Package ‘ggdmc’

April 29, 2019

Type Package
Title Cognitive Models
Version 0.2.6.0
Date 2019-04-29
Author Yi-Shin Lin [aut, cre], Andrew Heathcote [aut]
Maintainer Yi-Shin Lin <yishinlin001@gmail.com>
Description Hierarchical Bayesian models. The package provides tools to fit two response time models, using the population-based Markov Chain Monte Carlo.
License GPL-2
URL https://github.com/yxlin/ggdmc
BugReports https://github.com/yxlin/ggdmc/issues
LazyData TRUE
Imports Rcpp (>= 0.12.10), coda, stats, utils, ggplot2, matrixStats, data.table (>= 1.10.4)
Depends R (>= 3.3.0)
LinkingTo Rcpp (>= 0.12.10), RcppArmadillo (>= 0.7.100.3.0)
Suggests testthat
RoxygenNote 6.1.1
Encoding UTF-8
NeedsCompilation yes
Repository CRAN
Date/Publication 2019-04-29 05:10:03 UTC

R topics documented:

BuildDMI ............................................. 2
BuildModel ........................................... 3
BuildPrior ............................................ 4
check_pvec ........................................... 5
BuildDMI

**Description**

Binding a data set with a model object. The function also checks whether they are compatible and adds attributes on a data model instance.
Usage

BuildDMI(x, model)

Arguments

x  data as in data frame
model  a model object

Value

a data model instance

BuildModel  Create a model object

Description

A model object consists of arrays with model attributes.

Usage

BuildModel(p.map, responses, factors = list(A = "1"), match.map = NULL,
    constants = numeric(0), type = "norm", posdrift = TRUE,
    verbose = TRUE)

## S3 method for class 'model'
print(x, p.vector = NULL, ...)

## S3 method for class 'dmi'
print(x, ...)

Arguments

p.map  parameter map. This option maps a particular factorial design to model parameters
responses  specifying the response names and levels
factors  specifying a list of factors and their levels
match.map  match map. This option matches stimuli and responses
constants  specifying the parameters with fixed values
type  specifying model type, either "rd" or "norm".
posdrift  a Boolean, switching between enforcing strict positive drift rates by using truncated normal distribution. This option is only useful in "norm" model type.
verbose  Print p.vector, constants and model type
x  a model object
p.vector  parameter vector
...  other arguments
Examples

```r
model <- BuildModel(
  p.map = list(a = "1", v = "1", z = "1", d = "1", t0 = "1",
               sv = "1", sz = "1", st0 = "1"),
  constants = c(st0 = 0, d = 0, sz = 0, sv = 0),
  match.map = list(M = list(s1 = "r1", s2 = "r2")),
  factors = list(S = c("s1", "s2")),
  responses = c("r1", "r2"),
  type = "rd")
```

BuildPrior  Specifying Parameter Prior Distributions

Description

BuildPrior sets up parameter prior distributions for each model parameter. \( p_1 \) and \( p_2 \) refer to the first and second parameters a prior distribution.

Usage

```r
BuildPrior(p1, p2, lower = rep(NA, length(p1)), upper = rep(NA,
               length(p1)), dists = rep("tnorm", length(p1)),
               untrans = rep("identity", length(p1)), types = c("tnorm", "beta",
               "gamma", "lnorm", "unif", "constant", "tnorm2", NA))
```

Arguments

- \( p_1 \) the first parameter of a distribution
- \( p_2 \) the second parameter of a distribution
- lower lower support (boundary)
- upper upper support (boundary)
- dists a vector of character string specifying a distribution.
- untrans whether to do log transformation. Default is not
- types available distribution types

Details

Four distribution types are implemented:

1. Normal and truncated normal, where: \( p_1 = \text{mean} \) and \( p_2 = \text{sd} \). It specifies a normal distribution when bounds are set -Inf and Inf.
2. Beta, where: \( p_1 = \text{shape1} \) and \( p_2 = \text{shape2} \) (see \texttt{pbeta}). Note the uniform distribution is a special case of the beta with \( p_1 \) and \( p_2 = 1 \).
3. Gamma, where \( p_1 = \text{shape} \) and \( p_2 = \text{scale} \) (see \texttt{pgamma}). Note \( p_2 \) is scale, not rate.
4. Lognormal, where \( p_1 = \text{meanlog} \) and \( p_2 = \text{sdlog} \) (see \texttt{plnorm}).
check_pvec

Value

a list of list

Description

Check a parameter vector

Usage

check_pvec(p.vector, model)

Arguments

p.vector parameter vector
model a model object

ConvertChains

Prepare posterior samples for plotting functions version 1

Description

Convert MCMC chains to a data frame for plotting functions

Usage

ConvertChains(x, start = 1, end = NA, pll = TRUE)

Arguments

x posterior samples
start which iteration to start
end end at which iteration
pll a Boolean switch to make posterior log likelihood
dbeta_lu \hspace{1cm} \textit{A modified dbeta function}

\textbf{Description}

A modified dbeta function

\textbf{Usage}

dbeta_lu(x, p1, p2, lower, upper, lg = FALSE)

\textbf{Arguments}

- \textit{x}: quantile
- \textit{p1}: shape1 parameter
- \textit{p2}: shape2 parameter
- \textit{lower}: lower bound
- \textit{upper}: upper bound
- \textit{lg}: logical; if TRUE, return log density.

\begin{verbatim}
dcauchy_l \hspace{1cm} \textit{A modified dcauchy functions}
\end{verbatim}

\textbf{Description}

A modified dcauchy functions

\textbf{Usage}

dcauchy_l(x, p1, p2, lg = FALSE)

\textbf{Arguments}

- \textit{x}: quantile
- \textit{p1}: location parameter
- \textit{p2}: scale parameter
- \textit{lg}: log density?
### dconstant

A pseudo constant function to get constant densities

**Description**

Used with constant prior

**Usage**

\[
d\text{constant}(x, p1, p2, \text{lower}, \text{upper}, \text{lg} = \text{false})
\]

**Arguments**

- **x**: quantile
- **p1**: constant value
- **p2**: unused argument
- **lower**: dummy variable
- **upper**: dummy variable
- **lg**: log density?

### deviance.model

Calculate the statistics of model complexity

**Description**

Calculate deviance for a model object for which a log-likelihood value can be obtained, according to the formula \(-2\log\text{-likelihood}\).

**Usage**

```r
## S3 method for class 'model'
deviance(object, ...)
```

**Arguments**

- **object**: posterior samples
- **...**: other plotting arguments passing through dot dot dot.
**dgamma_l**  
*A modified dgamma function*

---

**Description**  
A modified dgamma function

**Usage**  
dgamma_l(x, p1, p2, lower, upper, lg = FALSE)

**Arguments**  
- `x`: quantile  
- `p1`: shape parameter  
- `p2`: scale parameter  
- `lower`: lower bound  
- `upper`: upper bound  
- `lg`: log density?

---

**DIC**  
*Deviance information criteria*

---

**Description**  
Calculate DIC and BPIC.

**Usage**  
DIC(object, ...)  
BPIC(object, ...)

**Arguments**  
- `object`: posterior samples  
- `...`: other plotting arguments passing through dot dot dot.
**dlnorm_l**

A modified dlnorm functions

**Description**

A modified dlnorm functions

**Usage**

dlnorm_l(x, p1, p2, lower, upper, lg = FALSE)

**Arguments**

- **x**: quantile
- **p1**: meanlog parameter
- **p2**: sdlog parameter
- **lower**: lower bound
- **upper**: upper bound
- **lg**: log density?

---

**dtnorm**

Truncated Normal Distribution

**Description**

Random number generation, probability density and cumulative density functions for truncated normal distribution.

**Usage**

dtnorm(x, p1, p2, lower, upper, lg = FALSE)

rtnorm(n, p1, p2, lower, upper)

ptnorm(q, p1, p2, lower, upper, lt = TRUE, lg = FALSE)

**Arguments**

- **x, q**: vector of quantiles;
- **p1**: mean (must be scalar).
- **p2**: standard deviation (must be scalar).
- **lower**: lower truncation value (must be scalar).
- **upper**: upper truncation value (must be scalar).
- **lg**: log probability. If TRUE (default is FALSE) probabilities p are given as log(p).
- **n**: number of observations. n must be a scalar.
- **lt**: lower tail. If TRUE (default) probabilities are P[X <= x], otherwise, P[X > x].
Value

a column vector.

Examples

```r
## rtn example
dat1 <- rtnorm(1e5, 0, 1, 0, Inf)
hist(dat1, breaks = "fd", freq = FALSE, xlab = "", main = "Truncated normal distributions")

## dtn example
x <- seq(-5, 5, length.out = 1e3)
dat1 <- dt(x, 0, 1, -2, 2, 0)
plot(x, dat1, type = "l", lwd = 2, xlab = "", ylab = "Density", main = "Truncated normal distributions")

## ptn example
x <- seq(-10, 10, length.out = 1e2)
mean <- 0
sd <- 1
lower <- 0
upper <- 5
dat1 <- ptnorm(x, 0, 1, 0, 5, lg = TRUE)
```

effectiveSize_hyper Calculate effective sample sizes

description
effectiveSize calls effectiveSize in `coda` package to calculate sample sizes.

Usage
effectiveSize_hyper(x, start, end, digits, verbose)
effectiveSize_many(x, start, end, verbose)
effectiveSize_one(x, start, end, digits, verbose)
effectiveSize(x, hyper = FALSE, start = 1, end = NA, digits = 0, verbose = FALSE)

Arguments

- `x`: posterior samples
- `start`: starting iteration
- `end`: ending iteration
Potential scale reduction factor

gelman function calls the function, gelman.diag in the coda package to calculates PSRF.

Usage

gelman(x, hyper = FALSE, start = 1, end = NA, confidence = 0.95, transform = TRUE, autoburnin = FALSE, multivariate = TRUE, split = TRUE, subchain = FALSE, nsubchain = 3, digits = 2, verbose = FALSE, ...)

hgelman(x, start = 1, end = NA, confidence = 0.95, transform = TRUE, autoburnin = FALSE, split = TRUE, subchain = FALSE, nsubchain = 3, digits = 2, verbose = FALSE, ...)

Arguments

x posterior samples

hyper a Boolean switch, indicating posterior samples are from hierarchical modeling

start start iteration
end  end iteration
confidence  confident interval
transform  turn on transform
autoburnin  turn on auto burnin
multivariate  multivariate Boolean switch
split  split whether split mcmc chains; When split is TRUE, the function doubles the number of chains by splitting into 1st and 2nd halves.
subchain  whether only calculate a subset of chains
nsubchain  indicate how many chains in a subset
digits  print out how many digits
verbose  print more information
...  arguments passing to coda gelman.diag.

Examples

```r
## Not run:
rhat1 <- hgelman(hsam); rhat1
rhat2 <- hgelman(hsam, end = 51); rhat2
rhat3 <- hgelman(hsam, confidence = .90); rhat3
rhat4 <- hgelman(hsam, transform = FALSE); rhat4
rhat5 <- hgelman(hsam, autoburnin = TRUE); rhat5
rhat6 <- hgelman(hsam, split = FALSE); rhat6
rhat7 <- hgelman(hsam, subchain = TRUE); rhat7
rhat8 <- hgelman(hsam, subchain = TRUE, nsubchain = 4);
rhat9 <- hgelman(hsam, subchain = TRUE, nsubchain = 4,
digits = 1, verbose = TRUE);

hat1 <- gelman(hsam[[1]], multivariate = FALSE); hat1
hat2 <- gelman(hsam[[1]], hyper = TRUE, verbose = TRUE); hat2
hat3 <- gelman(hsam, hyper = TRUE, verbose = TRUE); hat3
hat4 <- gelman(hsam, multivariate = TRUE, verbose = FALSE);
hat5 <- gelman(hsam, multivariate = FALSE, verbose = FALSE);
hat6 <- gelman(hsam, multivariate = FALSE, verbose = TRUE);
hat7 <- gelman(hsam, multivariate = T, verbose = TRUE);

## End(Not run)
```

GetNsim  Get a n-cell matrix

Description

Constructs a matrix, showing how many responses to in each cell. The function checks whether the format of n and ns conform.
GetNsim

Usage

GetNsim(model, n, ns)

Arguments

model a model object.

n number of trials.

ns number of subjects.

Details

n can be:

1. an integer for a balanced design,

2. a matrix for an unbalanced design, where rows are subjects and columns are cells. If the matrix is a row vector, all subjects have the same n in each cell. If it is a column vector, all cells have the same n. Otherwise each entry specifies the n for a particular subject x cell combination. See below for concrete examples.

Examples

```r
model <- BuildModel(
  p.map = list(A = "1", B = "R", t0 = "1", mean_v = "M", sd_v = "M",
                 st0 = "1"),
  match.map = list(M = list(s1 = 1, s2 = 2)),
  constants = c(sd_v = FALSE, st0 = 0),
  factors = list(s = c("s1","s2")),
  responses = c("r1", "r2"),
  type = "norm")
```

# Example 1

GetNsim(model, ns = 2, n = 1)

# [1,] 1 1
# [2,] 1 1

# Example 2

n <- matrix(c(1:2), ncol = 1)

GetNsim(model, ns = 2, n = n)

# [1,] 1 1
# [2,] 2 2
### Example 3
n <- matrix(c(1:2), nrow = 1)
# [,1] [,2]
# [1,] 1 2
GetNsim(model, ns = 2, n = n)
# [,1] [,2]
# [1,] 1 2 ## subject 1 has 1 response for cell 1 and 2 responses for cell 2
# [2,] 1 2 ## subject 2 has 1 response for cell 1 and 2 responses for cell 2

### Example 4
n <- matrix(c(1:4), nrow=2)
# [,1] [,2]
# [1,] 1 3
# [2,] 2 4
ggdm::GetNsim(model, ns = 2, n = n)
# [,1] [,2]
# [1,] 1 3 ## subject 1 has 1 response for cell 1 and 3 responses for cell 2
# [2,] 2 4 ## subject 2 has 2 responses for cell 1 and 4 responses for cell 2

---

**GetParameterMatrix**  
*Constructs a ns x npar matrix.*

---

**Description**

The matrix is used to simulate data. Each row represents one set of parameters for a participant.

**Usage**

`GetParameterMatrix(x, nsub, prior = NA, ps = NA, seed = NULL)`

**Arguments**

- `x`  
a model object
- `nsub`  
number of subjects.
- `prior`  
a prior object
- `ps`  
a vector or a matrix.
- `seed`  
an integer specifying a random seed.

**Details**

One must enter either a vector or a matrix as true parameters to the argument, `ps`, when presuming to simulate data based on a fixed-effect model. When the assumption is to simulate data based on a random-effect model, one must enter a prior object to the argument, `prior` to first randomly generate a true parameter matrix.
**GetPNames**

**Value**

A `ns x npar` matrix

**Examples**

```r
model <- BuildModel(
p.map = list(a = "1", v = "1", z = "1", d = "1", sz = "1", sv = "1", 
t0 = "1", st0 = "1"),
match.map = list(M = list(s1 = "r1", s2 = "r2")),
factors = list(s = c("s1", "s2")),
constants = c(st0 = 0, d = 0),
responses = c("r1", "r2"),
type = "rd")
p.prior <- BuildPrior(
dists = c("tnorm", "tnorm", "beta", "beta", "tnorm", "beta"),
p1 = c(a = 1, v = 0, z = 1, sz = 1, sv = 1, t0 = 1),
p2 = c(a = 1, v = 2, z = 1, sz = 1, sv = 1, t0 = 1),
lower = c(0, -5, NA, NA, 0, NA),
upper = c(2, 5, NA, NA, 2, NA))

## Example 1: Randomly generate 2 sets of true parameters from
## parameter priors (p.prior)
GetParameterMatrix(model, 2, p.prior)
## a v z sz sv t0
## [1,] 1.963067 1.472940 0.9509158 0.5145047 1.344705 0.0850591
## [2,] 1.512276 -1.995631 0.6981290 0.2626882 1.867853 0.1552828

## Example 2: Use a user-selected true parameters
true.vector <- c(a=1, v=1, z=0.5, sz=0.2, sv=1, t0=.15)
GetParameterMatrix(model, 2, NA, true.vector)
## a v z sz sv t0
## [1,] 1 1 0.5 0.2 1 0.15
## [2,] 1 1 0.5 0.2 1 0.15
GetParameterMatrix(model, 2, ps = true.vector)

## Example 3: When a user enter arbitrary sequence of parameters.
## Note sv is before sz. It should be sz before sv
## See correct sequence, by entering "attr(model, 'p.vector')"
## GetParameterMatrix will rearrange the sequence.
true.vector <- c(a=1, v=1, z=0.5, sv=1, sz = .2, t0= .15)
GetParameterMatrix(model, 2, NA, true.vector)
## a v z sz sv t0
## [1,] 1 1 0.5 0.2 1 0.15
## [2,] 1 1 0.5 0.2 1 0.15
```

---

**GetPNames**

Extract parameter names from a model object
Description

Extract parameter names from a model object

Usage

getPNames(x)

Arguments

x a model object

get_os

Retrieve information of operating system

Description

A wrapper function to extract system information from Sys.info and Platform

Usage

get_os()

Examples

get_os()

Example:

## sysname

## "linux"

ggdmc

Bayesian computation of response time models

Description

ggdmc uses the population-based Markov chain Monte Carlo to conduct Bayesian computation on cognitive models.

Author(s)

Yi-Shin Lin <yishinlin001@gmail.com>

Andrew Heathcote <andrew.heathcote@utas.edu.au>
References


### iseffective

*Model checking functions*

**Description**
The function tests whether we have drawn enough samples.

**Usage**

iseffective(x, minN, nfun, verbose = FALSE)

**Arguments**

- **x**: posterior samples
- **minN**: specify the size of minimal effective samples
- **nfun**: specify to use the mean or median function to calculate effective samples
- **verbose**: print more information

### isflat

*Model checking functions*

**Description**
The function tests whether Markov chains converge prematurely:

**Usage**

isflat(x, p1 = 1/3, p2 = 1/3, cut_location = 0.25, cut_scale = Inf, verbose = FALSE)
### ismixed

**Description**

The function tests whether Markov chains are mixed well.

**Usage**

```r
ismixed(x, cut = 1.01, split = TRUE, verbose = FALSE)
```

**Arguments**

- `x`: posterior samples
- `cut`: PSRF criterion for well mixed
- `split`: whether to split MCMC chains. This is an argument passing to `gelman` function
- `verbose`: print more information

*See Also*

`gelman`

### issstuck

**Description**

The function tests whether Markov chains encounter a parameter region that is difficult to search. `CheckConverged` is a wrapper function running the four checking functions, `issstuck`, `isflat`, `ismixed` and `iseffective`. 
**Usage**

```r
isstuck(x, hyper = FALSE, cut = 10, start = 1, end = NA, verbose = FALSE)

CheckConverged(x)
```

**Arguments**

- `x` : posterior samples
- `hyper` : a Boolean switch, extracting hyper attribute.
- `cut` : the criteria for suggesting abnormal chains found
- `start` : start iteration
- `end` : end iteration
- `verbose` : print more information

**Description**


**Usage**

```r
likelihood(pvector, data, min_lik = 1e-10)
```

**Arguments**

- `pvector` : a parameter vector
- `data` : data model instance
- `min_lik` : minimal likelihood.

**Value**

a vector

**References**


Examples

```r
model <- BuildModel(
  p.map = list(A = "1", B = "1", t0 = "1", mean_v = "M", sd_v = "1",
              st0 = "1"),
  match.map = list(M = list(s1 = 1, s2 = 2)),
  factors = list(S = c("s1", "s2")),
  constants = c(st0 = 0, sd_v = 1),
  responses = c("r1", "r2"),
  type = "norm")

p.vector <- c(A = .25, B = .35, t0 = .2, mean_v.true = 1, mean_v.false = .25)
dat <- simulate(model, le3, ps = p.vector)
dmi <- BuildDMI(dat, model)
den <- likelihood(p.vector, dmi)

model <- BuildModel(
  p.map = list(a = "1", v = "1", z = "1", d = "1", t0 = "1", sv = "1",
              sz = "1", st0 = "1"),
  constants = c(st0 = 0, d = 0),
  match.map = list(M = list(s1 = "r1", s2 = "r2")),
  factors = list(S = c("s1", "s2")),
  responses = c("r1", "r2"),
  type = "rd")

p.vector <- c(a = 1, v = 1, z = 0.5, sz = 0.25, sv = 0.2, t0 = .15)
dat <- simulate(model, le2, ps = p.vector)
dmi <- BuildDMI(dat, model)
den <- likelihood(p.vector, dmi)
```

mcmc_list.model

Create a MCMC list

Description

Create a MCMC list

Usage

```r
mcmc_list.model(x, start = 1, end = NA, pll = TRUE)
```

Arguments

- `x` : posterior samples
- `start` : start from which iteration
- `end` : end at which iteration
- `pll` : a Boolean switch for calculating posterior log-likelihood
Description

Calculate each chain separately for the mean (across many MCMC iterations) of posterior log-likelihood. If the difference of the means and the median (across chains) of the mean of posterior is greater than the cut, chains are considered stuck. The default value for cut is 10. 

Usage

```
PickStuck(x, hyper = FALSE, cut = 10, start = 1, end = NA,
         verbose = FALSE, digits = 2)
```

Arguments

- **x**: posterior samples
- **hyper**: whether x are hierarchical samples
- **cut**: a criterion deciding if a chain is stuck.
- **start**: start to evaluate from which iteration.
- **end**: end at which iteration for evaluation.
- **verbose**: a boolean switch to print more information
- **digits**: print how many digits. Default is 2

Value

pickStuck gives an index vector; unstuck gives a DMC sample.

Examples

```r
model <- buildModel(
  p.map = list(A = "1", B = "1", t0 = "1", mean_v = "M", sd_v = "1", st0 = "1"),
  match.map = list(M = list(s1 = 1, s2 = 2)),
  factors = list(S = c("s1", "s2")),
  constants = c(st0 = 0, sd_v = 1),
  responses = c("r1", "r2"),
  type = "norm")

p.vector <- c(A = .75, B = .25, t0 = .2, mean_v.true = 2.5, mean_v.false = 1.5)

p.prior <- buildPrior(
  dists = c("tnorm", "tnorm", "beta", "tnorm", "tnorm"),
  p1 = c(A = .3, B = .3, t0 = 1, mean_v.true = 1, mean_v.false = 0),
  p2 = c(1, 1, 1, 3, 3),
  lower = c(0, 0, 0, NA, NA),
  upper = c(NA, NA, 1, NA, NA))
```
## Not run:
```r
dat <- simulate(model, 30, ps = p.vector)
dmi <- BuildDMI(dat, model)
sam <- run(StartNewsamples(dmi, pNprior))
bad <- PickStuck(sam)

## End(Not run)
```

---

### plot_prior

#### Plot prior distributions

**Description**

`plot_prior` plots one member in a prior object. `plot.prior` plots all members in a prior object.

**Usage**

```r
plot_prior(i, prior, xlim = NA, natural = TRUE, npoint = 100,
trans = NA, save = FALSE, ...)
```

**Arguments**

- `i` an integer or a character string indicating which parameter to plot
- `prior` a prior object
- `xlim` set the range of on x axis. This is usually the range for each parameter.
- `natural` default TRUE.
- `npoint` default to plot 100
- `trans` default NA. `trans` can be a scalar or vector.
- `save` whether to save the data out
- `...` other plotting arguments passing through dot dot dot.
- `x` a prior object
- `ps` true parameter vectors or matrix in the case of many observation units

**Examples**

```r
p prior <- BuildPrior(
dists = rep("tnorm", 7),
p1 = c(a = 2, v.f1 = 4, v.f2 = 3, z = 0.5, sv = 1,
sz = 0.3, t0 = 0.3),
p2 = c(a = 0.5, v.f1 = .5, v.f2 = .5, z = 0.1, sv = .3,
sz = 0.1, t0 = 0.05),
lower = c(0, -5, -5, 0, 0, 0, 0),
upper = c(5, 7, 7, 1, 2, 1, 1))
```
print.prior

plot_prior("a", p.prior)
plot_prior(2, p.prior)
plot(p.prior)

print.prior  Print Prior Distribution

Description

a convenient function to rearrange p.prior or an element in a pp.prior as a data frame for inspection.

Usage

## S3 method for class 'prior'
print(x, ...)

Arguments

x a list of prior distributions list, usually created by BuildPrior

... other arguments

Value

a data frame listing prior distributions and their settings

Examples

pop.mean <- c(a=1, v.f1=1, v.f2=.2, z=.5, sz=.3, sv.f1=.25, sv.f2=.23, t0=.3)
pop.scale <- c(a=.2, v.f1=.2, v.f2=.2, z=.1, sz=.05, sv.f1=.05, sv.f2=.05, t0=.05)
p.prior <- BuildPrior(
dists = rep("tnorm", 8),
p1 = pop.mean,
p2 = pop.scale,
lower = c(0, -5, -5, 0, 0, 0, 0, 0),
upper = c(2, 5, 5, 1, 2, 2, 1, 1))

print(p.prior)
**random**

*Generate random numbers*

**Description**

A wrapper function for generating random numbers of either the model type, rd, or norm.

**Usage**

```r
random(typeL, pmatL, nL, seed = NULL)
```

**Arguments**

- **type**: a character string of the model type
- **pmat**: a matrix of response x parameter
- **n**: number of observations
- **seed**: an integer specifying a random seed

**rlba_norm**

*Generate Random Deviates of the LBA Distribution*

**Description**

rlba_norm, only slightly faster than maker, calls C++ function directly.

**Usage**

```r
rlba_norm(nL, A, bL, mean_vL, sd_vL, t0L, st0L, posdrift)
```

**Arguments**

- **n**: is the numbers of observation.
- **A**: start point upper bound, a vector of a scalar.
- **b**: decision threshold, a vector or a scalar.
- **mean_v**: mean drift rate vector
- **sd_v**: standard deviation of drift rate vector
- **t0**: nondecision time, a vector.
- **st0**: nondecision time variation, a vector.
- **posdrift**: if exclude negative drift rates

**Value**

a n x 2 matrix of RTs (first column) and responses (second column).
**rprior**  

*Parameter Prior Distributions*

**Description**

Probability density functions and random generation for parameter prior distributions.

**Usage**

```
rprior(prior, n = 1)
```

**Arguments**

- `prior` a list of list usually created by BuildPrior to store the information about parameter prior distributions.
- `n` number of observations/random draws

**Examples**

```r
# pvec <- c(a=1, v=1, z=0.5, sz=0.25, sv=0.2, t0=.15)
p.prior <- BuildPrior(
  dists = rep("tnorm", 6),
  p1 = c(a=2, v=2.5, z=0.5, sz=0.3, sv=1, t0=0.3),
  p2 = c(a=0.5, v=5, z=0.1, sz=0.1, sv=.3, t0=0.05) * 5,
  lower = c(0,-5, 0, 0, 0, 0),
  upper = c(5, 7, 2, 2, 2, 2))
```

```r
rprior(p.prior, 9)
```

```r
t # a v z sz sv t0
# [1,] 0.97413686 0.78446178 0.9975199 -0.5264946 0.5364492 0.55415052
# [2,] 0.72870190 0.97151662 0.8516604 1.6008591 0.3399731 0.96528848
# [3,] 1.63153685 1.96586939 0.9260939 0.7041254 0.4138329 0.78367440
# [4,] 1.55866180 1.43657110 0.6152371 0.1290078 0.2957604 0.23027759
# [5,] 1.32528281 -0.07328408 0.2051155 2.4046387 0.9663111 0.06127237
# [6,] 0.49628528 -0.19374778 0.5142829 2.1452972 0.4335482 0.38410626
# [7,] 0.83655549 0.77223432 0.1739831 1.4431507 0.6257398 0.63228368
# [8,] 0.71197612 -1.15798882 0.8265523 0.3813370 0.4465184 0.23955415
# [9,] 0.38049166 3.32132034 0.9888108 0.9684292 0.8437480 0.13502154
```

**simulate.model**  

*Simulate response time data*

**Description**

Simulate response time data either for one subject or multiple subjects. The simulation is based on a model object. For one subject, one must supply a true parameter vector to the `ps` argument.

**Usage**

```r
## S3 method for class 'model'
simulate(object, nsim = NA, seed = NULL, nsub = NA,
         prior = NA, ps = NA, ...)  
```

**Arguments**

- **object**: a model object.
- **nsim**: number of trials / responses. `n` can be a single number for a balanced design or a matrix for an unbalanced design, where rows are subjects and columns are design cells. If the matrix has one row then all subjects have the same `n` in each cell; if it has one column then all cells have the same `n`; Otherwise each entry specifies the `n` for a particular subject x design cell combination.
- **seed**: a user specified random seed.
- **nsub**: number of subjects
- **prior**: a prior object
- **ps**: a true parameter vector or matrix.
- **...**: additional optional arguments.

**Details**

For multiple subjects, one can enter a matrix (or a row vector) as true parameters. Each row is to generate data separately for a subject. This is the fixed-effect model. To generate data based on a random-effect model, one must supply a prior object. In this case, `ps` argument is unused. Note in some cases, a random-effect model may fail to draw data from the model, because true parameters are randomly drawn from a prior object. This would happen sometimes in diffusion model, because certain parameter combinations are considered invalid.

`ps` can be a row vector, in which case each subject has identical parameters. It can also be a matrix with one row per subject, in which case it must have `ns` rows. The true values will be saved as `parameters` attribute in the output object.

**Value**

a data frame
Description

Fit a hierarchical or a fixed-effect model, using Bayesian optimisation. We use a specific type of pMCMC algorithm, the DE-MCMC. This particular sampling method includes crossover and two different migration operators. The migration operators are similar to random-walk algorithm. They would be less efficient to find the target parameter space, if been used alone.

Usage

StartNewsamples(data, prior = NULL, nmc = 200, thin = 1,
             nchain = NULL, report = 100, rp = 0.001, gammamult = 2.38,
             pm0 = 0.05, pm1 = 0.05, block = TRUE, ncore = 1)

run(samples, nmc = 500, thin = 1, report = 100, rp = 0.001,
      gammamult = 2.38, pm0 = 0, pm1 = 0, block = TRUE, ncore = 1,
      add = FALSE)

Arguments

data          data model instance(s)
prior          prior objects. For hierarchical model, this must be a list with three sets of prior distributions. Each is respectively named, "pprior", "location", and "scale".
nmc            number of Monte Carlo samples
thin           thinning length
nchain         number of chains
report         progress report interval
rp             tuning parameter 1
gammamult     tuning parameter 2. This is the step size.
pm0            probability of migration type 0 (Hu & Tsui, 2010)
pm1            probability of migration type 1 (Turner et al., 2013)
block         Only for hierarchical modeling. A Boolean switch for update one parameter at a time
ncore         Only for non-hierarchical, fixed-effect models with many subjects.
samples       posterior samples.
add            Boolean whether to add new samples
**summary.model**

**Summarise posterior samples**

**Description**

This calls seven different variants of summary function to summarise posterior samples.

**Usage**

```r
## S3 method for class 'model'
summary(object, hyper = FALSE, start = 1, end = NA,
    hmeans = FALSE, hci = FALSE, prob = c(0.025, 0.25, 0.5, 0.75,
    0.975), recovery = FALSE, ps = NA, type = 1, verbose = FALSE,
    digits = 2, ...)
```

**Arguments**

- `object`: posterior samples
- `hyper`: whether to summarise hyper parameters
- `start`: start from which iteration.
- `end`: end at which iteration. For example, set `start = 101` and `end = 1000`, instructs the function to calculate from 101 to 1000 iteration.
- `hmeans`: a boolean switch indicating to calculate mean of hyper parameters
- `hci`: boolean switch; whether to calculate credible intervals of hyper parameters
- `prob`: a numeric vector, indicating the quantiles to calculate
- `recovery`: a boolean switch indicating if samples are from a recovery study
- `ps`: true parameter values. This is only for recovery studies
- `type`: calculate type 1 or 2 hyper parameters
- `verbose`: print more information
- `digits`: printing digits
- `...`: other arguments

**Examples**

```r
## Not run:
est1 <- summary(hsam[[1]], FALSE)
est2 <- summary(hsam[[1]], FALSE, 1, 100)
est3 <- summary(hsam)
est4 <- summary(hsam, verbose = TRUE)
est5 <- summary(hsam, verbose = FALSE)
hest1 <- summary(hsam, TRUE)
## End(Not run)
```
**summary_mcmc_list**  
*Summary statistic for posterior samples*

**Description**  
Calculate summary statistics for posterior samples

**Usage**  
`summary_mcmc_list(object, prob = c(0.025, 0.25, 0.5, 0.75, 0.975), ...)`

**Arguments**
- `object`  
  - posterior samples
- `prob`  
  - summary quantile summary
- `...`  
  - other arguments passing in

**TableParameters**  
*Table response and parameter*

**Description**  
TableParameters arranges the values in a parameter vector and creates a response x parameter matrix. The matrix is used by the likelihood function, assigning a trial to a cell for calculating probability densities.

**Usage**  
TableParameters(p.vector, cell, model, n1order)

**Arguments**
- `p.vector`  
  - a parameter vector
- `cell`  
  - a string or an integer indicating a design cell, e.g., `s1.f1.r1` or 1. Note the integer cannot exceed the number of cell. One can check this by entering `length(dimnames(model))`.
- `model`  
  - a model object
- `n1order`  
  - a Boolean switch, indicating using node 1 ordering. This is only for LBA-like models and its n1PDF likelihood function.

**Value**  
Each row corresponding to the model parameter for a response. When `n1.order` is FALSE, TableParameters returns a matrix without rearranging into node 1 order. For example, this is used in the `simulate` function. By default `n1.order` is TRUE.
Examples

```r
m1 <- BuildModel(
  p.map = list(a = "1", v = "F", z = "1", d = "1", sz = "1", sv = "F",
              t0 = "1", st0 = "1"),
  match.map = list(M = list(s1 = "r1", s2 = "r2")),
  factors = list(S = c("s1", "s2"), F = c("f1", "f2")),
  constants = c(st0 = 0, d = 0),
  responses = c("r1", "r2"),
  type = "rd")

m2 <- BuildModel(
  p.map = list(A = "1", B = "1", mean_v = "M", sd_v = "1",
              t0 = "1", st0 = "1"),
  constants = c(st0 = 0, sd_v = 1),
  match.map = list(M = list(s1 = 1, s2 = 2)),
  factors = list(S = c("s1", "s2")),
  responses = c("r1", "r2"),
  type = "norm")

pvec1 <- c(a = 1.15, v.f1 = -0.10, v.f2 = 3, z = 0.74, sz = 1.23,
           sv.f1 = 0.11, sv.f2 = 0.21, t0 = 0.87)
pvec2 <- c(A = .75, B = .25, mean_v.true = 2.5, mean_v.false = 1.5,
           t0 = .2)

print(m1, pvec1)
print(m2, pvec2)

accMat1 <- TableParameters(pvec1, "s1.r1", m1, FALSE)
accMat2 <- TableParameters(pvec2, "s1.r1", m2, FALSE)
```

```
# a v t0 z d sz sv st0
# 1.15 -0.1 0.87 0.26 0.123 0.11 0
# 1.15 -0.1 0.87 0.26 0.123 0.11 0

# A b t0 mean_v sd_v st0
# 0.75 1.0 2.5 1 0
# 0.75 1.0 1.5 1 0
```

theta2mcmclist  

Convert theta to a mcmc List

Description

Extracts the parameter array (ie theta) from posterior samples of a participant and convert it to a coda mcmc.list.

Usage

`theta2mcmclist(x, start = 1, end = NA, split = FALSE,
subchain = FALSE, nsubchain = 3, thin = NA)`
phi2mcmclist(x, start = 1, end = NA, split = FALSE, subchain = FALSE, nsubchain = 3)

Arguments

- **x**: posterior samples
- **start**: start iteration
- **end**: end iteration
- **split**: whether to divide one MCMC sequence into two sequences.
- **subchain**: boolean switch to convert only a subset of chains
- **nsubchain**: indicate the number of chains in the subset
- **thin**: thinning length of the posterior samples

Details

phi2mcmclist extracts the phi parameter array, which stores the location and scale parameters at the hyper level.

Examples

```r
## Not run:
model <- buildModel(
  p.map = list(a = "RACE", v = c("S", "RACE"), z = "RACE", d = "1",
    sz = "1", sv = "1", t0 = c("S", "RACE"), st0 = "1"),
  match.map = list(M = list(gun = "shoot", non = "not")),
  factors = list(S = c("gun", "non"), RACE = c("black", "white"),
    constants = c(st0 = 0, d = 0, sz = 0, sv = 0),
    responses = c("shoot", "not"),
    type = "rd")

pnames <- getPNames(model)
npar <- length(pnames)

pop.mean <- c(1, 1, 2.5, 2.5, 2.5, 2.5, 2.5, .50, .50, .4, .4, .4, .4)

pop.scale <- c(1.15, .15, 1, 1, 1, 1, .05, .05, .05, .05, .05, .05)

names(pop.mean) <- pnames

names(pop.scale) <- pnames

pop.prior <- buildPrior(
  dists = rep("tnorm", npar),
  p1 = pop.mean,
  p2 = pop.scale,
  lower = c(rep(0, 2), rep(-5, 4), rep(0, 6)),
  upper = c(rep(5, 2), rep(7, 4), rep(2, 6)))

p.prior <- buildPrior(
  dists = rep("tnorm", npar),
  p1 = pop.mean,
  p2 = pop.scale*10,
  lower = c(rep(0, 2), rep(-5, 4), rep(0, 6)),
  upper = c(rep(10, 2), rep(NA, 4), rep(5, 6)))

mu.prior <- buildPrior(
```
unstick_one

Unstick posterios samples (One subject)

Description
Unstick posterios samples (One subject)

Usage
unstick_one(x, bad)

Arguments
x
  posterior samples
bad
  a numeric vector, indicating which chains to remove
Index

*Topic package
  ggdmc, 16

BPIC (DIC), 8
BuildDMI, 2
BuildModel, 3
BuildPrior, 4

check_pvec, 5
CheckConverged (isstuck), 18
ConvertChains, 5

dbeta_lu, 6
dcauchy_l, 6
dconstant, 7
deviance.model, 7
dgamma_l, 8
DIC, 8
dlnormal, 9
dtnorm, 9
effectiveSize (effectiveSize_hyper), 10
effectiveSize_hyper, 10
effectiveSize_many
  (effectiveSize_hyper), 10
effectiveSize_one
  (effectiveSize_hyper), 10
gelman, 11, /8
get_os, 16
GetNsim, 12
GetParameterMatrix, 14
GetPNames, 15

ggdmc, 16
{ggdmc-package (ggdmc)}, 16

hgelman (gelman), 11

iseffective, 17
isflat, 17
ismixed, 18

isstuck, 18
likelihood, 19
mcmc_list.model, 20

pbeta, 4
gamma, 4
phi2mcmcclist (theta2mcmcclist), 30
PickStuck, 21
plnorm, 4
plot.prior (plot_prior), 22
plot_prior, 22
print.dmi (BuildModel), 3
print.model (BuildModel), 3
print.prior, 23
ptnorm (dtgamma), 9

random, 24
rlba_norm, 24
rprior, 25
rtnorm (dtgamma), 9
run (StartNewsamples), 27

simulate.model, 26
StartNewsamples, 27
summary.model, 28
summary_mcmc_list, 29

TableParameters, 29
theta2mcmcclist, 30
unstick_one, 32