% quantiles.
`digits`	An integer specifying rounding of output.
`...`	Optional arguments that can be passed to `get_pars`

Details

Note that if `selection` = `alpha` and `by_subject` = `TRUE` (default) is used, summary statistics are computed at the individual level. to the console but summary statistics for all subjects are returned by the function.

Value

A list of summary output.

Index