

Package ‘EMC2’

January 12, 2026

Title Bayesian Hierarchical Analysis of Cognitive Models of Choice

Version 3.4.1

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](https://doi.org/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 7.3.3

VignetteBuilder knitr

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

Config/testthat/edition 3

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

LinkingTo Rcpp, RcppArmadillo

Depends R (>= 3.5.0)

LazyData true

Config/testthat/parallel true

NeedsCompilation yes

Author Niek Stevenson [aut, cre] (ORCID:

<<https://orcid.org/0000-0003-3206-7544>>),

Michelle Donzallaz [aut],

Andrew Heathcote [aut],

Steven Miletic [aut],

Luke Strickland [ctb],

Frank Hezemans [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 2026-01-12 10:10:02 UTC

Contents

align_loadings	4
apply_kernel	6
auto_thin.emc	7
chain_n	8
check.emc	8
compare	9
compare_subject	11
contr.anova	12
contr.bayes	12
contr.decreasing	13
contr.increasing	14
convolve_design_matrix	14
credible.emc	16
credint.emc.prior	18
cut_factors	19
DDM	19
DDMGNG	20
design	21
design_fmri	23
ess_summary.emc	24
factor_diagram	25

fit.emc	26
fix_custom_kernel_pointers	29
forstmann	29
gd_summary.emc	30
get_BayesFactor	31
get_custom_kernel_pointers	32
get_data.emc	32
get_design.emc.prior	33
get_group_design.emc.prior	34
get_pars	34
get_power_spectra	36
get_prior.emc	37
get_trend_pnames	38
group_design	38
high_pass_filter	39
hypothesis.emc	40
init_chains	42
LBA	43
LNR	44
make_data	46
make_emc	47
make_random_effects	49
make_SEM_diagram	50
make_sem_structure	51
make_trend	53
mapped_pars	57
merge_chains	59
model_averaging	59
MRI	60
MRI_AR1	61
pairs_posterior	62
parameters.emc.prior	63
plot.emc	64
plot.emc.design	65
plot.emc.prior	66
plot_caf	67
plot_cdf	69
plot_delta	70
plot_density	72
plot_design.emc.design	73
plot_design_fmri	75
plot_fmri	77
plot_pars	78
plot_relations	80
plot_sbc_ecdf	81
plot_sbc_hist	81
plot_spectrum	82
plot_stat	83

plot_trend	84
predict.emc.prior	86
prior	87
prior_help	88
profile_plot	89
RDM	90
recovery.emc	92
register_trend	93
reshape_events	94
rotate_loadings	95
run_bridge_sampling	95
run_emc	97
run_hyper	99
run_sbc	100
sampled_pars	101
samples_LNR	102
SDT	104
set_custom_kernel_pointers	104
split_timeseries	105
subset.emc	106
summary.emc	107
summary.emc.design	108
summary.emc.group_design	108
summary.emc.prior	109
trend_help	109
update2version	110

Index 111

align_loadings	<i>Reorder MCMC Samples of Factor Loadings</i>
----------------	--

Description

This function reorders MCMC samples of factor loadings to address the label switching problem in Bayesian factor analysis. It implements a parallelized version of the code and algorithm proposed by Papastamoulis and Ntzoufras (2022)

Usage

```
align_loadings(
  emc = NULL,
  lambda = NULL,
  n_cores = 1,
  verbose = TRUE,
  rotate_fun = NULL
)
```

Arguments

emc	an 'emc' object of type <code>infnt_factor</code> .
lambda	Needs to be supplied if emc is not supplied. Array of factor loadings with dimensions p (variables) $\times$ q (factors) $\times$ n (MCMC iterations)
n_cores	Number of cores for parallel processing
verbose	Logical; whether to print progress information
rotate_fun	A function that returns an orthogonally rotated factor loadings matrix. If NULL uses <code>varimax</code>

Value

A list containing:

lambda_reordered	Array of reordered loadings
lambda_reordered_mcmc	Array of reordered loadings as MCMC object
lambda_hat	Matrix of mean loadings after reordering
v_vectors	Matrix of permutation vectors
c_vectors	Matrix of sign-switching vectors

References

Papastamoulis, P., & Ntzoufras, I. (2022). On the identifiability of Bayesian factor analytic models. *Statistical Computing*, 32(2), 1-29. doi: 10.1007/s11222-022-10084-4

Examples

```
# This function works natively with emc objects, but also factor arrays:
# Simulate a small example with 5 variables, 2 factors, and 10 MCMC iterations
set.seed(123)
p <- 5 # Number of variables
q <- 2 # Number of factors
n <- 10 # Number of MCMC iterations

# Create random factor loadings with label switching
lambda <- array(0, dim = c(p, q, n))
for (i in 1:n) {
  # Generate base loadings
  base_loadings <- matrix(rnorm(p*q, 0, 0.5), p, q)
  base_loadings[1:3, 1] <- abs(base_loadings[1:3, 1]) + 0.5 # Strong loadings on factor 1
  base_loadings[4:5, 2] <- abs(base_loadings[4:5, 2]) + 0.5 # Strong loadings on factor 2

  # Randomly switch labels and signs
  if (runif(1) > 0.5) {
    # Switch factor order
    base_loadings <- base_loadings[, c(2, 1)]
  }
}
```

```

    if (runif(1) > 0.5) {
      # Switch sign of factor 1
      base_loadings[, 1] <- -base_loadings[, 1]
    }
    if (runif(1) > 0.5) {
      # Switch sign of factor 2
      base_loadings[, 2] <- -base_loadings[, 2]
    }

    lambda[,i] <- base_loadings
  }

# Align the loadings
result <- align_loadings(lambda = lambda, verbose = TRUE, n_cores = 1)

# Examine the aligned loadings
print(result)

```

apply_kernel	<i>Apply a kernel implied in an emc object</i>
--------------	--

Description

This function extracts the appropriate trend, and applies its implied kernel using either the pure R implementation or the Rcpp version. When `mode = "compare"`, the function checks whether the two implementations produce identical output.

Usage

```

apply_kernel(
  kernel_pars,
  emc,
  subject = 1,
  input_pars = NULL,
  trend_n = 1,
  mode = "Rcpp"
)

```

Arguments

<code>kernel_pars</code>	A named vector of kernel parameters on the natural scale. Use <code>NULL</code> for kernels that do not require parameters.
<code>emc</code>	An emc.
<code>subject</code>	Subject index for which to apply the kernel. Defaults to 1.
<code>input_pars</code>	Optional parameter matrix containing externally supplied parameter values (e.g., trend parameters). Only needed for custom kernels.

trend_n	Integer specifying which trend to apply when multiple trends exist in the model. Defaults to 1. A warning is issued if the model contains more than one trend.
mode	Character string specifying which implementation to use: "R" Use the pure R implementation. "Rcpp" Use the Rcpp implementation (default).

Details

Applies the trend-specific kernel associated with an emc model to the a subject's data and returns the resulting kernel matrix.

Value

Returns a kernel matrix produced by the corresponding implementation.

auto_thin.emc	<i>Automatically Thin an emc Object</i>
---------------	---

Description

Uses the effective sample size of selection to determine how much to optimally thin an emc object

Usage

```
## S3 method for class 'emc'
auto_thin(emc, stage = "sample", selection = c("alpha", "mu"), ...)

auto_thin(emc, stage = "sample", selection = c("alpha", "mu"), ...)
```

Arguments

emc	an emc object.
stage	A character string. Indicates from which sampling stage(s) to take the samples from (i.e. preburn, burn, adapt, sample)
selection	Which parameter types (i.e. 'alpha' or 'mu' to consider when determining the effective sample size)
...	additional optional arguments

chain_n	<i>MCMC Chain Iterations</i>
---------	------------------------------

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	<i>Convergence Checks for an emc Object</i>
-----------	---

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

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

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 ($-2 \times \text{marginal likelihood}$) 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

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

```
compare(list(samples_LNR), cores_for_props = 1)
# Typically we would 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)
```

compare_subject	<i>Information Criteria For Each Participant</i>
-----------------	--

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,
  return_summary = FALSE,
  n_cores = 1,
  subject = NULL
)
```

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 tables of model weight results
digits	Integer, significant digits in printed table
return_summary	Return tables of model weight results
n_cores	Number of cores for parallel processing
subject	Used to select subset of subjects (integer or character vector)

Value

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

Examples

```
# For a broader illustration see `compare`.
# Here we just take two times the same model, but normally one would compare
# different models
compare_subject(list(m0 = samples_LNR, m1 = samples_LNR))
```

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 Enforcing 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_DDME <- 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	<i>Contrast Enforcing Decreasing Estimates</i>
------------------	--

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_DDME <- 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)))
}
```

<code>contr.increasing</code>	<i>Contrast Enforcing Increasing Estimates</i>
-------------------------------	--

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)))
}
```

<code>convolve_design_matrix</code>

Convolve Events with HRF to Construct Design Matrices

Description

This function convolves events with the HRF to construct design matrices for fMRI analysis.

Usage

```
convolve_design_matrix(
  timeseries,
  events,
  factors = NULL,
  contrasts = NULL,
  covariates = NULL,
  add_constant = TRUE,
  hrf_model = "glover",
  cell_coding = NULL,
```

```

    scale = TRUE,
    high_pass = TRUE,
    high_pass_model = "cosine",
    cut_off = 1e-12
  )

```

Arguments

timeseries	A data frame containing fMRI time series data with columns 'subjects', 'run', 'time', and at least one ROI column
events	A data frame containing event information with required columns subjects, run, onset, duration, event_type, and modulation
factors	A named list mapping factor names to event types
contrasts	A named list of contrast matrices for each factor
covariates	A character vector of event types to include as covariates
add_constant	A boolean specifying whether a 1 should be included to the design matrix post convolution
hrf_model	A character string specifying the HRF model to use ('glover', 'spm', 'glover + derivative', or 'spm + derivative')
cell_coding	A character vector of factor names to use cell coding for
scale	A boolean indicating whether to scale the design matrix.
high_pass	Logical indicating whether to apply high-pass filtering. Alternatively, specifying 'add' adds the regressors to the design matrix
high_pass_model	Character indicating which type of high-pass filtering to apply ('cosine', 'poly')
cut_off	A numeric value specifying the cutoff for the high-pass filter

Value

A list containing the design matrices

Examples

```

# Generate a simple example timeseries
ts <- data.frame(
  subjects = rep(1, 100),
  run = rep(1, 100),
  time = seq(0, 99),
  ROI1 = rnorm(100)
)

# Generate example events
events <- data.frame(
  subjects = rep(1, 4),
  run = rep(1, 4),
  onset = c(10, 30, 50, 70),
  duration = rep(0.5, 4),

```

```

    event_type = c("hard", "easy", "hard", "easy"),
    modulation = c(1, 1, 1, 1)
  )

  # Build design matrices
  design_matrices <- convolve_design_matrix(
    timeseries = ts,
    events = events,
    factors = list(difficulty = c("hard", "easy")),
    contrasts = list(difficulty = matrix(c(-1, 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 μ . 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

```
{
# Run a credible interval test (Bayesian 't-test')
credible(samples_LNR, x_name = "m")
# 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"])
}
```

credint.emc.prior *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.prior'
credint(
  x,
  selection = "mu",
  probs = c(0.025, 0.5, 0.975),
  digits = 3,
  N = 1000,
  covariates = NULL,
  ...
)

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

credint(x, ...)
```

Arguments

x	An emc or emc.prior 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
N	An integer. Number of samples to use for the quantile calculation (only for prior.emc objects)
covariates	A list of covariates to use for the quantile calculation (only for prior.emc objects)
...	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

```
credint(samples_LNR)
```

cut_factors

*Cut Factors Based on Credible Loadings***Description**

This function removes factors that do not have more than one credible loading based on the specified confidence interval.

Usage

```
cut_factors(emc, CI = 95)
```

Arguments

emc An 'emc' object containing factor analysis results
CI Numeric. Confidence interval percentage (default is 95)

Value

An 'emc' object with factors that don't meet the credibility criterion removed

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	-	$[-\text{Inf}, \text{Inf}]$	1		Mean evidence-accumulation
a	log	$[0, \text{Inf}]$	$\log(1)$		Boundary separation
$t0$	log	$[0, \text{Inf}]$	$\log(0)$		Non-decision time
s	log	$[0, \text{Inf}]$	$\log(1)$		Within-trial standard deviation
Z	probit	$[0, 1]$	$\text{qnorm}(0.5)$	$z = Z \times a$	Relative start point (bias)
SZ	probit	$[0, 1]$	$\text{qnorm}(0)$	$sz = 2 \times SZ \times \min(a \times Z, a \times (1-Z))$	Relative between-trial variability

<i>sv</i>	log	[0, Inf]	log(0)	Between-trial standard deviation
<i>st0</i>	log	[0, Inf]	log(0)	Between-trial variation (range)

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, *s* 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).
```

DDMGNG	<i>The GNG (go/nogo) Diffusion Decision Model</i>
--------	---

Description

In the GNG paradigm one of the two possible choices results in a response being withheld (a non-response), which is indicated in the data by an NA for the *rt*, with the corresponding level of the *R* (response) factor still being specified. For example, suppose the go response is coded as "yes" and nogo is coded as "no", then for a non-response (*R*,*rt*) = ("no",NA) and for a response e.g., (*R*,*rt*) = ("yes",1.36). The GNG paradigm must also have a response

Usage

```
DDMGNG()
```

Details

The model used is described in the following paper, with the addition of modeling the TIMEOUT (which is considered but not used in this paper).

Gomez, P., Ratcliff, R., & Perea, M. (2007). A Model of the Go/No-Go Task. *Journal of Experimental Psychology: General*, 136(3), 389–413. <https://doi.org/10.1037/0096-3445.136.3.389>

The likelihood of non-responses requires and evaluation of the DDM cdf, specifically $1 - p(\text{hitting the yes boundary before TIMEOUT})$.

To use these models three functions must be supplied in the design's function argument with the names TIMEOUT, Rnogo and Rgo. For example, assuming a 2.5 second timeout, and R factor with levels c("no", "yes") and "no" mapping to a non-response.

```
TIMEOUT=function(d)rep(2.5,nrow(d)) Rnogo=function(d)factor(rep("no",nrow(d)),levels=c("no","yes"))
Rgo=function(d)factor(rep("yes",nrow(d)),levels=c("no","yes"))
```

See the help for DDM for further details. At present this model is not fully implemented in C, so is a little slower to use than the DDM, but not greatly.

Value

A model list with all the necessary functions to sample

window (i.e., a length of time, TIMEOUT period, after which withholding is

assumed).

Examples

```
dGNG <- design(Rlevels = c("left","right"),
               factors=list(subjects=1,S=c("left","right")),
               functions=list(
                 TIMEOUT=function(d)rep(2.5,nrow(d)),
                 # no go response level
                 Rnogo=function(d)factor(rep("left",nrow(d)),levels=c("left","right")),
                 # go response level
                 Rgo=function(d)factor(rep("right",nrow(d)),levels=c("left","right"))),
               formula=list(v~S,a~1, Z~1, t0~1),
               model=DDMGNG)

p_vector <- sampled_pars(dGNG)
```

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,
    trend = NULL,
    transform = NULL,
    bound = NULL,
    ...
)

```

Arguments

formula	A list. Contains the design formulae in the format <code>list(y ~ x, a ~ z)</code> .
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 <code>subjects</code> must be used to indicate the participant factor variable, also in the data. Example: <code>list(subjects=levels(dat\$subjects), condition=levels(dat\$condition))</code>
Rlevels	A character vector. Contains the response factor levels. Example: <code>c("right", "left")</code>
model	A function, specifies the model type. Choose from the drift diffusion model (<code>DDM()</code> , <code>DDMt0natural()</code>), the log-normal race model (<code>LNRM()</code>), the linear ballistic model (<code>LBA()</code>), the racing diffusion model (<code>RDM()</code> , <code>RDMt0natural()</code>), or define your own model functions.
data	A data frame. <code>data</code> can be used to automatically detect factors, <code>Rlevels</code> and <code>covariates</code> in a dataset. The variable <code>R</code> needs to be a factor variable indicating the response variable. Any numeric column except <code>trials</code> and <code>rt</code> are treated as <code>covariates</code> , and all remaining factor variables are internally used in <code>factors</code> .
contrasts	Optional. A named list specifying a design matrix. Example for supplying a customized design matrix: <code>list(lm = matrix(c(-1/2, 1/2), ncol=1, dimnames=list(NULL, "diff")))</code>
matchfun	A function. Only needed for race models. Specifies whether a response was correct or not. Example: <code>function(d)d\$S==d\$IR</code> where <code>IR</code> refers to the latent response factor.
constants	A named vector that sets constants. Any parameter in <code>sampled_pars</code> can be set constant.
covariates	Names of numeric covariates.
functions	List of functions to create new factors based on those in the <code>factors</code> argument. These new factors can then be used in <code>formula</code> .
report_p_vector	Boolean. If <code>TRUE</code> (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.
trend	A trend list, as made by make_trend

transform	A list with custom transformations to be applied to the parameters of the model, if the conventional transformations aren't desired. See DDM() for an example of such transformations
bound	A list with custom bounds to be applied to the parameters of the model, if the conventional bound aren't desired. see DDM() for an example of such bounds. Bounds are used to set limits to the likelihood landscape that cannot reasonable be achieved with transform
...	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_LBABA <- 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)))
```

design_fmri

Create fMRI Design for EMC2 Sampling

Description

This function takes the output from convolve_design_matrix and transforms it into a design suitable for sampling with EMC2. It properly configures parameter types, bounds, and transformations for the specified model.

Usage

```
design_fmri(design_matrix, model = MRI_AR1, ...)
```

Arguments

`design_matrix` A list of design matrices, the output from `convolve_design_matrix`
`model` A function that returns a model specification, options are `MRI` or `MRI_AR1`
`...` Additional arguments passed to the model

Value

An object of class 'emc.design' suitable for EMC2 sampling

Examples

```
# Generate a simple example timeseries
ts <- data.frame(
  subjects = rep(1, 100),
  run = rep(1, 100),
  time = cumsum(rep(1.38, 100)),
  ROI1 = rnorm(100)
)

# Generate example events
events <- data.frame(
  subjects = rep(1, 4),
  run = rep(1, 4),
  onset = c(10, 30, 50, 70),
  duration = rep(0.5, 4),
  event_type = c("A", "B", "A", "B"),
  modulation = c(1, 1, 1, 1)
)

# Create convolved design matrix
design_matrix <- convolve_design_matrix(
  timeseries = ts,
  events = events,
  factors = list(condition = c("A", "B")),
  hrf_model = "glover"
)

# Create fMRI design for EMC2
fmri_design <- design_fmri(design_matrix, model = MRI_AR1)
```

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")
```

factor_diagram	<i>Factor diagram plot #Makes a factor diagram plot. Heavily based on the fa.diagram function of the psych package.</i>
----------------	---

Description

Factor diagram plot #Makes a factor diagram plot. Heavily based on the fa.diagram function of the psych package.

Usage

```
factor_diagram(
  emc = NULL,
  stage = "sample",
  loadings = NULL,
  standardize = TRUE,
  simple = FALSE,
  only_cred = TRUE,
  cut = 0,
  nice_names = NULL,
  factor_names = NULL,
  sort = TRUE,
  adj = 1,
  main = NULL,
  cex = NULL
)
```

Arguments

emc	An emc object
stage	Character. The stage from which to take the samples
loadings	An array of loadings. Can be alternatively supplied if emc is not supplied
standardize	Boolean. Whether to standardize the loadings
simple	Boolean. Whether the factor diagram should be simplified for visual clarity.
only_cred	Boolean. Whether to only plot the credible loadings
cut	Numeric. Mean loadings beneath this number will be excluded.
nice_names	Character vector. Alternative names to give the parameters
factor_names	Character vector. Names to give the different factors
sort	Boolean. Whether to sort the paramaters before plotting for visual clarity.
adj	Integer. Adjust to adjust loading values positions in the diagram if illegible.
main	Character vector. Title of the plot
cex	Integer. Font size

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,
  search_width = 1,
  step_size = 100,
  verbose = TRUE,
  fileName = NULL,
  particle_factor = 50,
  cores_per_chain = 1,
  cores_for_chains = length(emc),
  max_tries = 20,
  thin = FALSE,
  ...
)

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.
search_width	A double. Tunes target acceptance probability of the MCMC process. This fine-tunes the width of the search space to obtain the desired acceptance probability. 1 is the default width, increases lead to broader search.
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.
fileName	A string. If specified, will auto-save emc object at this location on every iteration.
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.
thin	A boolean. If TRUE will automatically thin the MCMC samples, closely matched to the ESS. Can also be set to a double, in which case 1/thin of the chain will be removed (does not have to be an integer).
...	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

```
# Define a design first
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$1R

# Drop most subjects for speed
dat <- forstmann[forstmann$subjects %in% unique(forstmann$subjects)[1:2],]
```

```

dat$subjects <- droplevels(dat$subjects)

design_LNR <- design(data = dat,model=LNR,matchfun=matchfun,
                    formula=list(m~1M,s~1,t0~1),
                    contrasts=list(m=list(1M=ADmat)))
# Before fit can be called, we first need to make an emc object
LNR_s <- make_emc(dat, design_LNR, rt_resolution = 0.05, n_chains = 2, compress = FALSE)
# Run fit, here illustrating how to use stop_criteria (also for speed purposes)
# LNR_s <- fit(LNR_s, cores_for_chains = 1, stop_criteria = list(
#   preburn = list(iter = 10), burn = list(mean_gd = 2.5), adapt = list(min_unique = 20),
#   sample = list(iter = 25, max_gd = 2)), verbose = FALSE, particle_factor = 30, step_size = 25)

```

fix_custom_kernel_pointers

Reset pointers of custom C++ trend kernels to an emc object

Description

When an emc object is loaded from disk, or returned by forked processes, the pointers to custom kernels need to be re-created. This is a convenience function to do this.

Usage

```
fix_custom_kernel_pointers(emc, pointer_source)
```

Arguments

emc	A target emc object with missing pointers
pointer_source	Either a trend object with correct pointers or another emc object with correct pointers

Value

An emc object with the custom pointers re-instated.

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 <code>?coda::gelman.diag</code>).
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 <code>get_pars</code>

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	<i>Bayes Factors</i>
-----------------	----------------------

Description

returns the Bayes Factor for two models

Usage

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

Arguments

MLL1	Numeric. Marginal likelihood of model 1. Obtained with <code>run_bridge_sampling()</code>
MLL2	Numeric. Marginal likelihood of model 2. Obtained with <code>run_bridge_sampling()</code>

Value

The BayesFactor for model 1 over model 2

Examples

```
# Normally one would compare two different models
# Here we use two times the same model:
M1 <- M0 <- run_bridge_sampling(samples_LNR, both_splits = FALSE, cores_for_props = 1)
get_BayesFactor(M1, M0)
```

```
get_custom_kernel_pointers
```

Extract pointers of custom C++ trend kernels from trend list or emc object

Description

Extract the pointers so they can be re-added to an emc object after loading it from disk.

Usage

```
get_custom_kernel_pointers(input_data)
```

Arguments

`input_data` Either an emc object or a trend list.

Value

A list of custom pointers. The list is of the same size as total number of trends; trends without a custom kernel return NULL.

```
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_design.emc.prior	<i>Get Design</i>
----------------------	-------------------

Description

Extracts design from an emc object

Usage

```
## S3 method for class 'emc.prior'
get_design(x)

## S3 method for class 'emc'
get_design(x)

get_design(x)
```

Arguments

x an emc or emc.prior object

Value

A design with class emc.design

Examples

```
get_design(samples_LNR)
```

```
get_group_design.emc.prior
```

Get Group Design

Description

Extracts group design from an emc object

Usage

```
## S3 method for class 'emc.prior'
get_group_design(x)

## S3 method for class 'emc'
get_group_design(x)

get_group_design(x)
```

Arguments

x an emc or emc.prior object

Value

A design with class emc.group_design

```
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 = get_last_stage(emc),
  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 for priors 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)
```

get_power_spectra	<i>Compute Power Spectra With Optional Subject-Level Aggregation</i>
-------------------	--

Description

Computes power spectral density estimates using [spectrum](#), optionally aggregated across subjects or posterior predictive samples. All arguments intended for the underlying spectral estimator should be supplied through spectrum.args.

Usage

```
get_power_spectra(data, by.postn = FALSE, spectrum.args = list())
```

Arguments

data	A data frame with reaction time data. Must contain subjects and rt, and for posterior predictive data optionally postn and trials.
by.postn	Logical. If TRUE, compute a separate spectrum for each posterior predictive draw and each posterior sample index.

`spectrum.args` A named list of arguments passed directly to `spectrum`. These override the defaults internally used in this function. Useful for customizing smoothing spans, detrending, tapering, and so on. Defaults: `list(spans=c(3, 5), detrend=FALSE, demean=TRUE, log=FALSE, taper=0)`. By default, we run `spectrum` without log, and log-transform while plotting

Details

The function organizes the data by subject (and optionally posterior sample index), computes spectra individually, interpolates spectra to a common frequency grid if needed, and averages them appropriately.

Value

Either a data frame with columns `freq` and `power`, or (if `by.postn = TRUE`) a list with frequency vector and a matrix of spectra across posterior samples.

`get_prior.emc`

Get Prior

Description

Extracts prior from an `emc` object

Usage

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

get_prior(emc)
```

Arguments

`emc` an `emc` object

Value

A prior with class `emc.prior`

Examples

```
get_prior(samples_LNR)
```

get_trend_pnames	<i>Get parameter types from trend object</i>
------------------	--

Description

Get parameter types from trend object

Usage

```
get_trend_pnames(trend)
```

Arguments

trend	A trend object created by make_trend()
-------	--

Value

A character vector of parameter names used in the trend

Examples

```
trend <- make_trend(par_names = "v", cov_names = "trial", kernels = "exp_incr")
get_trend_pnames(trend)
```

group_design	<i>Create Group-Level Design Matrices</i>
--------------	---

Description

Creates design matrices for group-level parameters based on subject-level design and formulas. This function is used for hierarchical modeling to specify how subject-level parameters vary across groups or conditions.

Usage

```
group_design(formula, data, subject_design, contrasts = NULL)
```

Arguments

formula	A list of formulas specifying the relationship between subject-level parameters and group-level predictors. Each formula should have a subject-level parameter on the left-hand side and group-level predictors on the right-hand side.
data	The same data as used in the subject-level design. Must include a 'subjects' column.
subject_design	An emc.design object containing the subject-level design.
contrasts	Optional list of contrast matrices to be used for categorical predictors.

Details

Here it is important to consider the interpretation of the group-level mean. This allows one to add covariates/group-level factors to the model. However, μ , the group-level mean, is still included for all parameters. μ represents the intercept in the design matrix, this intercept is always added to the group-level model. Therefore, to keep the interpretation of μ as the group-level mean, it is important to ensure that the design matrix has a mean of zero. If not, this function will throw a warning. For some unbalanced designs, this is unavoidable and the warning can be ignored.

Value

A list of design matrices, one for each parameter specified in the formula. The intercept is automatically included as the group-level mean and is omitted from the design matrices.

Examples

```
# Create subject-level design
subj_design <- design(data = forstmann, model = DDM,
                      formula = list(v ~ S, a ~ E, t0 ~ 1),
                      contrasts = list(S = contr.helmert))
# Add some age covariate and roughly demean
# Demeaning is important to ensure that the interpretation of the group-level intercept
# is the mean of the group (i.e., 'mu' still represents the group-level mean)
forstmann$age <- as.numeric(forstmann$subjects) - mean(as.numeric(forstmann$subjects))
# Create fake group column
forstmann$group <- ifelse(forstmann$subjects %in%
                          unique(forstmann$subjects)[seq(1, 19, 2)], "A", "B")

# Create group-level design matrices
group_des <- group_design(
  formula = list(v_S1 ~ age + group, a ~ age),
  data = forstmann,
  subject_design = subj_design,
  contrasts = list(group = contr.bayes)
)
# Then you can make the emc object with
emc <- make_emc(forstmann, subj_design, compress = FALSE, group_design = group_des)
```

high_pass_filter

Apply High-Pass Filtering to fMRI Data

Description

This function applies high-pass filtering to fMRI data to remove low-frequency noise and drift. It supports two filtering methods: cosine basis functions and polynomial regressors.

Usage

```
high_pass_filter(X, high_pass_model = "cosine", frame_times = NULL, ...)
```

Arguments

<code>X</code>	A data frame or matrix containing the data to be filtered. If it contains columns 'subjects' and 'run', the function will apply filtering separately for each subject-run combination.
<code>high_pass_model</code>	A character string specifying the high-pass filtering method. Options are 'cosine' (default) or 'poly' for polynomial regressors.
<code>frame_times</code>	A numeric vector of time points for each frame. If NULL, the function will attempt to extract this from a 'time' column in X.
<code>...</code>	Additional arguments passed to the function.

Value

A data frame or matrix with the same structure as X, but with high-frequency components removed from the data columns.

Examples

```
# Create a simple example data frame with drift
set.seed(123)
n_frames <- 100
time <- seq(0, 99)

# Create a signal with low-frequency drift
drift <- 0.1 * time
signal <- sin(2 * pi * 0.1 * time) + drift
noise <- rnorm(n_frames, 0, 0.5)
data <- signal + noise

# Create a data frame
df <- data.frame(
  time = time,
  signal = data
)

# Apply high-pass filtering using cosine basis functions
filtered_df <- high_pass_filter(df, high_pass_model = "cosine")
```

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_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 xlims based on prior density, otherwise based on posterior density.
N	Integer. How many prior samples to draw
prior_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_1Md")])
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 <code>make_emc()</code>
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.
...	optional additional arguments

Value

An emc object

Examples

```
# Make a design and an emc object
design_DDME <- design(data = forstmann,model=DDM,
                     formula =list(v~0+S,a~E, t0~1, s~1),
                     constants=c(s=log(1)))

DDME <- make_emc(forstmann, design_DDME, compress = FALSE)
# 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))
# 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.
DDME <- init_chains(DDME, start_mu = mu, start_var = var,
                   cores_per_chain = 1, cores_for_chains = 1, particles = 3)
# Afterwards we can just use fit
# DDME <- fit(DDME, cores_per_chain = 4)
```

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	-	$[-\text{Inf}, \text{Inf}]$	1		Mean evidence-accumulation rate
A	log	$[0, \text{Inf}]$	$\log(0)$		Between-trial variation (range) in start point
B	log	$[0, \text{Inf}]$	$\log(1)$	$b = B+A$	Distance from A to b (response threshold)
$t0$	log	$[0, \text{Inf}]$	$\log(0)$		Non-decision time
sv	log	$[0, \text{Inf}]$	$\log(1)$		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 1R (i.e., the latent response) with level names taken from the R column in the data.

The 1R 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 $B \sim 1R$ 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 1R 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$1R`. Then `matchfun` is used to automatically create a latent match (1M) 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 1M it is useful to design an "average and difference"
# contrast matrix, which for binary responses has a simple canonical form:
ADmat <- matrix(c(-1/2,1/2),ncol=1,dimnames=list(NULL,"d"))
# We also define a match function for 1M
matchfun=function(d)d$S==d$1R
# We now construct our design, with  $v \sim 1M$  and the contrast for 1M the ADmat.
design_LBABA <- design(data = forstmann,model=LBA,matchfun=matchfun,
                      formula=list(v~1M,sv~1M,B~E+1R,A~1,t0~1),
                      contrasts=list(v=list(1M=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
<i>m</i>	-	$[-\text{Inf}, \text{Inf}]$	1		Scale parameter
<i>s</i>	log	$[0, \text{Inf}]$	$\log(1)$		Shape parameter
<i>t0</i>	log	$[0, \text{Inf}]$	$\log(0)$		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 1R (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 (1M) 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 1M it is useful to design an "average and difference"
# contrast matrix, which for binary responses has a simple canonical form:
ADmat <- matrix(c(-1/2,1/2),ncol=1,dimnames=list(NULL,"d"))
# We also define a match function for 1M
matchfun=function(d)d$S==d$1R
# We now construct our design, with v ~ 1M and the contrast for 1M the ADmat.
design_LNRmE <- design(data = forstmann,model=LNR,matchfun=matchfun,
                      formula=list(m~1M + E,s~1,t0~1),
                      contrasts=list(m=list(1M=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,
  staircase = NULL,
  functions = NULL,
  ...
)
```

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.
staircase	Default NULL, used with stop-signal paradigm simulation to specify a staircase algorithm. If non-null and a list then passed through as is, if not it is assigned the default list structure: list(p=.25,SSD0=.25,stairstep=.05,stairmin=0,stairmax=Inf)
functions	List of functions you want to apply to the data generation.
...	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_DDME <- 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_DDME, n_trials = 30)

# We can also simulate data based on a specific dataset
design_DDME <- 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_DDME, 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 = 1/60,
  prior_list = NULL,
  group_design = NULL,
```

```

    par_groups = NULL,
    ...
)

```

Arguments

<code>data</code>	A data frame, or a list of data frames. Needs to have the variable subjects as participant identifier.
<code>design</code>	A list with a pre-specified design, the output of <code>design()</code> .
<code>model</code>	A model list. If none is supplied, the model specified in <code>design()</code> is used.
<code>type</code>	A string indicating whether to run a standard group-level, blocked, diagonal, factor, or single (i.e., non-hierarchical) model.
<code>n_chains</code>	An integer. Specifies the number of mcmc chains to be run (has to be more than 1 to compute rhat).
<code>compress</code>	A Boolean, if TRUE (i.e., the default), the data is compressed to speed up likelihood calculations.
<code>rt_resolution</code>	A double. Used for compression, response times will be binned based on this resolution.
<code>prior_list</code>	A named list containing the prior. Default prior created if NULL. For the default priors, see <code>?get_prior_{type}</code> .
<code>group_design</code>	A design for group-level mappings, made using <code>group_design()</code> .
<code>par_groups</code>	A vector. Indicates which parameters are allowed to correlate. Could either be a list of character vectors of covariance blocks. Or a numeric vector, e.g., <code>c(1,1,1,2,2)</code> means the covariances of the first three and of the last two parameters are estimated as two separate blocks.
<code>...</code>	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)$S==d$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~1M,sv~1M,B~E+1R,A~1,t0~1),
contrasts=list(v=list(1M=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,
                  compress = FALSE)
```

make_random_effects	<i>Generate Subject-Level Parameters</i>
---------------------	--

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

<code>design</code>	A design list. The design as specified by <code>design()</code>
<code>group_means</code>	A numeric vector. The group level means for each parameter, in the same order as <code>sampled_pars(design)</code>
<code>n_subj</code>	An integer. The number of subjects to generate parameters for. If <code>NULL</code> will be inferred from <code>design</code>
<code>variance_proportion</code>	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.
<code>covariances</code>	A covariance matrix. Optional. Specify the intended covariance matrix.

Value

A matrix of subject-level parameters.

Examples

```
# First create a design
design_DDME <- 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_DDME, group_means, n_subj = 19)

# We can also define a covariance matrix to simulate from
subj_pars <- make_random_effects(design_DDME, group_means, n_subj = 19,
                                covariances = diag(.1, length(group_means)))

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

make_SEM_diagram

Make SEM Diagram

Description

Make SEM Diagram

Usage

```
make_SEM_diagram(
  emc,
  plot_values = TRUE,
  cred_only = FALSE,
  par_names = NULL,
  cut = NULL,
  ...
)
```

Arguments

emc	an emc object
plot_values	whether to plot the values or just the nodes/edges
cred_only	whether to only plot credible values
par_names	optional, if specified will overwrite the parameter names with user-defined names
cut	optional. A numeric value factor loadings smaller than this value will be omitted.
...	optional additional arguments passed to render_graph from DiagrammeR

Value

Invisibly returns a DiagrammeR graph object

make_sem_structure	<i>Define Structural Equation Model (SEM) Matrices</i>
--------------------	--

Description

This function helps create the specification matrices (Lambda, B, K, G) for an SEM. It takes a design object, data, factor names, covariate column names, and list-based specifications for the paths to be estimated. The subject-level parameter names for Lambda_mat and K_mat rows are derived from `sampled_pars(design)`. It validates that covariates are consistent per subject (subject column in data must be named "subjects") and includes an aggregated subject-level covariate data frame named `covariates` in the output list. For identifiability, the first parameter listed in `lambda_specs` for each factor is fixed to 1.

Usage

```
make_sem_structure(
  data = NULL,
  design,
  covariate_cols = NULL,
  lambda_specs = NULL,
  b_specs = NULL,
  k_specs = NULL,
  g_specs = NULL,
  fixed_value = 0
)
```

Arguments

- | | |
|-----------------------------|--|
| <code>data</code> | A data frame containing a column named "subjects" and any covariate columns specified in <code>covariate_cols</code> . |
| <code>design</code> | An <code>emc.design</code> object, as created by the <code>design()</code> function. The parameter names for the SEM are derived from <code>names(sampled_pars(design))</code> . |
| <code>covariate_cols</code> | Character vector or NULL. Column names in data to be used as covariates for K_mat and G_mat. If NULL, no covariates are processed. |
| <code>lambda_specs</code> | A list defining factor loadings. The list names should be factor names and each element should be a character vector of parameter names (from <code>names(sampled_pars(design))</code>) that load onto that factor. The first parameter listed for each factor will be fixed to 1 for identifiability. Example: <code>list(Factor1 = c("v_Sleft", "a_Eneutral"), Factor2 = "t0")</code> Here, <code>Lambda_mat["v_Sleft", "Factor1"]</code> would be 1. |
| <code>b_specs</code> | A list defining regressions among factors. List names are outcome factors, elements are character vectors of predictor factors. Example: <code>list(Factor2 = "Factor1", Factor3 = c("Factor1", "Factor2"))</code> |

k_specs	A list defining covariate effects on subject-level parameters. List names are parameter names (from <code>names(sampled_pars(design))</code>), elements are character vectors of covariate names (must be present in <code>covariate_cols</code> and thus in the processed covariates data frame). Example: <code>list(v_Sleft = "cov1", a_Eneutral = c("cov1", "cov2"))</code>
g_specs	A list defining covariate effects on factors. List names are factor names, elements are character vectors of covariate names. Example: <code>list(Factor1 = "cov1", Factor2 = c("cov1", "cov2"))</code>
fixed_value	Numeric. The value used for fixed paths in the matrices that are not set to 1 for identifiability or Inf for estimation. Default is 0.

Value

A list containing:

- `Lambda_mat`: The factor loading matrix.
- `B_mat`: The matrix of regressions among factors.
- `K_mat`: The matrix of covariate effects on subject-level parameters.
- `G_mat`: The matrix of covariate effects on factors.
- `par_names`: The subject-level parameter names derived from `sampled_pars(design)`.
- `factor_names`: The provided SEM factor names.
- `covariates`: A data frame with one row per unique subject and columns for each covariate, containing the unique subject-level values. Column names are the covariate names.

Examples

```
# Create a design object (simplified from design.R example)
ADmat <- matrix(c(-1/2,1/2),ncol=1,dimnames=list(NULL,"diff"))
matchfun_example <- function(d) d$S==d$I $R # Example match function

example_design_obj <- design(
  data = forstmann,
  model= LBA,
  matchfun=matchfun_example,
  formula=list(v~IM,sv~IM,B~E+IR,A~1,t0~1),
  contrasts=list(v=list(IM=ADmat)),
  constants=c(sv=log(1)),
)

# SEM Factor names

# Make a copy of forstmann for example modification
forstmann_mod <- forstmann
set.seed(123) # for reproducibility
subj_trait_values <- stats::setNames(rnorm(length(levels(forstmann_mod$subjects))),
                                     levels(forstmann_mod$subjects))
forstmann_mod$SubjTrait <- subj_trait_values[forstmann_mod$subjects]

my_cov_cols <- c("SubjTrait")
```

```

lambda_example_specs <- list(
  Speed = c("v", "v_1Mdiff"), # "v" will be fixed to 1
  Caution = c("B", "B_Eneutral", "B_Eaccuracy", "B_1Rright", "A") # "B" fixed to 1
)
b_example_specs <- list(Caution = "Speed")
k_example_specs <- list(t0 = "SubjTrait") # "SubjTrait" must be in my_cov_cols
g_example_specs <- list(Speed = "SubjTrait")

sem_settings_definition <- make_sem_structure(
  data = forstmann_mod,
  design = example_design_obj,
  covariate_cols = my_cov_cols,
  lambda_specs = lambda_example_specs,
  b_specs = b_example_specs,
  k_specs = k_example_specs,
  g_specs = g_example_specs
)

print(sem_settings_definition$Lambda_mat)
print(sem_settings_definition$B_mat)
print(sem_settings_definition$K_mat)
print(sem_settings_definition$G_mat)
print(head(sem_settings_definition$covariates))

```

make_trend

Create a trend specification for model parameters

Description

Create a trend specification for model parameters

Usage

```

make_trend(
  par_names,
  cov_names = NULL,
  kernels,
  bases = NULL,
  shared = NULL,
  trend_pnames = NULL,
  phase = "premap",
  par_input = NULL,
  at = "1R",
  maps = NULL,
  custom_trend = NULL,
  ffill_na = NULL
)

```

Arguments

<code>par_names</code>	Character vector specifying which parameters to apply trend to
<code>cov_names</code>	Character vector specifying which covariates to use for each trend
<code>kernels</code>	Character vector specifying which kernel function to use for each trend
<code>bases</code>	Optional character vector specifying which base function to use for each trend
<code>shared</code>	Named list with entries the parameter names to be shared and the names the new names of the shared parameter.
<code>trend_pnames</code>	Optional character vector specifying custom parameter names
<code>phase</code>	Character vector (length 1 or <code>length(par_names)</code>) specifying the phase for each trend entry; one of "premap", "pretransform", or "posttransform". Defaults to "premap".
<code>par_input</code>	Optional character vector(s) of parameter names to use as additional inputs for the trend
<code>at</code>	If NULL, trend is applied to every row in the <code>dadm</code> . If a factor name (e.g., "IR"), trend is applied only to entries corresponding to the first level of that factor, and fed forward to the other levels of that factor. Defaults to "IR". For DDMs, <code>at</code> should be set to NULL.
<code>maps</code>	List of functions that create matrices with which to multiply the covariates before applying the base. See details.
<code>custom_trend</code>	A trend registered with <code>register_trend</code>
<code>ffill_na</code>	Determines how missing covariate values are handled. If TRUE, missing values are forward-filled using the last known non-NA value after applying the kernel. If FALSE, trials with missing covariates contribute 0 instead. The default (NULL) is interpreted as TRUE for delta-rule models and FALSE otherwise.

Details

The `maps` argument accepts one or more functions that translate trial-level covariates into accumulator-specific predictors.

Example of a minimal map function:

```

advantage_map <- function(dadm, cov_names) {
  lS    <- paste0('cov', ifelse(dadm$IR == 'left', dadm$cov_left, dadm$cov_right))
  lSoth <- paste0('cov', ifelse(dadm$IR == 'right', dadm$cov_left, dadm$cov_right))
  plus  <- sapply(cov_names, function(x) ifelse(lS == x, 1, 0))
  minus <- sapply(cov_names, function(x) ifelse(lSoth == x, -1, 0))
  plus + minus
}

```

Multiple maps may be supplied, in which case the model will create a separate base parameter for each map. See more examples below.

Value

A list containing the trend specifications for each parameter

Examples

```
# Put trend on B and v parameters
trend <- make_trend(
  par_names = c("B", "v"),
  cov_names = "strial",
  kernels = c("exp_incr", "poly3"),
  phase = "premap",
  shared = list(shrd = list("B.B0", "v.d1"))
)
get_trend_pnames(trend)

# Using covariate maps

# Covariate maps allow you to specify how trial-by-trial covariates influence
# model parameters for each accumulator. The example below uses a simple data
# frame with two trials. `cov_left` and `cov_right` specify which covariates
# correspond to the left and right accumulators on each trial. `S` indicates the
# correct response, and `cov1`-`cov4` contain the actual covariate values.

data <- data.frame(
  subjects = rep(1, 2),
  S        = c('left', 'right'),
  cov_left = c('1', '4'),
  cov_right= c('3', '2'),
  rt       = c(1.2, 0.8),
  R        = factor(c('left', 'right')),
  cov1     = c(1, NA),
  cov2     = c(NA, 1),
  cov3     = c(NA, 1),
  cov4     = c(1, 1)
)

# A covariate map function must take `dadm` and `cov_names` as inputs and return
# a matrix of size (nrow(dadm), length(cov_names)), coding how each covariate
# contributes to each accumulator.

advantage_map <- function(dadm, cov_names) {

  # Which stimulus does the accumulator correspond to on each trial?
  lS <- paste0('cov', ifelse(dadm$R == 'left', dadm$cov_left, dadm$cov_right))

  # Which stimulus does the *other* accumulator correspond to?
  lSother <- paste0('cov', ifelse(dadm$R == 'right', dadm$cov_left, dadm$cov_right))

  # Build indicator matrices
  map_plus1 <- sapply(cov_names, function(col) ifelse(lS == col, 1, 0))
  map_minus1 <- sapply(cov_names, function(col) ifelse(lSother == col, -1, 0))

  map_plus1 + map_minus1
}
```

A covariate map function can be supplied to make_trend(), which creates the mapping
specification for the model for each participant. Here, a single map ('differences') is provided.

```
trend <- make_trend(
  par_names = 'v',
  kernels   = 'delta',
  bases     = 'lin',
  cov_names = list(c('cov1', 'cov2', 'cov3', 'cov4')),
  maps      = list('differences' = advantage_map),
  at        = 'lR'
)
```

```
design_RDM <- design(
  model = RDM,
  data  = data,
  formula= list(B ~ 1, v ~ 1, t0 ~ 1),
  trend = trend
)
```

```
emc <- make_emc(data, design_RDM, type = 'single')
```

```
# The resulting covariate maps for each subject are attached to the `dadm`:
attr(emc[[1]]$data[[1]], 'covariate_maps')
# And to confirm that this mapping is correct, compare with the corresponding `dadm`
emc[[1]]$data[[1]]
```

```
# You can also provide multiple covariate maps. Each additional map introduces
# a separate base parameter. For example, the following `sum_map` is suitable
# for RL-ARD-type models:
```

```
sum_map <- function(dadm, cov_names) {
  # Which stimulus does the accumulator correspond to on each trial?
  lS <- paste0('cov', ifelse(dadm$lR == 'left', dadm$cov_left, dadm$cov_right))
  # Which stimulus does the *other* accumulator correspond to?
  lSother <- paste0('cov', ifelse(dadm$lR == 'right', dadm$cov_left, dadm$cov_right))

  # Indicator matrices (note: both are added rather than subtracted)
  map_this <- sapply(cov_names, function(col) ifelse(lS == col, 1, 0))
  map_other <- sapply(cov_names, function(col) ifelse(lSother == col, 1, 0))

  map_this + map_other
}
```

```
trend <- make_trend(
  par_names = 'v',
  kernels   = 'delta',
  bases     = 'lin',
  cov_names = list(c('cov1', 'cov2', 'cov3', 'cov4')),
  maps      = list('differences' = advantage_map,
                    'sums'       = sum_map),
  at        = 'lR'
)
```

```

design_RDM <- design(
  model = RDM,
  data = data,
  formula= list(B ~ 1, v ~ 1, t0 ~ 1),
  trend = trend
)

emc <- make_emc(data, design_RDM, type = 'single')

# Now the dadm contains two covariate maps, and the model includes two
# corresponding base parameters (e.g., v.w1 and v.w2):
attr(emc[[1]]$data[[1]], 'covariate_maps')

```

mapped_pars

*Parameter Mapping Back to the Design Factors***Description**

Maps parameters of the cognitive model back to the experimental design. If `p_vector` is left unspecified will print a textual description of the mapping. Otherwise the `p_vector` can be created using `sampled_pars()`. The returned matrix shows whether/how parameters differ across the experimental factors.

Usage

```

mapped_pars(
  x,
  p_vector = NULL,
  model = NULL,
  digits = 3,
  remove_subjects = TRUE,
  covariates = NULL,
  ...
)

## S3 method for class 'emc.design'
mapped_pars(
  x,
  p_vector = NULL,
  model = NULL,
  digits = 3,
  remove_subjects = TRUE,
  covariates = NULL,
  ...
)

```

```
## S3 method for class 'emc.prior'
mapped_pars(
  x,
  p_vector = NULL,
  model = NULL,
  digits = 3,
  remove_subjects = TRUE,
  covariates = NULL,
  ...
)

## S3 method for class 'emc'
mapped_pars(
  x,
  p_vector = NULL,
  model = NULL,
  digits = 3,
  remove_subjects = TRUE,
  covariates = NULL,
  ...
)
```

Arguments

<code>x</code>	an emc, emc.prior or emc.design object
<code>p_vector</code>	Optional. Specify parameter vector to get numeric mappings. Must be in the form of <code>sampled_pars(design)</code>
<code>model</code>	Optional model type (if not already specified in design)
<code>digits</code>	Integer. Will round the output parameter values to this many decimals
<code>remove_subjects</code>	Boolean. Whether to include subjects as a factor in the design
<code>covariates</code>	Covariates specified in the design can be included here.
<code>...</code>	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_DDME <- 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)))

mapped_pars(design_DDME)
# 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_pars(design_DDME, p_vector)
```

merge_chains	<i>Merge Samples</i>
--------------	----------------------

Description

Merges samples from all chains as one unlisted object.

Usage

```
merge_chains(emc)
```

Arguments

emc	An emc object, commonly the output of <code>fit()</code>
-----	--

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

model_averaging	<i>Model Averaging</i>
-----------------	------------------------

Description

Computes model weights and a Bayes factor by comparing two groups of models based on their Information Criterion (IC) values. The function works with either numeric vectors or data frames containing multiple IC measures (e.g., MD, BPIC, DIC).

Usage

```
model_averaging(IC_for, IC_against)
```

Arguments

IC_for	A numeric vector or the output of <code>compare</code>
IC_against	A numeric vector or the output of <code>compare</code>

Details

When provided with numeric vectors, it computes the weights for the two groups by first converting the IC values into relative weights and then normalizing them. When provided with a data frame, it assumes that the data frame is the output of a call to `compare` and applies averaging to each IC metric

Value

A data.frame with the following columns:

`wFor` The aggregated weight of the models in favor.

`wAgainst` The aggregated weight of the models against.

`Factor` The Bayes factor (ratio of `wFor` to `wAgainst`).

If `IC_for` is a data frame, a matrix with rows corresponding to each IC measure is returned.

Examples

```
# First set up some example models (normally these would be alternative models)
samples_LNR2 <- subset(samples_LNR, length.out = 45)
samples_LNR3 <- subset(samples_LNR, length.out = 40)
samples_LNR4 <- subset(samples_LNR, length.out = 35)

# Run compare on them, BayesFactor = F is set for speed.
ICs <- compare(list(S1 = samples_LNR, S2 = samples_LNR2,
                    S3 = samples_LNR3, S4 = samples_LNR4), BayesFactor = FALSE)

# Model averaging can either be done with a vector of ICs:
model_averaging(ICs$BPIC[1:2], ICs$BPIC[2:4])

# Or the output of compare:
model_averaging(ICs[1:2,], ICs[3:4,])
```

MRI

GLM model for fMRI data

Description

Creates a model specification for fMRI data using a normal distribution. This model assumes that the observed BOLD signal follows a normal distribution with a mean determined by the design matrix and betas, and a standard deviation parameter for noise.

Usage

`MRI()`

Details

The model uses a normal distribution to model fMRI BOLD signals. Beta parameters represent the effect sizes for different conditions, and the sd parameter represents the standard deviation of the noise.

The log-likelihood function centers the predicted values by subtracting the mean, which helps with model identifiability.

Value

A list containing model specification

Examples

```
# Create a normal MRI model specification
model_spec <- MRI()

# Access model parameters
model_spec$p_types
```

MRI_AR1*Create an AR(1) GLM model for fMRI data*

Description

This function creates a model specification for MRI data with an AR(1) error structure. The model includes beta parameters for the design matrix, a rho parameter for the autocorrelation, and a standard deviation parameter for the noise.

Usage

```
MRI_AR1()
```

Details

The AR(1) model accounts for temporal autocorrelation in the data, where each timepoint is correlated with the previous timepoint according to the rho parameter.

Value

A list containing the model specifications

Examples

```
# Create an AR(1) GLM model for fMRI data
model_spec <- MRI_AR1()

# Access model parameters
model_spec$p_types
```

pairs_posterior	<i>Plot Within-Chain Correlations</i>
-----------------	---------------------------------------

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 dividing 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.prior *Return Data Frame of Parameters*

Description

Return Data Frame of Parameters

Usage

```
## S3 method for class 'emc.prior'
parameters(x, selection = "mu", N = 1000, covariates = NULL, ...)

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

parameters(x, ...)
```

Arguments

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

Value

A data frame with one row for each sample (with a subjects column if selection = "alpha" and using draws from the posterior)

Examples

```
# For prior inference:
# First set up a prior
design_DDME <- 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_DDME <- prior(design_DDME,mu_mean=p_vector,mu_sd=psd)
# Get our prior samples
parameters(prior_DDME, N = 100)
# For posterior inference:
# Get 100 samples of the group-level mean (the default)
parameters(samples_LNR, N = 100)
# or from the individual-level parameters and mapped
parameters(samples_LNR, selection = "alpha", map = TRUE)
```

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())

Details

If an emc object that has not been run with `fit` yet is supplied prior plots will be returned.

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.emc.design	<i>Plot method for emc.design objects</i>
-----------------	---

Description

Makes design illustration by plotting simulated data based on the design

Usage

```
## S3 method for class 'emc.design'
plot(
  x,
  p_vector,
  data = NULL,
  factors = NULL,
  plot_factor = NULL,
  n_data_sim = 10,
  functions = NULL,
  ...
)
```

Arguments

<code>x</code>	An object of class <code>emc.design</code> containing the design to plot
<code>p_vector</code>	A named vector of parameter values to use for data generation
<code>data</code>	Optional data frame to overlay on the design plot. If <code>NULL</code> , data will be simulated.
<code>factors</code>	Character vector. Factors to use for varying parameters in the plot
<code>plot_factor</code>	Optional character. Make separate plots for each level of this factor

n_data_sim	Integer. If data is NULL, number of simulated datasets to generate for the plot. Default is 10.
functions	Optional named list of functions that create additional columns in the data
...	Additional arguments passed to make_design_plot

Value

No return value, called for side effect of plotting

plot.emc.prior	<i>Plot a prior</i>
----------------	---------------------

Description

Takes a prior object and plots the selected implied prior

Usage

```
## S3 method for class 'emc.prior'
plot(
  x,
  selection = "mu",
  do_plot = TRUE,
  covariates = NULL,
  layout = NA,
  N = 5000,
  ...
)
```

Arguments

x	An emc_prior element
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 invisible mcmc.list object with prior samples of the selected type

Examples

```
# First define a design for the model
design_DDME <- 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_DDME <- prior(design_DDME,mu_mean=p_vector,mu_sd=psd)
# Now we can plot all sorts of (implied) priors
plot(prior_DDME, selection = "mu", N = 1e2)
plot(prior_DDME, selection = "mu", mapped = FALSE, N=1e2)
# We can also plot the implied prior on the participant level effects.
plot(prior_DDME, selection = "alpha", col = "green", N = 1e2)
```

plot_caf

Plot conditional accuracy functions

Description

Plots panels of conditional accuracy functions (CAFs, one for each level of `caf_factor` on the same panel). Accuracy is calculated with smoothing box car filter on percentile ranges, 0..X, 1..(X+1), ... , (100-X+1).. Inf, where $1 < X \leq 50$. Optionally, posterior and/or prior predictive CAFs can be overlaid.

Usage

```
plot_caf(
  input,
  post_predict = NULL,
  prior_predict = NULL,
  subject = NULL,
  quants = c(0.025, 0.975),
  functions = NULL,
  factors = NULL,
  caf_factor = NULL,
  n_cores = 1,
  n_post = 50,
  layout = NA,
  to_plot = c("data", "posterior", "prior")[1:2],
  use_lim = c("data", "posterior", "prior")[1:2],
  legendpos = c("bottomleft", "bottomright"),
  posterior_args = list(),
  prior_args = list(),
```

```

accuracy_function = function(d) d$S == d$R,
smooth_window = 5,
which_plot = 1:2,
...
)

```

Arguments

input	Either an emc object or a data frame, or a <i>list</i> of such objects.
post_predict	Optional posterior predictive data (matching columns) or <i>list</i> thereof.
prior_predict	Optional prior predictive data (matching columns) or <i>list</i> thereof.
subject	Subset the data to a single subject (by index or name).
quants	Numeric vector of credible interval bounds (e.g. <code>c(0.025, 0.975)</code>).
functions	A function (or list of functions) that create new columns in the datasets or predictives
factors	Character vector of factor names to aggregate over; defaults to plotting full data set ungrouped by factors if NULL.
caf_factor	The name of within-panel factor
n_cores	Number of CPU cores to use if generating predictives from an emc object.
n_post	Number of posterior draws to simulate if needed for predictives.
layout	Numeric vector used in <code>par(mfrow=...)</code> ; use NA for auto-layout.
to_plot	Character vector: any of "data", "posterior", "prior".
use_lim	Character vector controlling which source(s) define xlim.
legendpos	Character vector controlling the positions of the legends
posterior_args	Optional list of graphical parameters for posterior lines/ribbons.
prior_args	Optional list of graphical parameters for prior lines/ribbons.
accuracy_function	Accuracy score, default: <code>function(d) d\$S==d\$R</code> ,
smooth_window	range of RT over which calculate accuracy, default 5
which_plot	which of levels of caf_factor to plot, default is both i.e., <code>which_plot = 1:2</code>
...	Other graphical parameters for the real data lines.

Value

Returns NULL invisibly.

Examples

```

# Plot conditional accuracy function for data only,
# NB: the caf_factor must have two levels levels.
# forstmann_speed_accuracy <- forstmann[forstmann$E!="neutral",]
# forstmann_speed_accuracy$E <- droplevels(forstmann_speed_accuracy$E)
# plot_caf(forstmann_speed_accuracy, caf_factor="E", factors="S", smooth_window=10)
#
# Or a list of multiple emc objects ...

```

plot_cdf

*Plot Defective Cumulative Distribution Functions***Description**

Plots panels of cumulative distribution functions (CDFs) for each level of the specified defective factor in the data. The CDFs are *defective*; each factor level's CDF scales only up to that level's proportion. Summed across levels, the maximum is 1. Optionally, posterior and/or prior predictive CDFs can be overlaid.

Usage

```
plot_cdf(
  input,
  post_predict = NULL,
  prior_predict = NULL,
  subject = NULL,
  quants = c(0.025, 0.975),
  functions = NULL,
  factors = NULL,
  defective_factor = "R",
  n_cores = 1,
  n_post = 50,
  layout = NA,
  to_plot = c("data", "posterior", "prior")[1:2],
  use_lim = c("data", "posterior", "prior")[1:2],
  legendpos = c("topleft", "right"),
  posterior_args = list(),
  prior_args = list(),
  add_percentiles = c(10, 50, 90),
  ...
)
```

Arguments

input	Either an emc object or a data frame, or a <i>list</i> of such objects.
post_predict	Optional posterior predictive data (matching columns) or <i>list</i> thereof.
prior_predict	Optional prior predictive data (matching columns) or <i>list</i> thereof.
subject	Subset the data to a single subject (by index or name).
quants	Numeric vector of credible interval bounds (e.g. <code>c(0.025, 0.975)</code>).
functions	A function (or list of functions) that create new columns in the datasets or predictives
factors	Character vector of factor names to aggregate over; defaults to plotting full data set ungrouped by factors if NULL.

defective_factor	Name of the factor used for the defective CDF (default "R").
n_cores	Number of CPU cores to use if generating predictives from an emc object.
n_post	Number of posterior draws to simulate if needed for predictives.
layout	Numeric vector used in <code>par(mfrow=...)</code> ; use NA for auto-layout.
to_plot	Character vector: any of "data", "posterior", "prior".
use_lim	Character vector controlling which source(s) define xlim.
legendpos	Character vector controlling the positions of the legends
posterior_args	Optional list of graphical parameters for posterior lines/ribbons.
prior_args	Optional list of graphical parameters for prior lines/ribbons.
add_percentiles	Vector of integers giving percentiles to plot as points, NULL stops plotting.
...	Other graphical parameters for the real data lines.

Value

Returns NULL invisibly.

Examples

```
# Plot defective CDF for data only
# plot_cdf(forstmann, to_plot = "data")
#
# Plot with posterior predictions
# plot_cdf(samples_LNR, to_plot = c("data","posterior"), n_post=10)
#
# Or a list of multiple emc objects ...
```

plot_delta

Plot Difference of Cumulative Distribution Functions

Description

Plots panels of differences in cumulative distribution functions (CDFs) between conditions specified by the delta factor in the data. Optionally, posterior and/or prior predictive delta functions can be overlaid.

Usage

```
plot_delta(
  input,
  post_predict = NULL,
  prior_predict = NULL,
  subject = NULL,
  quants = c(0.025, 0.975),
```

```

    functions = NULL,
    factors = NULL,
    delta_factor = "R",
    n_cores = 1,
    n_post = 50,
    layout = NA,
    to_plot = c("data", "posterior", "prior")[1:2],
    use_lim = c("data", "posterior", "prior")[1:2],
    legendpos = c("topleft"),
    posterior_args = list(),
    prior_args = list(),
    add_percentiles = c(1:9) * 10,
    rev_delta = FALSE,
    ...
)

```

Arguments

input	Either an emc object or a data frame, or a <i>list</i> of such objects.
post_predict	Optional posterior predictive data (matching columns) or <i>list</i> thereof.
prior_predict	Optional prior predictive data (matching columns) or <i>list</i> thereof.
subject	Subset the data to a single subject (by index or name).
quants	Numeric vector of credible interval bounds (e.g. <code>c(0.025, 0.975)</code>).
functions	A function (or list of functions) that create new columns in the datasets or predictives
factors	Character vector of factor names to aggregate over; defaults to plotting full data set ungrouped by factors if NULL.
delta_factor	The name of the factor to delta
n_cores	Number of CPU cores to use if generating predictives from an emc object.
n_post	Number of posterior draws to simulate if needed for predictives.
layout	Numeric vector used in <code>par(mfrow=...)</code> ; use NA for auto-layout.
to_plot	Character vector: any of "data", "posterior", "prior".
use_lim	Character vector controlling which source(s) define xlim.
legendpos	Character vector controlling the positions of the legends
posterior_args	Optional list of graphical parameters for posterior lines/ribbons.
prior_args	Optional list of graphical parameters for prior lines/ribbons.
add_percentiles	Vector of integers giving percentiles to plot as points, NULL stops plotting.
rev_delta	If FALSE (the default) the first level of the defective factor is subtracted from the second, if TRUE this is reversed.
...	Other graphical parameters for the real data lines.

Value

Returns NULL invisibly.

Examples

```
# Plot delta function for data only, not that the delta_factor must have two
# levels.
# forsmann_speed_accuracy <- forsmann[forsmann$E!="neutral",]
# forsmann_speed_accuracy$E <- droplevels(forsmann_speed_accuracy$E)
# plot_delta(forsmann_speed_accuracy, to_plot = "data")
#
# Plot with posterior predictions
# plot_delta(samples_LNR, to_plot = c("data","posterior"), n_post=10)
#
# Or a list of multiple emc objects ...
```

plot_density

Plot Defective Densities

Description

Plots panels that contain a set of densities for each level of the specified defective factor in the data. These densities are defective; their areas are relative to the respective proportions of the defective factor levels. Across all levels, the area sums to 1. Optionally, posterior/prior predictive densities can be overlaid.

Usage

```
plot_density(
  input,
  post_predict = NULL,
  prior_predict = NULL,
  subject = NULL,
  quants = c(0.025, 0.975),
  functions = NULL,
  factors = NULL,
  defective_factor = "R",
  n_cores = 1,
  n_post = 50,
  layout = NA,
  to_plot = c("data", "posterior", "prior")[1:2],
  use_lim = c("data", "posterior", "prior")[1:2],
  legendpos = c("topright", "top"),
  posterior_args = list(),
  prior_args = list(),
  ...
)
```

Arguments

input Either an emc object or a data frame, or a *list* of such objects.

post_predict	Optional posterior predictive data (matching columns) or <i>list</i> thereof.
prior_predict	Optional prior predictive data (matching columns) or <i>list</i> thereof.
subject	Subset the data to a single subject (by index or name).
quants	Numeric vector of credible interval bounds (e.g. <code>c(0.025, 0.975)</code>).
functions	A function (or list of functions) that create new columns in the datasets or predictives
factors	Character vector of factor names to aggregate over; defaults to plotting full data set ungrouped by factors if NULL.
defective_factor	Name of the factor used for the defective CDF (default "R").
n_cores	Number of CPU cores to use if generating predictives from an emc object.
n_post	Number of posterior draws to simulate if needed for predictives.
layout	Numeric vector used in <code>par(mfrow=...)</code> ; use NA for auto-layout.
to_plot	Character vector: any of "data", "posterior", "prior".
use_lim	Character vector controlling which source(s) define xlim.
legendpos	Character vector controlling the positions of the legends
posterior_args	Optional list of graphical parameters for posterior lines/ribbons.
prior_args	Optional list of graphical parameters for prior lines/ribbons.
...	Other graphical parameters for the real data lines.

Examples

```
# Plot defective densities for each subject and the factor combination in the design:
plot_density(forstmann)
# or for one subject:
plot_density(forstmann, subject = 1)
# Now collapsing across subjects and using a different defective factor:
plot_density(forstmann, factors = "S", defective_factor = "E")
# Or plot posterior predictives
plot_density(samples_LNR, n_post = 10)
```

plot_design.emc.design

Plot Design

Description

Makes design illustration by plotting simulated data based on the design

Usage

```
## S3 method for class 'emc.design'
plot_design(
  x,
  data = NULL,
  factors = NULL,
  plot_factor = NULL,
  n_data_sim = 10,
  p_vector = NULL,
  functions = NULL,
  ...
)

## S3 method for class 'emc.prior'
plot_design(
  x,
  data = NULL,
  factors = NULL,
  plot_factor = NULL,
  n_data_sim = 10,
  p_vector = NULL,
  functions = NULL,
  ...
)

plot_design(
  x,
  data = NULL,
  factors = NULL,
  plot_factor = NULL,
  n_data_sim = 10,
  p_vector = NULL,
  functions = NULL,
  ...
)

## S3 method for class 'emc'
plot_design(
  x,
  data = NULL,
  factors = NULL,
  plot_factor = NULL,
  n_data_sim = 10,
  p_vector = NULL,
  functions = NULL,
  ...
)
```

Arguments

x	An emc or emc.prior object containing the design to plot
data	Optional data to overlay on the design plot
factors	Factors to use for varying parameters
plot_factor	Optional. Make separate plots for each level of this factor
n_data_sim	If data is provided, number of simulated datasets to generate for the plot. Default is 10.
p_vector	Only needed when x is an emc.design object, which parameters to use for data generation.
functions	A named list of functions that create additional columns in the data.
...	Additional arguments to pass to make_design_plot

Value

No return value. Just plots the design

plot_design_fmri	<i>Plot fMRI Design Matrix</i>
------------------	--------------------------------

Description

This function creates a visualization of an fMRI design matrix, showing the temporal evolution of regressors over time. It can handle various input formats and provides options to customize the visualization.

Usage

```
plot_design_fmri(
  design_matrix,
  TRs = 100,
  events = NULL,
  remove_nuisance = TRUE,
  subject = 1,
  legend_pos = "bottomleft",
  ...
)
```

Arguments

design_matrix	A design matrix for fMRI analysis. Can be a data frame, matrix, list of matrices, or an object of class 'emc.design'.
TRs	The number of time points (TRs) to plot. Default is 100.
events	A character vector specifying which regressors to plot. If NULL, all non-nuisance regressors will be plotted.

remove_nuisance	Logical indicating whether to remove nuisance regressors (drift terms, polynomial terms, derivatives) from the plot. Default is TRUE.
subject	The subject number to plot. Only applies for list of design matrices. Default is 1.
legend_pos	Position of the legend. Default is "bottomleft".
...	Additional graphical parameters passed to matplot and legend.

Value

A plot showing the design matrix regressors over time.

Examples

```
# Example time series
ts <- data.frame(
  subjects = rep(1, 100),
  run      = rep(1, 100),
  time     = seq(0, 99),
  ROI      = rnorm(100)
)
# Create a simple events data frame
events <- data.frame(
  subjects = rep(1, 10),
  run = rep(1, 10),
  onset = seq(0, 90, by = 10),
  condition = rep(c("A", "B"), 5),
  rt = runif(10, 0.5, 1.5),
  accuracy = sample(0:1, 10, replace = TRUE)
)
# Reshape with custom duration for each event_type
reshaped <- reshape_events(events,
  event_types = c("condition", "accuracy", "rt"),
  duration = list(condition = 0.5,
    accuracy = 0.2,
    rt = function(x) x$rt))
design_matrices <- convolve_design_matrix(
  timeseries = ts,
  events = reshaped,
  covariates = c('accuracy', 'rt'),
  factors = list(cond = c("condition_A", "condition_B")),
  contrasts = list(cond = matrix(c(-1, 1))))

# Plot the design matrix
plot_design_fmri(design_matrices)
```

plot_fmri

*Plot fMRI peri-stimulus time courses***Description**

This function plots average BOLD response around specified events for a single ROI by using FIR based event estimation, all event_types in events are taken into account in the FIR. Posterior predictives can be overlaid via the post_predict argument.

Usage

```
plot_fmri(
  timeseries,
  post_predict = NULL,
  events,
  event_type,
  high_pass = TRUE,
  high_pass_model = "cosine",
  posterior_args = list(),
  legend_pos = "topleft",
  layout = NA,
  n_cores = 1,
  ...
)
```

Arguments

timeseries	A data frame with columns 'subjects', 'run', 'time', and one ROI measurement column.
post_predict	Optional posterior predictive samples data frame (not shown in examples).
events	A data frame with columns 'subjects', 'run', 'onset', 'duration', 'event_type', and 'modulation'.
event_type	Character string specifying which event_type in events to plot.
high_pass	Logical indicating whether to apply high-pass filtering. Alternatively, specifying 'add' adds the regressors to the design matrix in the FIR. The choice here should be the same as the choice for convolve_design_matrix
high_pass_model	Character indicating which type of high-pass filtering to apply ('cosine', 'poly')
posterior_args	Named list of graphical parameters for posterior predictive lines.
legend_pos	Position of the legend. Default: "topleft".
layout	Panel layout matrix for multiple modulation groups. NULL leaves current layout
n_cores	Number of cores to calculate FIR across subjects with.
...	Additional graphical parameters passed to plotting functions (e.g., col, lwd, lty).

Value

NULL. Produces plots as a side-effect.

Examples

```
ts <- data.frame(
  subjects = rep(1, 100),
  run      = rep(1, 100),
  time     = seq(0, 99),
  ROI      = rnorm(100)
)
events <- data.frame(
  subjects = rep(1, 5),
  run      = rep(1, 5),
  onset    = c(10, 30, 50, 70, 90),
  event_type = rep("A", 5),
  modulation = rep(1, 5),
  duration  = rep(0.5, 5)
)
plot_fmri(ts, events = events, event_type = "A")
```

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_args = list(),
  true_args = list(),
  ...
)
```

Arguments

emc	An emc object
layout	A vector indicating which layout to use as in <code>par(mfrow = layout)</code> . 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 xlims 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_args	A list. Optional additional arguments to be passed to <code>plot.default</code> for the plotting of the prior density (see <code>par()</code>)
true_args	A list. Optional additional arguments to be passed to <code>plot.default</code> for the plotting of the true parameters (see <code>par()</code>)
...	Optional arguments that can be passed to <code>get_pars</code> , <code>density</code> , or <code>plot.default</code> (see <code>par()</code>)

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_args = list(col = 'blue'), adjust = 2)
```

plot_relations	<i>Plot Group-Level Relations</i>
----------------	-----------------------------------

Description

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

Usage

```
plot_relations(
  emc = NULL,
  stage = "sample",
  plot_cred = FALSE,
  plot_means = TRUE,
  only_cred = TRUE,
  nice_names = NULL,
  selection = "correlation",
  use_par = 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
selection	Character. Whether to plot correlations or loadings
use_par	Character. Which parameters to include. If null, includes all.
...	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	<i>Plot the ECDF Difference in SBC Ranks</i>
---------------	--

Description

Plots $F_{\hat{}}(z) - z$ where $F_{\hat{}}$ is the empirical CDF of the (normalized) ranks and z is the Uniform(0,1) CDF. The shaded band is the simultaneous $(1 - \gamma)$ envelope for $F_{\hat{}}(z) - z$.

Usage

```
plot_sbc_ecdf(ranks, layout = NA, add_stats = TRUE, main = NULL, K = 500)
```

Arguments

ranks	A list of named dataframes of the rank statistic (raw or normalized)
layout	Optional. A numeric vector specifying the layout using <code>par(mfrow = layout)</code>
add_stats	Boolean. Should coverage, bias and precision be included in the figure.
main	Optional. A character specifying plot title.
K	Optional. Effective sample size of the MCMC that produced the ranks.

Value

No returns

plot_sbc_hist	<i>Plot the Histogram of the Observed Rank Statistics of SBC</i>
---------------	--

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, add_stats = TRUE)
```

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 <code>par(mfrow = layout)</code>
add_stats	Boolean. Should coverage, bias and precision be included in the figure.

Value

No returns

plot_spectrum

*Plot Empirical and Posterior Predictive Power Spectra***Description**

Computes and plots the empirical power spectrum, with optional overlay of spectra from posterior predictive simulations. All customization of the spectral estimator is done through `spectrum.args`, while plot appearance is controlled via `...`.

Usage

```
plot_spectrum(
  dat,
  pp = NULL,
  plot.log = TRUE,
  spectrum.args = list(),
  trial_duration = NULL,
  ...
)
```

Arguments

<code>dat</code>	A data frame containing empirical reaction time data, with at least columns <code>subjects</code> and <code>rt</code> .
<code>pp</code>	Optional posterior predictive data in the same format as <code>dat</code> , including <code>subjects</code> , <code>postn</code> , and <code>trials</code> .
<code>plot.log</code>	Logical. Whether to log-transform frequencies and power before plotting. This does not affect the call to spectrum : use <code>spectrum.args\$list(log = TRUE)</code> to request log spectral density from the estimator itself.
<code>spectrum.args</code>	A named list of arguments forwarded directly to spectrum inside get_power_spectra .
<code>trial_duration</code>	Optional duration of a trial in seconds. If supplied, the x-axis is labeled in human-readable time units. Otherwise the x-axis is in (log) frequencies of 1/trial.
<code>...</code>	Additional graphical parameters passed to plot .

Value

Invisibly returns a list containing the empirical spectrum and, if posterior predictive data is supplied, the posterior predictive spectra and their mean.

plot_stat

*Plot Statistics on Data***Description**

Plots panels that contain a set of densities for each level of the specified factor. The densities represent the predicted data across the posterior, the vertical lines represent the real data.

Usage

```
plot_stat(
  input,
  post_predict = NULL,
  prior_predict = NULL,
  stat_fun,
  stat_name = NULL,
  subject = NULL,
  factors = NULL,
  n_cores = 1,
  n_post = 50,
  quants = c(0.025, 0.5, 0.975),
  functions = NULL,
  layout = NA,
  to_plot = c("data", "posterior", "prior")[1:2],
  use_lim = c("data", "posterior", "prior")[1:2],
  legendpos = c("topleft", "top"),
  posterior_args = list(),
  prior_args = list(),
  ...
)
```

Arguments

input	Either an emc object or a data frame, or a <i>list</i> of such objects.
post_predict	Optional posterior predictive data (matching columns) or <i>list</i> thereof.
prior_predict	Optional prior predictive data (matching columns) or <i>list</i> thereof.
stat_fun	A function that can be applied to the data and returns a single value or a vector of values.
stat_name	The name of the calculated quantity
subject	Subset the data to a single subject (by index or name).
factors	Character vector of factor names to aggregate over; defaults to plotting full data set ungrouped by factors if NULL.
n_cores	Number of CPU cores to use if generating predictives from an emc object.
n_post	Number of posterior draws to simulate if needed for predictives.

quants	Numeric vector of credible interval bounds (e.g. <code>c(0.025, 0.975)</code>).
functions	A function (or list of functions) that create new columns in the datasets or predictives
layout	Numeric vector used in <code>par(mfrow=...)</code> ; use NA for auto-layout.
to_plot	Character vector: any of "data", "posterior", "prior".
use_lim	Character vector controlling which source(s) define xlim.
legendpos	Character vector controlling the positions of the legends
posterior_args	Optional list of graphical parameters for posterior lines/ribbons.
prior_args	Optional list of graphical parameters for prior lines/ribbons.
...	Other graphical parameters for the real data lines.

Value

an invisible data frame with the stat applied to the real data, posterior predictives and/or prior predictives

Examples

```
# For example plot the observed and predicted response accuracy
# Can also apply more sophisticated statistics
drt <- function(data) diff(tapply(data$rt,data[,c("E")],mean))
plot_stat(samples_LNR, stat_fun = drt, n_post = 10, stat_name = "RT diff Speed - A/N")
```

plot_trend	<i>Plots trends over time</i>
------------	-------------------------------

Description

Plots the trend for selected parameters of a model. Can be used either with a `p_vector`, or trial-wise parameters or covariates obtained from `predict()`

Usage

```
plot_trend(
  input_data,
  emc,
  par_name,
  subject = 1,
  filter = NULL,
  on_x_axis = "trials",
  pp_shaded = TRUE,
  ...
)
```

Arguments

input_data	a p_vector or posterior predictives compatible with the provided emc object
emc	An emc object
par_name	Parameter name (or covariate name) to plot
subject	Subject number to plot
filter	Optional function that takes a data frame and returns a logical vector indicating which rows to include in the plot
on_x_axis	Column name in the dadm to plot on the x-axis. By default 'trials'.
pp_shaded	Boolean. If TRUE will plot 95% credible interval as a shaded area. Otherwise plots separate lines for each iteration of the posterior predictives. Only applicable if input_data are posterior predictives.
...	Optional arguments that can be passed to plots.

Value

A trend plot

Examples

```

dat <- EMC2:::add_trials(forstmann)
dat$trials2 <- dat$trials/1000

lin_trend <- make_trend(cov_names='trials2',
                        kernels = 'exp_incr',
                        par_names='B',
                        bases='lin',
                        phase = "premap")

design_RDM_lin_B <- design(model=RDM,
                           data=dat,
                           covariates='trials2', # specify relevant covariate columns
                           matchfun=function(d) d$S==d$I,
                           transform=list(func=c('B'='identity')),
                           formula=list(B ~ 1, v ~ 1M, t0 ~ 1),
                           trend=lin_trend)      # add trend

emc <- make_emc(dat, design=design_RDM_lin_B, compress = FALSE)
p_vector <- c('B'=1, 'v'=1, 'v_1MTRUE'=1, 't0'=0.1, 'B.w'=1, 'B.d_ei'=1)

# Visualize trend
plot_trend(p_vector, emc=emc,
            par_name='B', subject='as1t',
            filter=function(d) d$I==1, main='Threshold for right')

```

predict.emc.prior	<i>Generate Posterior/Prior Predictives</i>
-------------------	---

Description

Simulate `n_post` data sets using the posterior/prior parameter estimates

Usage

```
## S3 method for class 'emc.prior'
predict(object, data = NULL, n_post = 50, n_cores = 1, n_trials = NULL, ...)

## S3 method for class 'emc'
predict(
  object,
  hyper = FALSE,
  n_post = 50,
  n_cores = 1,
  stat = c("random", "mean", "median")[1],
  ...
)
```

Arguments

<code>object</code>	An emc or emc.prior object from which to generate predictives
<code>data</code>	A data frame needed to exactly match the original design
<code>n_post</code>	Integer. Number of generated datasets
<code>n_cores</code>	Integer. Number of cores across which there should be parallellized
<code>n_trials</code>	An integer. If data isn't provided (although preferred), can generate data based on <code>n_trials</code> per cell of design
<code>...</code>	Optional additional arguments passed to <code>get_pars</code> or <code>make_data</code>
<code>hyper</code>	Boolean. Defaults to FALSE. If TRUE, simulates from the group-level (hyper) parameters instead of the subject-level parameters.
<code>stat</code>	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.

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 = 2)
```

prior

*Specify Priors for the Chosen Model***Description**

These values are entered manually by default but can be recycled from another prior (given in the update argument).

Usage

```
prior(
  design,
  type = NULL,
  group_design = NULL,
  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
group_design	An emc.group_design object created with group_design()
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_DDME <- 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_DDME <- prior(design_DDME,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_DDME <- prior(design_DDME,mu_mean=p_vector,mu_sd=psd, A = pscale, df = df)
# If we specify a new design
design_DDME0 <- 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_DDME0 <- prior(design_DDME0, update = prior_DDME)
```

prior_help

Prior Specification Information

Description

Prints information associated with the prior for certain 'type'

Usage

```
prior_help(type)
```

Arguments

type	A character string indicating which 'type' of model to run (e.g. 'standard' or 'single')
------	--

Value

Invisible return with a list of all the information that is also printed

Examples

```
prior_help('diagonal')
```

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_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 <code>sampled_pars(design)</code>)
range	Numeric. The max and min will be $p_vector + range/2$ and $p_vector - range/2$, unless specified in <code>p_min</code> or <code>p_max</code> .
layout	A vector indicating which layout to use as in <code>par(mfrow = layout)</code> . 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_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	<i>The Racing Diffusion Model</i>
-----	-----------------------------------

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
<i>v</i>	log	[0, Inf]	log(1)		Evidence-accumulation rate (drift rate)
<i>A</i>	log	[0, Inf]	log(0)		Between-trial variation (range) in start point
<i>B</i>	log	[0, Inf]	log(1)	$b = B + A$	Distance from <i>A</i> to <i>b</i> (response threshold)
<i>t0</i>	log	[0, Inf]	log(0)		Non-decision time
<i>s</i>	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 1R (i.e., the latent response) with level names taken from the R column in the data.

The 1R 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 $B \sim 1R$ 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 1R 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$1R`. Using `matchfun` a latent match factor (1M) 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 1M it is useful to design an "average and difference"
# contrast matrix, which for binary responses has a simple canonical form:
ADmat <- matrix(c(-1/2,1/2),ncol=1,dimnames=list(NULL,"d"))
# We also define a match function for 1M
matchfun=function(d)d$S==d$1R
# We now construct our design, with v ~ 1M and the contrast for 1M the ADmat.
design_RDMBE <- design(data = forstmann,model=RDM,matchfun=matchfun,
                      formula=list(v~1M,s~1M,B~E+1R,A~1,t0~1),
                      contrasts=list(v=list(1M=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

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

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"))
```

register_trend

Register a custom C++ trend kernel

Description

Compiles and registers a user-provided C++ function that maps per-trial kernel parameters and inputs to a numeric vector. The C++ function must have signature: `NumericVector f(NumericMatrix trend_pars, NumericMatrix input)` and provide an exported pointer creator using `EMC2_MAKE_PTR`.

Usage

```
register_trend(trend_parameters, file, transforms = NULL, base = "add")
```

Arguments

trend_parameters	Character vector of kernel parameter names (in order).
file	Path to the C++ file implementing the custom kernel. The file should include <code>EMC2/userfun.hpp</code> and define a pointer creator (via <code>EMC2_MAKE_PTR</code>) that is exported to R.
transforms	Optional named character vector or list mapping each custom kernel parameter name to a transform name (e.g., "identity", "exp", "pnorm"). Length must match <code>trend_parameters</code> . If unnamed but the correct length, the order is assumed to match <code>trend_parameters</code> .
base	Default base to use when creating trends with this custom kernel if no bases argument is supplied to <code>make_trend</code> . One of <code>c("lin", "exp_lin", "centered", "add", "identity")</code> . Default "add".

Value

An object to pass to `make_trend(custom_trend=...)`, carrying the pointer, parameter names, default base, and optional transform mapping.

reshape_events	<i>Reshape events data for fMRI analysis</i>
----------------	--

Description

This function reshapes event data into a format suitable for fMRI analysis by converting specified `event_types` into separate event types with appropriate modulation values.

Usage

```
reshape_events(events, event_types, duration = 0.001, modulation = NULL)
```

Arguments

<code>events</code>	A data frame containing event information with required columns 'subjects', 'run', and 'onset'
<code>event_types</code>	A character vector of column names in the events data frame to be treated as event_types
<code>duration</code>	Either a single numeric value (applied to all event_types), a list with named elements corresponding to event_types, or a function that takes the events data frame and returns durations
<code>modulation</code>	Either a list with named elements corresponding to event_types, or a function that takes the events data frame and returns durations

Value

A data frame with columns 'subjects', 'onset', 'run', 'modulation', 'duration', and 'event_type'

Examples

```
# Create a simple events data frame
events <- data.frame(
  subjects = rep(1, 10),
  run = rep(1, 10),
  onset = seq(0, 90, by = 10),
  condition = rep(c("A", "B"), 5),
  rt = runif(10, 0.5, 1.5),
  accuracy = sample(0:1, 10, replace = TRUE)
)

# Reshape with default duration
reshaped1 <- reshape_events(events, event_types = c("condition", "accuracy"))
```

```
# Reshape with custom duration for each event_type
reshaped2 <- reshape_events(events,
                             event_types = c("condition", "accuracy", "rt"),
                             duration = list(condition = 0.5,
                                              accuracy = 0.2,
                                              rt = function(x) x$rt))
```

rotate_loadings	<i>Rotate loadings based on posterior median</i>
-----------------	--

Description

This function rotates factor loadings using a rotation function based on the posterior median.

Usage

```
rotate_loadings(emc, rot_fun)
```

Arguments

emc	An 'emc' object containing factor analysis results
rot_fun	a rotation function for factor loadings, see also GPARotationW

Value

An 'emc' object with rotated factor loadings

run_bridge_sampling	<i>Estimating Marginal Likelihoods Using WARP-III Bridge Sampling</i>
---------------------	---

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 $\text{cores_for_props} * \text{cores_per_prop}$. 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 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

```
# 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(samples_LNR, cores_for_props = 1, both_splits = FALSE)
# This will run on 2*4 cores (since 4 is the default for ``cores_for_props``)
```

run_emc

Fine-Tuned 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,
  search_width = 1,
  step_size = 100,
  verbose = FALSE,
  verboseProgress = FALSE,
  fileName = NULL,
  particle_factor = 50,
  cores_per_chain = 1,
  cores_for_chains = length(emc),
  max_tries = 20,
  n_blocks = 1,
  thin = FALSE,
  trim = TRUE,
  r_cores = 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 <code>?fit</code> .

search_width	A double. Tunes target acceptance probability of the MCMC process. This fine-tunes the width of the search space to obtain the desired acceptance probability. 1 is the default width, increases lead to broader search.
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.
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.
thin	A boolean. If TRUE will automatically thin the MCMC samples, closely matched to the ESS. Can also be set to a double, in which case 1/thin of the chain will be removed (does not have to be an integer).
trim	A boolean. If TRUE will automatically remove redundant samples (i.e. from preburn, burn, adapt).
r_cores	An integer for number of cores to use in R-based likelihood calculations, default 1.

Value

An emc object

Examples

```
# First define a design
design_in <- design(data = forstmann,model=DDM,
                   formula =list(v~0+S,a~E, t0~1, s~1, Z~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_in, compress = FALSE)

# Now for example we can specify that we only want to run the "preburn" phase
# for MCMC 10 iterations
# emc <- run_emc(emc, stage = "preburn", stop_criteria = list(iter = 10), cores_for_chains = 1)

```

run_hyper

Run a Group-level Model.

Description

Separate function for running only the group-level model. This can be useful in a two-step analysis. Works similar in functionality to `make_emc`, except also does the fitting and returns an `emc` object that works with most posterior checking tests (but not the data generation/posterior predictives).

Usage

```

run_hyper(
  type = "standard",
  data,
  prior = NULL,
  iter = 1000,
  n_chains = 3,
  ...
)

```

Arguments

<code>type</code>	A string indicating whether to run a standard group-level, blocked, diagonal, factor, or single (i.e., non-hierarchical) model.
<code>data</code>	A data frame, or a list of data frames. Needs to have the variable subjects as participant identifier.
<code>prior</code>	an <code>emc.prior</code> object.
<code>iter</code>	Number of MCMC samples to collect.
<code>n_chains</code>	An integer. Specifies the number of mcmc chains to be run (has to be more than 1 to compute rhat).
<code>...</code>	Additional, optional arguments.

Value

an `emc` object with only group-level samples

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 <code>fit</code> and <code>make_emc</code>

Value

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

sampled_pars*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_pars(  
  x,  
  group_design = NULL,  
  doMap = FALSE,  
  add_da = FALSE,  
  all_cells_dm = FALSE,  
  data = NULL  
)  
  
## S3 method for class 'emc.design'  
sampled_pars(  
  x,  
  group_design = NULL,  
  doMap = FALSE,  
  add_da = FALSE,  
  all_cells_dm = FALSE,  
  data = NULL  
)  
  
## S3 method for class 'emc.group_design'  
sampled_pars(  
  x,  
  group_design = NULL,  
  doMap = FALSE,  
  add_da = FALSE,  
  all_cells_dm = FALSE,  
  data = NULL  
)  
  
## S3 method for class 'emc.prior'  
sampled_pars(  
  x,  
  group_design = NULL,  
  doMap = FALSE,  
  add_da = FALSE,  
  all_cells_dm = FALSE,  
  data = NULL
```

```

)

## S3 method for class 'emc'
sampled_pars(
  x,
  group_design = NULL,
  doMap = FALSE,
  add_da = FALSE,
  all_cells_dm = FALSE,
  data = NULL
)

```

Arguments

<code>x</code>	an <code>emc.design</code> object made with <code>design()</code> or an <code>emc</code> object.
<code>group_design</code>	an <code>emc.group_design</code> object made with <code>group_design()</code>
<code>doMap</code>	logical. If TRUE will also include an attribute map with the design matrices that perform the mapping back to the design
<code>add_da</code>	Boolean. Whether to include the relevant data columns in the map attribute
<code>all_cells_dm</code>	Boolean. Whether to include all levels of a factor in the mapping attribute, even when one is dropped in the design
<code>data</code>	A data frame to be included for accurate covariate mapping in <code>summary.design</code>

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_pars(design_DDMaE)

```

samples_LNR

LNR Model of Forstmann Data (First 3 Subjects)

Description

An `emc` object with a limited number of samples and subjects of the Forstmann dataset. The object is a nested list of length 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

model A list containing the model functions

nuisance A logical vector indicating which parameters are nuisance parameters

subjects A vector containing the unique subject ID's

type The type of model e.g., "standard" or "diagonal"

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)

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

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

Source

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

SDT

*Gaussian Signal Detection Theory Model for Binary Responses***Description**

Discrete binary choice based on continuous Gaussian latent, with no rt (rt must be set to NA in data).

Usage

```
SDT()
```

Details

Model parameters are: mean (unbounded) sd (log scale) and threshold (unbounded).

For identifiability in one condition two parameters must be fixed (conventionally mean=0 and sd = 1). When used with data that records only accuracy (so reponse bias cannot be evaluated) a single threshold must be assumed and fixed (e.g., threshold = 0).

At present this model is not fully implemented in C, but as its likelihood requires only pnorm evaluation it is quite fast.

Value

A model list with all the necessary functions to sample

Examples

```
dprobit <- design(Rlevels = c("left","right"),
  factors=list(subjects=1,S=c("left","right")),
  formula=list(mean ~ 0+S, sd ~ 1,threshold ~ 1),
  matchfun=function(d)d$S==d$lR,
  constants=c(sd=log(1),threshold=0),
  model=SDT)

p_vector <- sampled_pars(dprobit)
```

set_custom_kernel_pointers

*(Re-)Set pointers of custom C++ trend kernels to an emc object***Description**

When an emc object is loaded from disk, or returned by forked processes, the pointers to custom kernels need to be re-created. This is a convenience function to do this.

Usage

```
set_custom_kernel_pointers(emc, ptrs)
```

Arguments

emc	An emc object
ptrs	A list of pointers, as generated by <code>get_custom_kernel_pointers()</code>

Value

An emc object with the custom pointers re-instated.

split_timeseries	<i>Split fMRI Timeseries Data by ROI Columns</i>
------------------	--

Description

This function splits a timeseries data frame containing multiple ROI columns into a list of data frames, where each data frame contains the common columns (subjects, run, time) and one ROI column.

Usage

```
split_timeseries(timeseries, columns = NULL)
```

Arguments

timeseries	A data frame containing fMRI timeseries data with required columns 'subjects', 'run', and 'time', plus one or more ROI columns.
columns	A character vector specifying which columns to split by. If NULL (default), all columns except 'subjects', 'run', and 'time' will be used.

Value

A named list of data frames, where each data frame contains the common columns (subjects, run, time) and one ROI column. The names of the list elements correspond to the ROI column names.

Examples

```
# Create a simple example timeseries with multiple ROIs
set.seed(123)
n_frames <- 100

# Create a data frame with multiple ROIs
timeseries <- data.frame(
  subjects = rep(1, n_frames),
  run = rep(1, n_frames),
  time = seq(0, n_frames-1),
```

```

ROI1 = rnorm(n_frames),
ROI2 = rnorm(n_frames),
ROI3 = rnorm(n_frames)
)

# Split the timeseries by all ROI columns
split_data <- split_timeseries(timeseries)

```

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

<code>x</code>	an emc object
<code>stage</code>	A character string. Indicates from which sampling stage(s) to take the samples from (i.e. preburn, burn, adapt, sample)
<code>filter</code>	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.
<code>thin</code>	An integer. By how much to thin the chains
<code>keep_stages</code>	Boolean. If TRUE, will not remove samples from unselected stages.
<code>length.out</code>	Integer. Alternatively to thinning, you can also select a desired length of the MCMC chains, which will be thinned appropriately.
<code>...</code>	additional optional arguments

Value

A shortened emc object

Examples

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

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.

If an emc object that has not been run with fit yet is supplied, summary of the design will be returned.

Value

A list of summary output.

summary.emc.design	<i>Summary method for emc.design objects</i>
--------------------	--

Description

Prints a summary of the design object, including sampled parameters and design matrices. For continuous covariates just prints one row, instead of all covariates.

Usage

```
## S3 method for class 'emc.design'
summary(object, ...)
```

Arguments

object	An object of class emc.design containing the design to summarize
...	Additional arguments (not used)

Value

Invisibly returns the design matrices

summary.emc.group_design	<i>Summary method for emc.group_design objects</i>
--------------------------	--

Description

Prints a summary of the group_design object For continuous covariates just prints one row, instead of all covariates.

Usage

```
## S3 method for class 'emc.group_design'
summary(object, ...)
```

Arguments

object	An object of class emc.group_design containing the design to summarize
...	Additional arguments (not used)

Value

Invisibly returns the design matrices

summary.emc.prior	<i>Summary method for emc.prior objects</i>
-------------------	---

Description

Prints a summary of the prior specification, including descriptions of the prior types and their associated hyperparameters.

Usage

```
## S3 method for class 'emc.prior'
summary(object, ...)
```

Arguments

object	An object of class 'emc.prior' containing prior specifications
...	Additional arguments passed to other methods (not currently used)

Value

Invisibly returns NULL. Called for its side effect of printing the summary.

See Also

[prior](#) for creating prior objects

Examples

```
# Take a prior object
prior <- get_prior(samples_LNR)
summary(prior)
```

trend_help	<i>Get help information for trend kernels and bases</i>
------------	---

Description

Get help information for trend kernels and bases

Usage

```
trend_help(kernel = NULL, base = NULL, ...)
```

Arguments

kernel	Character string specifying the kernel type to get information about
base	Character string specifying the base type to get information about
...	Additional arguments

Value

Formatted trend information

Examples

```
# Get information about exponential increasing kernel
trend_help(kernel = "exp_incr")

# Get information about linear base
trend_help(base = "lin")

# Return available kernel and base types
trend_help()
```

update2version

Update EMC Objects to the Current Version

Description

This function updates EMC objects created with older versions of the package to be compatible with the current version.

Usage

```
update2version(emc)
```

Arguments

emc	An EMC object to update
-----	-------------------------

Value

An updated EMC object compatible with the current version

Examples

```
# Update the model to current version
updated_model <- update2version(samples_LNR)
```

Index

* datasets

- forstmann, [29](#)
- samples_LNR, [102](#)
- align_loadings, [4](#)
- apply_kernel, [6](#)
- auto_thin (auto_thin.emc), [7](#)
- auto_thin.emc, [7](#)
- chain_n, [8](#)
- check (check.emc), [8](#)
- check.emc, [8](#)
- compare, [9](#)
- compare_subject, [11](#)
- contr.anova, [12](#)
- contr.bayes, [12](#)
- contr.decreasing, [13](#)
- contr.increasing, [14](#)
- convolve_design_matrix, [14](#)
- credible (credible.emc), [16](#)
- credible.emc, [16](#)
- credint (credint.emc.prior), [18](#)
- credint.emc.prior, [18](#)
- cut_factors, [19](#)
- DDM, [19](#)
- DDMGNG, [20](#)
- design, [21](#)
- design_fmri, [23](#)
- ess_summary (ess_summary.emc), [24](#)
- ess_summary.emc, [24](#)
- factor_diagram, [25](#)
- fit (fit.emc), [26](#)
- fit.emc, [26](#)
- fix_custom_kernel_pointers, [29](#)
- forstmann, [29](#)
- gd_summary (gd_summary.emc), [30](#)
- gd_summary.emc, [30](#)
- get_BayesFactor, [31](#)
- get_custom_kernel_pointers, [32](#)
- get_data (get_data.emc), [32](#)
- get_data.emc, [32](#)
- get_design (get_design.emc.prior), [33](#)
- get_design.emc.prior, [33](#)
- get_group_design
(get_group_design.emc.prior),
[34](#)
- get_group_design.emc.prior, [34](#)
- get_pars, [34](#)
- get_power_spectra, [36](#), [82](#)
- get_prior (get_prior.emc), [37](#)
- get_prior.emc, [37](#)
- get_trend_pnames, [38](#)
- group_design, [38](#)
- high_pass_filter, [39](#)
- hypothesis (hypothesis.emc), [40](#)
- hypothesis.emc, [40](#)
- init_chains, [42](#)
- LBA, [43](#)
- LNR, [44](#)
- make_data, [46](#)
- make_emc, [47](#)
- make_random_effects, [49](#)
- make_SEM_diagram, [50](#)
- make_sem_structure, [51](#)
- make_trend, [22](#), [53](#)
- mapped_pars, [57](#)
- merge_chains, [59](#)
- model_averaging, [59](#)
- MRI, [60](#)
- MRI_AR1, [61](#)
- pairs_posterior, [62](#)
- parameters (parameters.emc.prior), [63](#)
- parameters.emc.prior, [63](#)

plot, 82
plot.emc, 64
plot.emc.design, 65
plot.emc.prior, 66
plot_caf, 67
plot_cdf, 69
plot_delta, 70
plot_density, 72
plot_design(plot_design.emc.design), 73
plot_design.emc.design, 73
plot_design_fmri, 75
plot_fmri, 77
plot_pars, 78
plot_relations, 80
plot_sbc_ecdf, 81
plot_sbc_hist, 81
plot_spectrum, 82
plot_stat, 83
plot_trend, 84
predict.emc(predict.emc.prior), 86
predict.emc.prior, 86
prior, 87, 109
prior_help, 88
profile_plot, 89

RDM, 90
recovery(recovery.emc), 92
recovery.emc, 92
register_trend, 93
reshape_events, 94
rotate_loadings, 95
run_bridge_sampling, 95
run_emc, 97
run_hyper, 99
run_sbc, 100

sampled_pars, 101
samples_LNR, 102
SDT, 104
set_custom_kernel_pointers, 104
spectrum, 36, 37, 82
split_timeseries, 105
subset.emc, 106
summary.emc, 107
summary.emc.design, 108
summary.emc.group_design, 108
summary.emc.prior, 109

trend_help, 109
update2version, 110