Package 'ggdmc'

September 2, 2018

Type Package

Title Cognitive Models

Version 0.2.5.2

Date 2018-08-27

Author Yi-Shin Lin [aut, cre], Andrew Heathcote [aut]

Maintainer Yi-Shin Lin <yishin.lin@utas.edu.au>

Description Hierarchical Bayesian modelling. The 'ggdmc' package provides tools designed for fitting cognitive models, using population-based Markov Chain Monte Carlo (pMCMC) and fast C++ libraries. The paper by Heathcote, Lin, Reynolds, Strickland, Gretton, and Matzke (2018, <doi:10.3758/s13428-018-1067-y>) describes the early prototype of the package (by the version 0.4.2). The paper accompanied with the revamped latest version (> 0.2.4.5) is under preparation Lin, Strickland, Reynold, and Heathcote (2018) and a further paper regarding the new pMCMC sampling modified based on Hu and Tsai's (2010) work is also under preparation. See 'citation(``ggdmc'')' for details.

License GPL-2

URL https://github.com/yxlin/ggdmc

BugReports https://github.com/yxlin/ggdmc/issues

LazyData TRUE

Imports Rcpp (>= 0.12.10), coda, rtdists, ggmcmc (>= 0.7.3), stats, utils, data.table (>= 1.10.4), tmvtnorm, ggplot2, matrixStats

Depends R (>= 3.4.0)

LinkingTo Rcpp (>= 0.12.10), RcppArmadillo (>= 0.7.100.3.0), BH

RoxygenNote 6.1.0

Encoding UTF-8

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-09-01 23:20:02 UTC

R topics documented:

---

ac *Calculate autocorrelation*

Description

This function calculate autocorrealtion for a Markov Chain

Usage

```
ac(x, nlag)
```

Arguments

x	likelihood vector
nlag	number of autocorrelation lags

assign_pp *Slot pp values into p.prior*

Description

Assign hyper-prior distributions to prior distributions

Usage

```
assign_pp(pp, prior)
```

Arguments

pp	pp.prior
prior	p.prior

autocor *Autocorrelation Plot*

Description

Plot the autocorrelation of posterior samples,

Usage

```
autocor(x, start = 1, end = NA)
```

Arguments

x	posterior samples
start	start from which iteration.
end	end at which iteration

BuildDMI *Bind data and models*

Description

Binding a data with a model object. This function also checks if they are compatible and adds a `cell.index` and many other attributes to the data frame.

Usage

```
BuildDMI(x, model, npda = 16384, bw = 0.01, gpuid = 0,
  debug = FALSE)
```

Arguments

`x`	data in a data frame format
`model`	a model object
`npda`	number of model simulations
`bw`	kernel bandwidth
`gpuid`	GPU ID, indicating using which GPU cards on a machine with multieple GPUs.
`debug`	debugging switch

BuildModel *Create a model object*

Description

Create a model array and attach many model attributes. These attributes specify a particular model and parameterisation.

Usage

```
BuildModel(p.map, responses, factors = list(dummy = "1"),
  match.map = NULL, constants = numeric(0), type = "norm",
  posdrift = TRUE, verbose = TRUE, cvs = NULL, responses2 = NULL,
  constant_prior = NULL)

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

Arguments

p.map	mapping factorial design to model parameters
responses	Response (accumulator) names
factors	specifying factors and factor levels
match.map	matching stimuli and responses
constants	setting parameters as constant value
type	using character string to specifying model type.
posdrift	enforce postive drift rate, using truncated normal or just using normal distribution. This is used only by norm type (any LBA variants and extensions)
verbose	Print parameter vector, constants and model type
cvs	Names of trial covariates (in data). A redundant argument.
responses2	Second response name (multi-threshold models)
constant_prior	Parameter sampled from a fixed prior
x	a model object
p.vector	parameter vector (for printing model)
...	other arguments

Examples

```
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. p1 and p2 refer to the first and second parameters a prior distribution.

Usage

```
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", "cauchy", "constant"))
```

Arguments

p1	the first parameter of a distribution
p2	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: p1 = mean, p2 = sd. It specifies a normal distribution when bounds are set -Inf and Inf,
2. Beta, where: p1 = shape1 and p2 = shape2 (see pbeta). Note the uniform distribution is a special case of the beta with p1 and p2 = 1),
3. Gamma, where p1 = shape and p2 = scale (see pgamma). Note p2 is scale, not rate,
4. Lognormal, where p1 = meanlog and p2 = sdlog (see plnorm).

Value

a list of list

---

cellIdx2Mat *Convert cell index list to a matrix*

Description

Convert cell index list to a matrix. This is a convenient function.

Usage

```
cellIdx2Mat(data)
```

Arguments

data	a model data instance

Value

a matrix

Examples

```
m1 <- 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 = rep("tnorm", 6),
    p1    = c(a=2,  v=2.5, z=.5, sz=.3, sv=1,  t0=.3),
    p2    = c(a=.5, v=.5,  z=.1, sz=.1, sv=.3, t0=.05),
    lower = c(0,-5, 0, 0, 0, 0),
    upper = c(5, 7, 2, 2, 2, 2))
dat <- simulate(m1, nsim = 4, prior = p.prior)
dmi <- BuildDMI(dat, m1)
cellIdx2Mat(dmi)
```

---

censor *Censor missing values and RT outliers*

Description

censor requests a data frame minimally with three columns, indicating stimulus (S factor), response (R factor) and response time (RT).

Usage

```
censor(x, xlim = c(0, Inf))
```

Arguments

x	a data frame
xlim	the lower and upper censoring boundaries

Examples

```
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")
p.vector <- c(a = 1, v = 1, z = 0.5, t0 = .15)
dat <- simulate(model, 100, ps = p.vector)
dmi <- BuildDMI(dat, model)
```

```
## Trim off RTs below .05 and above 10 s
censored_data <- censor(dat, xlim = c(.05, 10))
```

---

checkddm2 *Check parameter vector of DDM*

Description

Check if all DDM parameters are within reasonable ragnes

Usage

```
checkddm2(pVec)
```

Arguments

pVec	a numeric vector storing the DDM parameters

CheckDMI *Check Data Model Instance*

Description

Return a model object extracted either from a data model instance or an object storing posterior samples. The function checks also

1. When x stores DMI of one participant, if DMI is a `data.frame`,
2. When x is an object of posterior samples, if x is a list of many subjects,
3. whether model is successfully created,
4. whether prior is suppled or we can extract it from 'samples'

Usage

```
CheckDMI(x = NULL, prior = NULL, theta1 = NULL, nchain = NULL)
```

Arguments

x	a data-model instance
prior	a parameter prior list
theta1	a user-supplied theta cube
nchain	number of MCMC chains

CheckRJ *Check rejection rate*

Description

Check rejection rate for posterior samples

Usage

```
CheckRJ(object, verbose = TRUE)
```

Arguments

object	posterior samples
verbose	print more information

check_pvec *Check if parameter vector compatible with model object?*

Description

Check if the user supplies a parameter vector, compatiable with the model object.

Usage

```
check_pvec(x, model)
```

Arguments

x	parameter vector
model	a model object

ConvertChains *Prepare posterior samples for plotting version 1*

Description

Convert MCMC chains to a data frame for plotting

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

ConvertChains2 *Prepare posterior samples for plotting version 2*

Description

Convert MCMC chains to a data frame for plotting

Usage

```
ConvertChains2(x, pll)
```

Arguments

x	posterior samples
pll	a Boolean switch to make posterior log likelihood

dbeta_lu *A slightly modified dbeta and rbeta functions*

Description

A slightly modified dbeta and rbeta functions

Usage

```
dbeta_lu(x, shape1, shape2, lower, upper, log = FALSE)

rbeta_lu(n, shape1, shape2, lower, upper)
```

Arguments

x	posterior samples
shape1	shape1 parameter
shape2	shape1 parameter
lower	lower bound
upper	upper bound
log	log density?
n	number of simulations

dcauchy_l *A slightly modified dcauchy and rcauchy functions*

Description

A slightly modified dcauchy and rcauchy functions

Usage

```
dcauchy_l(x, location, scale, log = FALSE)

rcauchy_l(n, location, scale)
```

Arguments

x	posterior samples
location	location parameter
scale	scale parameter
log	log density?
n	number of simulations

dconstant *A pseudo constant function to get constant densities*

Description

Used with constant prior

Usage

```
dconstant(x, constant, log = FALSE)

rconstant(n, constant)
```

Arguments

x	posterior samples
constant	constant value
log	log density?
n	number of simulations

dgamma_l *A slightly modified dgamma and rgamma functions*

Description

A slightly modified dgamma and rgamma functions

Usage

```
dgamma_l(x, shape, scale, lower, upper, log = FALSE)

rgamma_l(n, shape, scale, lower, upper)
```

Arguments

x	posterior samples
shape	shape parameter
scale	scale parameter
lower	lower bound
upper	upper bound
log	log density?
n	number of simulations

dlnorm_l *A slightly modified dlnorm and rlnorm functions*

Description

A slightly modified dlnorm and rlnorm functions

Usage

```
dlnorm_l(x, meanlog, sdlog, lower, upper, log = FALSE)

rlnorm_l(n, meanlog, sdlog, lower, upper)
```

Arguments

x	posterior samples
meanlog	meanlog parameter
sdlog	sdlog parameter
lower	lower bound
upper	upper bound
log	log density?
n	number of simulations

dprior *Probability densities of prior distributions*

Description

sumlogpriorNV calculate sum log-likelihood. rprior generates random observation from prior distributions

Usage

```
dprior(pvec, prior)

sumlogpriorNV(pvec, prior)
```

Arguments

pvec	parameter vector
prior	prior distributions

Value

a vector

dtnorm *Truncated Normal Distribution*

Description

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

Usage

```
dtnorm(x, mean, sd, lower, upper, log = FALSE)

rtnorm(n, mean, sd, lower, upper)

ptnorm(q, mean, sd, lower, upper, lt = TRUE, log = FALSE)
```

Arguments

x, q	vector of quantiles;
mean	mean (must be scalar).
sd	standard deviation (must be scalar).
lower	lower truncation value (must be scalar).
upper	upper truncation value (must be scalar).
log	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

```
## 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 <- dtnorm(x, 0, 1, -2, 2, 0)
plot(x, dat1, type = "l", lwd = 2, xlab = "", ylab= "Density",
     main = "Truncated normal distributions")

## ptn example
x <- seq(-50, 10, length.out = 1e3)
```

```
mean <- 0
sd <- 1
lower <- 0
upper <- 5
dat1 <- ptnorm(x, 0, 1, 0, 5, log = TRUE)
```

`effectiveSize_hyper` *Effective sample size*

Description

`effectiveSize` calls **coda** effectiveSize to calculate effective posterior sample size.

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`	a samples object
`start`	starting iteration
`end`	ending iteraton
`digits`	printing digits
`verbose`	printing more information
`hyper`	a switch to extract hyper attribute and calculate it

Examples

```
#################################40
## effectiveSize example
#################################40
## Not run:
es1 <- effectiveSize_one(hsam[[1]], 1, 100, 2, TRUE)
es2 <- effectiveSize_one(hsam[[1]], 1, 100, 2, FALSE)
es3 <- effectiveSize_many(hsam, 1, 100, TRUE)
es4 <- effectiveSize_many(hsam, 1, 100, FALSE)
es5 <- effectiveSize_hyper(hsam, 1, 100, 2)
es6 <- effectiveSize(hsam, TRUE, 1, 100, 2, TRUE)
es7 <- effectiveSize(hsam, TRUE, 1, 100, 2, FALSE)
es8 <- effectiveSize(hsam, FALSE, 1, 100, 2, TRUE)
```

```
es9 <- effectiveSize(hsam, FALSE, 1, 100, 2, FALSE)
es10 <- effectiveSize(hsam[[1]], FALSE, 1, 100, 2, TRUE)

## End(Not run)
```

---

fac2df *Convert factor levels to a data frame*

Description

fac2df takes a model object created by BuildModel and returns a data frame with all combination of factor levels.

Usage

```
fac2df(x)
```

Arguments

x	a model object

Value

a data frame

Examples

```
model <- BuildModel(
 p.map     = list(a = "1", v = "1", z = "1", d = "1", sz = "1", sv = "1",
                  t0 = "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")

df1 <- fac2df(model)


model <- BuildModel(
            p.map     = list(A = "1", B = "1", v = "M", sv = "M", t0 = "1",
                        st0 = "1"),
            constants = c(st0 = 0, sv.false = 1),
            match.map = list(M = list(s1 = 1, s2 = 2)),
            factors   = list(S = c("s1", "s2")),
            responses = c("r1", "r2"),
            type      = "lnorm")

fac2df(model)
```

```
model <- BuildModel(
  p.map     = list(A = "1", B = "R", t0 = "1", mean_v = c("F", "M"),
                   sd_v = "M", st0 = "1"),
  match.map = list(M = list(s1 = 1, s2 = 2)),
  factors   = list(S = c("s1", "s2"), F = c("f1", "f2")),
  constants = c(sd_v.false = 1,st0 = 0),
  responses = c("r1", "r2"),
  type="norm")

fac2df(model)
##    S  F
## 1 s1 f1
## 2 s2 f1
## 3 s1 f2
## 4 s2 f2
```

---

gelman *Potential scale reduction factor*

Description

gelman calls **coda** gelman.diag to get R hats for one or a list of subjects. It calculates at the either data or hyper level.

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 inteval
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 spliting 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

```
## 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)
```

GetConstIdx *Whether a hyper-prior distribution is set constant*

Description

Check if a prior distribution for a location and scale parameter is set constant. testHyper checks if the parameter names in location and in scale match.

Usage

```
GetConstIdx(ppprior)
```

Arguments

ppprior	hyper-parameter prior distributions. First element is a location prior and second is a scale prior.

Value

isConstant gives a npar x 2 matrix;

Examples

```
model <- BuildModel(p.map = list(A = "1", B = "R", t0 = "1",
  mean_v = c("F", "M"), sd_v = "M", st0 = "1"),
  match.map = list(M = list(s1=1, s2=2)),
  factors   = list(S = c("s1", "s2"),F = c("f1", "f2")),
    constants = c(sd_v.false = 1, st0 = 0),
    responses = c("r1", "r2"),
    type      = "norm")
  npar <- length(GetPNames(model))

## Population distribution, rate effect on F
  pop.mean <- c(A=.4, B.r1=.6, B.r2=.8, t0=.3, mean_v.f1.true=1.5,
    mean_v.f2.true=1, mean_v.f1.false=0, mean_v.f2.false=0,
    sd_v.true = .25)
    pop.scale <-c(A=.1, B.r1=.1, B.r2=.1, t0=.05, mean_v.f1.true=.2,
      mean_v.f2.true=.2, mean_v.f1.false=.2, mean_v.f2.false=.2,
      sd_v.true = .1)
    p.prior <- BuildPrior(
        dists = rep("tnorm", npar),
        p1    = pop.mean,
        p2    = pop.scale*5,
        lower = c(0,0,0,.1,NA,NA,NA,NA,0),
        upper = c(NA,NA,NA,NA,NA,NA,NA,NA,NA))
    mu.prior <- BuildPrior(
        dists = rep("tnorm", npar),
        p1    = pop.mean,
        p2    = c(1,1,1,1,2,2,2,2,1),
        lower = c(0,0,0,.1,NA,NA,NA,NA,0),
        upper = c(NA,NA,NA,NA,NA,NA,NA,NA,NA))
    sigma.prior <- BuildPrior(
        dists = rep("beta", npar),
        p1    = c(A=1, B.r1=1, B.r2=1, t0=1, mean_v.f1.true=1,
          mean_v.f2.true=1, mean_v.f1.false=1, mean_v.f2.false=1,
          sd_v.true = 1),
          p2    = rep(1, npar))
    pp.prior <- list(mu.prior, sigma.prior)

    test1 <- GetConstIdx(pp.prior)
    test2 <- lapply(pp.prior,function(x){
      lapply(x,function(y){attr(y,"dist") == "constant"})})
```

GetGamma *Generate a Gamma Vector*

Description

This is part of DE-MCMC algorithm. This function generates a gamma vector for element-wise computation in Armadillo C++. This function is based on p242 ter Braak (2006) who cited Roberts and Rosenthal (2001)

Usage

```
GetGamma(npar, gammamult, hyper = FALSE)
```

Arguments

npar	number of parameters.
gammamult	a tuning parameter stands for for gamma mutation. Default value is 2.38.
hyper	a boolean switch, indicating to calculate hyper gamma

Value

a vector

Examples

```
pVec <- c(A = 1.51, b = 2.7, muv1 = 3.32, muv2 = 2.24, t_ND = 0.08,
          muw1 = 1.51, muw2 = 3.69, t_delay = 0.31, sv = 1, swt = 0.5)
gamma <- GetGamma(length(pVec), 2.38)
```

GetNsim *Get n-cell matrix*

Description

Constructs a matrix, showing how many responses to in each cell. It also checks if the format of n and ns conform to the standard.

Usage

```
GetNsim(model, n, ns)
```

Arguments

model	a model object
n	number of trials / responses.
ns	number of subjects.

Details

n can be:

1. one 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 design cell combination. See below for concrete examples.

Examples

```
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 = 1, st0 = 0),
  factors   = list(S = c("s1","s2")),
  responses = c("r1", "r2"),
  type      = "norm")

#######################30
## Example 1
#######################30
GetNsim(model, ns = 2, n = 1)
#      [,1] [,2]
# [1,]    1    1
# [2,]    1    1

#######################30
## Example 2
#######################30
n <- matrix(c(1:2), ncol = 1)
#      [,1]
# [1,]    1  ## subject 1 has 1 response for each cell
# [2,]    2  ## subject 2 has 2 responses for each cell

GetNsim(model, ns = 2, n = n)
#      [,1] [,2]
# [1,]    1    1
# [2,]    2    2

#######################30
## Example 3
#######################30
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
```

```
######################30
## Example 4
######################30
n <- matrix(c(1:4), nrow=2)
#      [,1] [,2]
# [1,]    1    3
# [2,]    2    4
ggdmc::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` *Return ns-npar matrix*

Description

Contructs a ns x npar matrix, indicating the true paramters used to simualte data. Each row represents a set of parameters for a participant. One should enter either a valid vector or matrix for true parameters (i.e., ps) or a list of (parameter) prior distributions (p.prior). When `p.prior` is supplied, true parameters are drawn from prior distributions.

Usage

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

Arguments

`x`	a model object
`ns`	number of subjects.
`prior`	a list of parameter prior distributions
`ps`	a vector or matirx. Each row indicates a set of true parameters for a participant.
`seed`	an integer specifying if and how the random number generator should be initialized.

Examples

```
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 arbritary 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(model)
```

Arguments

`model`	a model object

GetTheta0 *Extract Start Posterior Sample*

Description

Extract the theta's of the first MCMC iteration across chains and participants. Note that the ps array in DMC is a nchain x nsubject x nparameter array. Armadillo operates on slice (the third dimension), so chain dimension has to be on slice.

Usage

```
GetTheta0(samples)
```

Arguments

samples	posterior samples

Value

a nsubject x npar x nchain array

get_os *Retrieve OS information*

Description

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

Usage

```
get_os()
```

Examples

```
get_os()
## sysname
## "linux"
```

ggdmc *Bayeisan Computation for Cognitive Models*

Description

ggdmc evolves from Dynamic Models of Choice (DMC), using graphic styles of ggplot2, highly efficient computations of Armadillo C++ and Rcpp and connecting GPU parallel computation to **ppda** to make fitting complex cognitive models feasible. **ggdmc** uses the sampling technique of population-based Monte Chain Monte Carlo.

Author(s)

Yi-Shin Lin <yishin.lin@utas.edu.au>
Andrew Heathcote <andrew.heathcote@utas.edu.au>

References

Heathcote, A., Lin, Y.-S., Reynolds, A., Strickland, L., Gretton, M. & Matzke, D., (2018). Dynamic model of choice. *Behavior Research Methods*. https://doi.org/10.3758/s13428-018-1067-y.

Turner, B. M., & Sederberg P. B. (2012). Approximate Bayesian computation with differential evolution, *Journal of Mathematical Psychology*, 56, 375–385.

Ter Braak (2006). A Markov Chain Monte Carlo version of the genetic algorithm Differential Evolution: easy Bayesian computing for real parameter spaces. *Statistics and Computing*, 16, 239-249.

g_minus *Calculate Drift-diffusion Probability Density*

Description

`g_minus` and `g_plus` implement A1 to A4 equations in Voss, Rothermund, and Voss (2004). These equations calculate Ratcliff's drift-diffusion model (1978). This source codes are derived from Voss & Voss's fast-dm 30.2 in density.c.

Usage

```
g_minus(pVec)

g_plus(pVec)
```

Arguments

`pVec`	a 9-element parameter (double) vector. The user has to follow the sequence strictly. a, v, zr, d, sz, sv, t0, st0, RT, precision.

Details

Two parallel functions `g_minus_parallel` and `g_plus_parallel`, using OpenMP libraries to do numerical integration. They resolve the problem when high precision (> 10) is required.

References

Voss, A., Rothermund, K., & Voss, J. (2004). Interpreting the parameters of the diffusion model: A empirical validation *Memory and Cognition*, **32(7)**, 1206–1220.

Ratcliff, R (1978). A theory of memory retrieval. *Psychology Review*, **85(2)**, 59–108.

Examples

```
pvec1 <- c(a=2, v=2.5, zr=0.5, d=0, sz=0.3, sv=1, t0=0.3, st0=0,
           RT=.550, precision=2.5)
g_minus(pvec1) ## 0.04965882
g_plus(pvec1) ## 2.146265

pvec2 <- c(a=2, v=2.5, zr=0.5, d=.2, sz=0.3, sv=1, t0=0.3, st0=.1,
           RT=.550, precision=2.5)

g_minus(pvec2) ## 0.04194364
g_plus(pvec2)  ## 1.94957
```

---

hsumloglike *Add log-likelihoods across subjects at the hyper level*

Description

An external sum log likelihoods for hyper parameters.

Usage

```
hsumloglike(ps, pp, prior)
```

Arguments

`ps`	a nsubject x npar matrix
`pp`	a temporary pp.prior values extracted from phi. The temporary pp.prior has no attached parameter names. phi is two-element list. First element is a location array of nchain x npar x nmc; second element is a scale array of nchain x npar x nmc.
`prior`	prior distributions at data level

iseffective *Four pMCMC checking tools for automatic sampling*

Description

The function tests whether MCMC chains have drawn enough samples.

Usage

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

Arguments

x	posterior samples
minN	minimal effective sample sizes
nfun	mean or median function
verbose	print more information

isflat *Four pMCMC checking tools for automatic sampling*

Description

The function tests whether MCMC chains converge prematurelly:

Usage

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

Arguments

x	posterior samples
p1	the range of the head of MCMC chains
p2	the range of the tail of the MCMC chains
cut_location	how far away a location chains been considered as stuck
cut_scale	how far away a scale chains been considered as stuck
verbose	print more information

ismanymodels *Test whether 'model' object has many models*

Description

Test whether 'model' object has many models

Usage

```
ismanymodels(model, ns = NA)
```

Arguments

model	a model object
ns	number of subjects.

ismixed *Four pMCMC checking tools for automatic sampling*

Description

The function tests whether MCMC chains mixed well.

Usage

```
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)

isstuck *Four pMCMC checking tools for automatic sampling*

Description

The function tests whether MCMC chains encounter a parameter region difficult to search (ie get stuck):

Usage

```
isstuck(x, cut = 10, verbose = FALSE)
```

Arguments

x	posterior samples
cut	the criteria for suggesting abnormal chains found
verbose	print more information

likelihood_norm *Calculate log likelihoods*

Description

These function calculate log likelihoods. `likelihood_rd` implements the equations in Voss, Rothermund, and Voss (2004). These equations calculate diffusion decision model (Ratcliff & Mckoon, 2008). Specifically, this function implements Voss, Rothermund, and Voss's (2004) equations A1 to A4 (page 1217) in C++.

Usage

```
likelihood_norm(x, data, min_lik = 1e-10)

likelihood_norm_pda(x, data, min_lik = 1e-10)

likelihood_rd(x, data, min_lik = 1e-10)

likelihood_cnorm(x, data, min_lik = 1e-10)
```

Arguments

x	a parameter vector
data	data model instance
min_lik	minimal likelihood.

Value

a vector

References

Voss, A., Rothermund, K., & Voss, J. (2004). Interpreting the parameters of the diffusion model: An empirical validation. *Memory & Cognition*, **32(7)**, 1206-1220.

Ratcliff, R. (1978). A theory of memory retrival. *Psychological Review*, **85**, 238-255.

Examples

```
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, 1e3,  ps = p.vector)
dmi <- BuildDMI(dat, model)
den <- likelihood_norm(p.vector, dmi)
## den <- likelihood_norm_pda(p.vector, dmi) ## This takes more than 1 s, so commented out

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, 1e2, ps = p.vector)
dmi <- BuildDMI(dat, model)
den <- likelihood_rd(p.vector, dmi)
```

MakeForce *Create a vector for re-calculation*

Description

The is a PDA function. It creates an index vector, indicating which iteration to recalculate likelihoods.

Usage

```
MakeForce(x, nth)
```

Arguments

x	posterior samples
nth	every nth step to recalculate

Examples

```
## Not run:
  MakeForce(hsam[[1]], 3)

## End(Not run)
```

`MakeLevelArray` *Create an array of all factorial combinations of factor levels*

Description

Take a list storing one or multiple string vectors and glues the strings in these vectors with a dot symbol.

Usage

```
MakeLevelArray(x = NA)
```

Arguments

x	a list storing one or multiple factor vectors. Each vector can have different numbers of level. For example, `f <- list(E = c("nc", "wc"), S = c("n", "w", "pn", "pw"))`

Value

a table showing the combinations of factor levels.

Examples

```
## Example 1
factors <- list(S = c("s1", "s2"))
MakeLevelArray(factors)
##   S
##  s1   s2
## "s1" "s2"

factors <- list(S = c("left", "right"))
MakeLevelArray(factors)
```

```
##      S
##   left   right
##  "left" "right"

## Example 2
factors <- list(A = c("00", "45", "90", "135", "180"),
                S = c("mirror", "normal"))
MakeLevelArray(factors)
##       S
## ------------------------------40
##   A        mirror       normal
## ------------------------------40
##  00   "00.mirror"  "00.normal"
##  45   "45.mirror"  "45.normal"
##  90   "90.mirror"  "90.normal"
## 135  "135.mirror" "135.normal"
## 180  "180.mirror" "180.normal"

## Example 3
factors <- list(E = c("nc", "wc"),
                S = c("n", "w", "pn", "pw"))
MakeLevelArray(factors)

##         S
## --------------------------------40
## E       n      w      pn      pw
## --------------------------------40
## nc "nc.n" "nc.w" "nc.pn" "nc.pw"
## wc "wc.n" "wc.w" "wc.pn" "wc.pw"
```

`maker` *Canonical Linear Ballistic Accumulation/Accumualtor Model*

Description

`makeR` stands for making/generating/simulating responses from a LBA model. `make_r` and `make.r` use C++ function. These make `r`, `_r`, `.r` functions are essentially `rLBA`, including `rlba_norm`. They uses a LBA model with parameters, b, A, mean_v, sd_v and t0 (no st0) to generate choice RT random deviates.

Usage

```
maker(drifts, n, b, A, n_v, t0, st0 = 0, seed = NULL,
  return_ttf = FALSE)
```

Arguments

`drifts`	a n x n_v drift rate matrix. It can be a vector with 2 or more elements. n is the numbers of observation. n_v is the numbers of response/accumulator.

n	numbers of observation/model simulations. This must be a scalar.
b	decision threshold, a vector or a scalar.
A	start point upper bound, a vector of a scalar.
n_v	numbers of response/accumulator, an integer. Note n_v must match the length/size of drifts vector.
t0	nondecision time, a vector or a scalar.
st0	nondecision time variation, a vector of a scalar. It is the upper bound of a uniform distribution for t0 variability.
seed	an integer specifying if and how the random number generator should be initialized.
return_ttf	a boolean switch indicating if return RTs for all accumulators. When return_ttf is TRUE, a n_v x n ttf matrix is returned.

Details

make_v draws drift rate from normal or truncated normal distribution. Each trial is stored as a row and each column is a drift rate for an accumulator. You need to transpose drift rates generated by make_v for makeR.

make.r is a wrapper function of make_r. You may need to use ":::" to call make.r, because of S3 method naming convention. If you call make_r directly, beware it returns C index and is only a numeric matrix. It does not carry a string vector for the column names, RTs and responses. See timing test to see why it might be a good idea not to return it as a data frame. rlbaCnorm is R version of correlated LBA model.

rlba_norm adds checks and calls make_v and make_r. rlba_norm is only slightly quicker than make_r.

n1PDFfixedt0 is defective density function for the fisrt node LBA model. Defective means its probability does not normally normalize to 1. Only the probabilities from all nodes/accumulators add to 1. n1PDFfixedt0 is equation (3) on page 159 in Brown and Heathcote (2008). This equation assumes st0 is 0.

fptcdf and fptpdf are distribution and density functions with four parameters A, b, v and sv, assuming t0 is zero. fptcdf and fptpdf are respectively equation (1) and equation (2) on page 159 in Brown and Heathcote (2008).

Value

make_r gives either a time-to-finish (ttf) matrix or a n x 2 matrix, storing RTs (first column) and responses (second column). n equals to number of model simulations. ttf is a n_v x n matrix with RTs from all accumulators.

mcmc_list.model *Create a MCMC list*

Description

Create a MCMC list

Usage

```
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

n1PDF_cnorm *Likelihood function for correlated accumulator model*

Description

Calculate first node probability densities of the correlated accumulator model

Usage

```
n1PDF_cnorm(x, A, b, t0, mean_v, sd_v, st0, corr_v, n, h, debug = FALSE)
```

Arguments

x	response time vector
A	start point variability
b	threshold
t0	nondecision time
mean_v	mean drift rate vector
sd_v	standard deviation of the drift rates
st0	nondecision time variability
corr_v	correlation among accumulators
n	number of simulations
h	kernel bandwidth
debug	debugging?

n1PDF_gpu *A Rcpp connection to the GPU-based LBA n1PDF in ppda package*

Description

This is an Rcpp function calling a n1PDF_gpu function defined in the global environment, which connects to an interanl ppda simulated-based LBA density function.

Usage

```
n1PDF_gpu(x, A, b, mean_v, sd_v, t0, n, nthread, gpuid, bw, debug)
```

Arguments

x	response time vector
A	start point variability
b	threshold
mean_v	mean drift rate vector
sd_v	standard deviation of the drift rates
t0	nondecision time
n	number of simulations
nthread	thread number to launch in GPU
gpuid	which gpu card to use, starting from 0.
bw	kernel bandwidth
debug	debugging?

n1PDF_plba0_gpu *A Rcpp connection to the GPU-based PLBA n1PDF type 0 and type 1*

Description

This is an Rcpp function calling a n1PDF_plba0_gpu function defined in the global environment, which connect to an interanl ppda simulated-based PLBA density function.

Usage

```
n1PDF_plba0_gpu(x, A, b, mean_v, sd_v, t0, mean_w, rD, swt, n, nthread,
  gpuid, bw, debug)

n1PDF_plba1_gpu(x, A, b, mean_v, sd_v, t0, mean_w, rD, swt, n, nthread,
  gpuid, bw, debug)
```

Arguments

x	response time vector
A	start point variability
b	threshold
mean_v	mean drift rate vector for the first piece of evidence accumulation.
sd_v	standard deviation of the drift rates
t0	nondecision time
mean_w	mean drift rate vector for the second piece of evidence accumulation.
rD	rate delay time (s)
swt	switch time (s)
n	number of simulations
nthread	thread number to launch in GPU
gpuid	which gpu card to use, starting from 0.
bw	kernel bandwidth
debug	debugging?

---

pairs.model *Correlation Matrix*

Description

Plot the correlation matrix of posterior samples,

Usage

```
## S3 method for class 'model'
pairs(x, start = 1, end = NA, ...)
```

Arguments

x	posterior samples
start	start from which iteration.
end	end at which iteration.
...	other arguments

PickChains *Draw n other chains and shuffle them*

Description

This is part of DE-MCMC algorithm. `PickChains` draws `n` chains out of `length(chains)` chains, excluding the kth chain. `GetSubchains` is used in `migration` operator. It draws a subset of chains in `nchain` chains.

Usage

```
PickChains(k, nchain, chains)

GetSubchains(nchain)

SelectEmigrants(ngroup, k)
```

Arguments

`k`	the kth processed chain. Must be an integer within the range of 0 to `nchain - 1`. No check for errorly using R index.
`nchain`	number of chains to draw.
`chains`	an integer vector, indicating chain index, e.g., 0:23
`ngroup`	number of distributed groups

Details

`Getsubchains` is part of the Migration algroithms. It does two-step shuffling. In step 1, it selects a number l (integer) uniformly between 1 and k to be the number of subpopulations for migration. In step 2, it generates the chain index (0 to nchain - 1) and lastely it shuffles them

Value

a column vector

Examples

```
chains <- 0:23

## Presuming current processing chain is the 1st chain (C index = 0)
## pick 2 chains out of 24 chains, excluding current chain.
PickChains(0, 2, chains)

## Example outputs
##      [,1]
## [1,]   17
## [2,]   12
```

```
##      [,1]
## [1,]    2
## [2,]    5
##      [,1]
## [1,]    5
## [2,]    3
##      [,1]
## [1,]   10
## [2,]    8
##      [,1]
## [1,]   15
## [2,]    8

## get a random number of subchains
GetSubchains(24)
##       [,1]
##  [1,]    0
##  [2,]    3
##  [3,]    5
##  [4,]    9
##  [5,]   10
##  [6,]   12
##  [7,]   14
##  [8,]   15
##  [9,]   18
## [10,]   20
## [11,]   21
## [12,]   22
```

`PickStuck` *Which chains get stuck*

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. `unstick` manually removes stuck chains from posterior samples.

Usage

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

Arguments

`x`	posterior samples
`hyper`	whether x are hierarhcial samples

cut	a criterion deciding if a chain is stuck.
start	start to evaluate from which iteration.
end	end at which iteration for evaeuation.
verbose	a boolean switch to print more information
digits	print how many digits. Default is 2

Value

PickStuck gives an index vector; unstick gives a DMC sample.

Examples

```
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:
dat <- simulate(model, 30, ps = p.vector)
dmi <- BuildDMI(dat, model)
sam <- run(StartNewsamples(5e2, dmi, p.prior))
bad <- PickStuck(sam)

## End(Not run)
```

---

plot_prior *Plot Prior Distributions*

Description

plot_prior plots the ith member of the list created by BuildPrior. If trans = TRUE, the function will plot on natural (logarithmic) scale using transform specified in attr(p.prior[[i]], "trans"). plot.prior plots all parameters at once.

Usage

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

## S3 method for class 'prior'
plot(x, save = FALSE, ...)
```

Arguments

i	an integer or a character string indicating to plot which parameter
prior	a list of list storing prior setting.
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 throught dot dot dot.
x	prior distributions

Details

NOTE: the tran function is not thoroughtly checked. Use it with your own risk. `plot_prior` checks if any of the elements in `trans` is `NA`. It then checks if `natural` is set `TRUE` (by default it's `TRUE`). If `natural` is set also `TRUE` (i.e., the user wants to do transformation), it then checks what has been set in the "untrans" attribute for the i'th parameter. Otherwise, its default is set all parameters as `identity`.

Examples

```
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))

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

print(p.prior)
plot(p.prior)

## require(ggplot2)
## p2 <- ggplot(d, aes(x = xpos, y = ypos)) + geom_line() +
##       facet_wrap(~gpvar, scales="free") + theme_bw(base_size =14)
```

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)
```

profile.model *Profile a model object*

Description

This function produces data for profiling model likelihoods based on a data model instance (ie `fitted`).

Usage

```
## S3 method for class 'model'
profile(fitted, pname, minp, maxp, p.vector,
  npoint = 100, digits = 2, ylim = NA, nthread = 32, ...)
```

Arguments

fitted	a data model instance
pname	indicate which parameter in theta to plot. For example, in a LBA model with a pVec <- c(A="1",B="1",mean_v="M",sd_v="1",t0="1",st0="1"), one can assign pname <- "A" to ask profile to profile *A* parameter
minp	lower bound for pname parameter
maxp	upper bound for pname parameter
p.vector	a parameter vector. Use Lognromal LBA model as an example, pVec <- c(A = .25, B = .35, meanlog.
npoint	grid for p.name parameter
digits	print out how many digits
ylim	y range
nthread	number of thread in a GPU block. This argument works for GPU-based PDA probability density functions)
...	additional optional arguments.

Details

The argument, pname indicates which model parameter to profile. For example, if we want to profile the boundary separation *a* in a DDM, we extract it from a p.vector and calculates its marginal likelihoods based on the data model instance on a grid of n.point.

Briefly, the function sets a range for the points on the x axis, which is the values for the profiled parameter (e.g., *a*). Then it initiates a log-likelihodd vector with length of n.point and 0 everywhere. Next, it keeps the other parameters in the model fixed and changes only the target parameter values to ps[i]. After that, it calculates their sum log-likelihoods. Finally, it stores the log-likelihoods in the i position of the ll vector.

Examples

```
model <- BuildModel(
        p.map     = list(a = "1",v = "1",z = "1", d = "1", sz = "1",
                    sv = "1", t0 = "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.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),
  lower=c(0,-5, 0, 0, 0, 0),
  upper=c(5, 7, 2, 2, 2, 2))

p.vector <- c(a=1,v=1, z=0.5, sz=0.25, sv=0.2,t0=.15)
dat <- simulate(model, 1e2, ps = p.vector)
dmi <- BuildDMI(dat, model)

## ------------------------------40
par(mfrow=c(2,3));
profile(dmi, "a",  .1,   2, p.vector)
profile(dmi, "v",  .1,   2, p.vector)
profile(dmi, "z",  .2,  .8, p.vector)
profile(dmi, "sz", .1,  .9, p.vector)
profile(dmi, "sv", .1,   2, p.vector)
profile(dmi, "t0", .01, .5, p.vector)
par(mfrow=c(1,1));
```

`random` *Generate random variates of various cognitive models*

Description

A wrapper function to `rd`, `norm`, `norm_pda`, `norm_pda_gpu`, `plba0_gpu`, `plba1`, `plba_gpu`, `plba2`, `plba3`, `lnr`, and `cnorm` models.

Usage

```
random(type, pmat, n, seed = NULL)
```

Arguments

`type`	a character string indicating the model type
`pmat`	a matrox of accumulator x parameter
`n`	number of simulations
`seed`	an integer specifying if and how the random number generator should be initialized.

rca *Generate random number from a correlated accumulator model*

Description

Generate random number from a correlated accumulator model

Usage

```
rca(n, A, b, t0, mean_v, sd_v, st0 = 0, corr_v = 0,
  return_ttf = FALSE)
```

Arguments

n	a data frame stored data
A	start point variability
b	threshold
t0	nondecision time
mean_v	mean drift rate vector for the first piece of evidence accumulation
sd_v	standard deviation of the drift rates
st0	nondecision time variability
corr_v	correlation among accumulators
return_ttf	a boolean switch to return time to finish matrix

remove_t0 *Substract nondecision times from response times*

Description

This function minus t0's from RTs.

Usage

```
remove_t0(x, t0)

removet0(x, t0)
```

Arguments

x	a response time vector
t0	a numeric scalar

Value

a vector

Examples

```
rt <- rlnorm(10) + .2
dt <- removet0(rt, .2)
all.equal(dt, rt - .2)
```

---

rlnr *Generate Random Choice RT Data from LNR Model*

Description

Race among nacc accumulators, using log-normal race model

Usage

```
rlnr(n, meanlog, sdlog, t0, st0 = 0)
```

Arguments

n	numbers of observation
meanlog	a n_acc length vector or a n_acc x n matrix. mean of the distribution on the log scale without default value
sdlog	a n_acc length vector or a n_acc x n matrix. Standard deviation of the distribution on the log scale without default value.
t0	a scalar, a vector of length number of accumulators or a matrix with 1 row per accumulator, when start time differ on each trial
st0	range of non-decision time variability, must be a scalar, as the same variability is assumed in a common encoding/production stage

Value

a matrix

Examples

```
## A simple demo
pmat <- matrix(c(-1, 0, 1, 1, .2, .2, 0, 0), 2)
set.seed(123)
dat0 <- rlnr(4, pmat[,1], pmat[,2], pmat[,3], pmat[1,4])
##           [,1] [,2]
## [1,] 0.4100361    0
## [2,] 0.4922407    0
## [3,] 1.7855260    1
## [4,] 0.4822220    1
```

```
##
## Three accumulators
n <- 1e5
meanlog <- c(.5, .75, 1);
sdlog <- c(1,1,1)
t0 <- c(.2,1,1)
set.seed(123)
dat1 <- rlnr(n, meanlog, sdlog, t0)
table(dat1[,2])
hist(dat1[,1], breaks = "fd", main = "", xlab ="")

## t0 has one element only
t0 <- .2
set.seed(123)
dat2 <- rlnr(n, meanlog, sdlog, t0)
table(dat2[,2])
hist(dat2[,1], breaks = "fd", freq = FALSE, main = "", xlab ="")
## check t0 noise
st0 <- 1
set.seed(123)
dat3 <- rlnr(n, meanlog, sdlog, t0, st0)
table(dat3[,2])
hist(dat3[,1], breaks = "fd", freq = FALSE, main = "", xlab ="")
```

---

rplba0 *Piecewise LBA model*

Description

Density and random generation of the PLBA Model Type 0, 1, and 2.

Usage

```
rplba0(n, A, b, t0, mean_v, mean_w, sd_v, rD, swt, ncore = 1L,
  debug = FALSE)

rplba1(n, A, b, t0, mean_v, mean_w, sd_v, rD, swt, ncore = 1L,
  debug = FALSE)

rplba2(n, A, b, t0, mean_v, mean_w, sd_v, sd_w, rD, swt, ncore = 1L,
  debug = FALSE)

rplba3(n, A, B, C, mean_v, mean_w, sd_v, sd_w, rD, tD, swt, t0)

n1PDF_plba1(x, A, b, mean_v, sd_v, t0, mean_w, rD, swt, n, h, ncore, debug)

n1PDF_plba2(x, A, b, mean_v, sd_v, t0, mean_w, sd_w, rD, swt, n, h, ncore,
  debug)
```

```
n1PDF_plba3(x, n, A, B, C, mean_v, sd_v, mean_w, sd_w, rD, tD, swt, t0, h)

rplba1R(n, A, b, t0, mean_v, mean_w, sd_v, rD, swt)

rplba2R(n, A, b, t0, mean_v, mean_w, sd_v, sd_w, rD, swt)

rplba3R(n = 10, pVec = c(A1 = 1.5, A2 = 1.5, B1 = 1.2, B2 = 1.2, C1 =
  0.3, C2 = 0.3, v1 = 3.32, v2 = 2.24, w1 = 1.51, w2 = 3.69, sv1 = 1, sv2 =
  1, sw1 = 1, sw2 = 1, rD = 0.3, tD = 0.3, swt = 0.5, t0 = 0.08))
```

Arguments

n	number of observations.
A	upper bound of start point. It can be an integer or a 2-element vector.
b	response threshold. It can be an integer or a 2-element vector.
t0	nondecision time (in second)
mean_v	stage 1 mean drift rate. It must be a 2-element vector
mean_w	stage 2 mean drift rate. It must be a 2-element vector
sd_v	common standard deviation of the piece 1 drift rates. If sd_w is not present, this will also be used as the piece 2 drift rate standard deviation, which cannot be negative.
rD	drift rate delay (in second)
swt	switch time (in second)
ncore	number of CPU cores for running Open MP.
debug	internal debug switch
sd_w	standard deviation of the piece 2 drift rates
B	first stage traveling distance
C	second stage traveling distance
tD	threshold delay time
x	vector of quantiles.
h	bandwidth for the kernel function
pVec	PLBA parameter vector

Value

a [RT R] matrix (C++) or a data frame (R)

References

Holmes, R. W., Trueblood, J. & Heathcote, A. (2016). A new framework for modeling decisions about changing information: The Piecewise Linear Ballistic Accumulator model. *Cognitive Psychology*, 85, 1–29, doi: http://dx.doi.org/10.1016/j.cogpsych.2015.11.002.Approximate

Examples

```
##############################################################################80
## rplba1
##############################################################################80
## Not run:
n <- 2^20; n
A <- 1.5
b <- 2.7
mean_v <- c(3.3, 2.2)
mean_w <- c(1.5, 3.7)
sd_v <- c(1, 1)
rD    <- .3
swt   <- .5
t0    <- .08
ncore <- 12
dat1 <- rplba1R(n, A, b, t0, mean_v, mean_w, sd_v, rD, swt)
dat2 <- rplba1(n, A, b, t0, mean_v, mean_w, sd_v, rD, swt, ncore)
dat3 <- ppda::rplba1(n, A, b, t0, mean_v, mean_w, sd_v, rD, swt)

dat1r1 <- dat1[dat1[, 2] == 1, 1]
dat1r2 <- dat1[dat1[, 2] == 2, 1]
dat2r1 <- dat2[dat2[, 2] == 1, 1]
dat2r2 <- dat2[dat2[, 2] == 2, 1]
dat3r1 <- dat3[dat3[, 2] == 1, 1]
dat3r2 <- dat3[dat3[, 2] == 2, 1]

xlim <- c(0, 3)
## Check if two methods produce SPDF overlaping with each other
par(mfrow = c(4, 2), mar = c(4, 5.3, 0.82, 1))
hist(dat1r1, breaks = "fd", freq = FALSE, main = "Choice1 R", xlim = xlim)
hist(dat1r2, breaks = "fd", freq = FALSE, main = "Choice2 R", xlim = xlim)
hist(dat2r1, breaks = "fd", freq = FALSE, main = "Choice1 C++", xlim = xlim)
hist(dat2r2, breaks = "fd", freq = FALSE, main = "Choice2 C++", xlim = xlim)
hist(dat3r1, breaks = "fd", freq = FALSE, main = "Choice1 GPU", xlim = xlim)
hist(dat3r2, breaks = "fd", freq = FALSE, main = "Choice2 GPU", xlim = xlim)

par(mfrow = c(1, 2))
hist(dat1r1, breaks = "fd", freq = FALSE, main = "Choice1 R, C++, & GPU",
  xlim = xlim, ylim = c(0, 3))
hist(dat2r1, breaks = "fd", freq = FALSE, add = TRUE, col = "lightblue")
hist(dat3r1, breaks = "fd", freq = FALSE, add = TRUE, col = "lightgreen")

hist(dat1r2, breaks = "fd", freq = FALSE, main = "Choice2 R, C++ & GPU",
  xlim = xlim, ylim = c(0, 3))
hist(dat2r2, breaks = "fd", freq = FALSE, add = TRUE, col = "lightblue")
hist(dat3r2, breaks = "fd", freq = FALSE, add = TRUE, col = "lightgreen")

## End(Not run)

##############20
## rplba2    ##
##############20
```

```
## Not run:
n <- 2^15
ncore <- 4
A <- c(1.5, 1.5)
b <- c(2.7, 2.7)
mean_v <- c(3.3, 2.2)
mean_w <- c(1.5, 3.7)
sd_v <- c(1, 1)
sd_w <- c(1, 1)
rD <- .3
swt <- .5
t0 <- .08
dat1 <- rplba2R(n, A, b, t0, mean_v, mean_w, sd_v, sd_w, rD, swt)
dat2 <- rplba2(n, A, b, t0, mean_v, mean_w, sd_v, sd_w, rD, swt, ncore)
dat3 <- rplba2(n, A, b, t0, mean_v, mean_w, sd_v, sd_w, rD, swt)
dat4 <- rplba2_test(n, A, b, t0, mean_v, mean_w, sd_v, sd_w, rD, swt)

dat1r1 <- dat1[dat1[, 2] == 1, 1]
dat1r2 <- dat1[dat1[, 2] == 2, 1]
dat2r1 <- dat2[dat2[, 2] == 1, 1]
dat2r2 <- dat2[dat2[, 2] == 2, 1]
dat3r1 <- dat3[dat3[, 2] == 1, 1]
dat3r2 <- dat3[dat3[, 2] == 2, 1]
dat4r1 <- dat4[dat4[, 2] == 1, 1]
dat4r2 <- dat4[dat4[, 2] == 2, 1]

wesanderson::wes_palette("Royal1")
palettes  <- wesanderson::wes_palettes$GrandBudapest
palettes2 <- wesanderson::wes_palettes$GrandBudapest2
xlim <- c(0, 3)
## Check if two methods produce SPDF overlaping with each other
par(mfrow = c(4, 2), mar = c(4, 5.3, 0.82, 1))
hist(dat1r1, breaks = "fd", freq = FALSE, main = "Choice1 R", xlim = xlim)
hist(dat1r2, breaks = "fd", freq = FALSE, main = "Choice2 R", xlim = xlim)
hist(dat2r1, breaks = "fd", freq = FALSE, main = "Choice1 C++", xlim = xlim)
hist(dat2r2, breaks = "fd", freq = FALSE, main = "Choice2 C++", xlim = xlim)
hist(dat3r1, breaks = "fd", freq = FALSE, main = "Choice1 GPU", xlim = xlim)
hist(dat3r2, breaks = "fd", freq = FALSE, main = "Choice2 GPU", xlim = xlim)
hist(dat4r1, breaks = "fd", freq = FALSE, main = "Choice1 test", xlim = xlim)
hist(dat4r2, breaks = "fd", freq = FALSE, main = "Choice2 test", xlim = xlim)

par(mfrow = c(1, 2))
hist(dat1r1, breaks = "fd", freq = FALSE, main = "Choice1 R, C++, & GPU",
  xlim = xlim, ylim = c(0, 3))
hist(dat2r1, breaks = "fd", freq = FALSE, add = TRUE, col = palettes[1])
hist(dat3r1, breaks = "fd", freq = FALSE, add = TRUE, col = palettes[2])
hist(dat4r1, breaks = "fd", freq = FALSE, add = TRUE, col = palettes[4])

hist(dat1r2, breaks = "fd", freq = FALSE, main = "Choice2 R, C++ & GPU",
  xlim = xlim, ylim = c(0, 3))
hist(dat2r2, breaks = "fd", freq = FALSE, add = TRUE, col = palettes2[1])
hist(dat3r2, breaks = "fd", freq = FALSE, add = TRUE, col = palettes2[2])
hist(dat4r2, breaks = "fd", freq = FALSE, add = TRUE, col = palettes2[3])
```

```
## End(Not run)
```

`rprior_scalar` *Parameter Prior Distributions*

Description

Probability density functions and random generation for parameter prior distributions.

Usage

```
rprior_scalar(prior)

rprior_mat(prior, n)

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

```
p.prior <- BuildPrior(
 dists = c("tnorm", "tnorm", "beta", "tnorm", "beta", "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))

rprior(p.prior, 9)
##               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 0.96520848
## [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,] 1.32520281 -0.07328408 0.2051155  2.4040387 0.9663111 0.06127237
## [6,] 0.49628528 -0.19374770 0.5142829  2.1452972 0.4335482 0.38410626
## [7,] 0.03655549  0.77223432 0.1739831  1.4431507 0.6257398 0.63228368
## [8,] 0.71197612 -1.15798082 0.8265523  0.3813370 0.4465184 0.23955415
## [9,] 0.38049166  3.32132034 0.9888108  0.9684292 0.8437480 0.13502154

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))
```

`run` *Run model fits*

Description

This function fit a hierarchical or a fixed-effect model, using Bayeisan sampling. We use pMCMC, with a suite of DE-MCMC, DGMC, and simply, crossover (i.e., DE-MC), mutation, or migration operators. Note that the latter two operators essentially are random-walk Metroplolis, so they will be very inefficient, if been applied alone, even with our fast C++ implementation.

Usage

```
run(samples, report = 100, ncore = 1, pm = 0, qm = 0, hpm = 0,
  hqm = 0, gammamult = 2.38, ngroup = 5, force = FALSE,
  sampler = "DE-MCMC", slice = FALSE)

CheckConverged(samples)
```

Arguments

`samples`	a sample list generated by calling DMC's samples.dmc.
`report`	how many iterations to return a report
`ncore`	parallel core for run_many
`pm`	probability of migration
`qm`	probability of mutation
`hpm`	probability of migration at the hyper level
`hqm`	probability of mutation at the hyper level
`gammamult`	a tuning parameter, affecting the size of jump
`ngroup`	number of distributed groups
`force`	set force to FALSE for turning off recalculation of PDA. Set it as an integer between 1 and 10, forcing to re-calculate new likelihood, every e.g., 1, 2, 3 step.
`sampler`	which sampler to run MCMC, "DE-MCMC" or "DGMC"
`slice`	use for debugging blocked sampling

run_many *Fit a Bayesian Model to multiple Participants*

Description

Use either DE-MCMC or DGMC sampler to fit independent Bayesian model to many participants.

Usage

```
run_many(samples, report, ncore, pm, qm, gammamult, ngroup, force, sampler,
  slice)
```

Arguments

samples	a initialized samples list. Each element should contain samples for a participant.
report	progress report intervel
ncore	number of CPU cores
pm	probability of migration
qm	probability of mutation
gammamult	turning parameter for crossover sampler
ngroup	number of independent groups
force	PDA re-calculate interval
sampler	a string indicating to use which sampler
slice	a Boolean switch to do slice sampling

run_one *Fit a Bayesian Model to a Single Participant*

Description

Use either DE-MCMC or DGMC sampler to fit Bayesian model to a participant

Usage

```
run_one(samples, report, pm, qm, gammamult, ngroup, force, sampler, slice)
```

Arguments

samples	a initialized sample
report	progress report intervel
pm	probability of migration
qm	probability of mutation
gammamult	tuning parameter of the crossover sampler
ngroup	number of independent groups
force	PDA re-calculate interval
sampler	a string indicating to use which sampler
slice	a Boolean switch to debug blocked sampling

Value

Bayesian samples

---

score *Scoring RT data*

Description

A convenient function to calculate mean, interquantile range, standard deviation for correct and error RTs

Usage

```
score(x, digits = 2)
```

Arguments

x	a parameter vector
digits	printing digits

simulate.model *Simulate RT Data*

Description

Simulate stochastic responses either for one subject or multiple subjects. The simulation is based on the model object. For one subject, the user must supply true parameters, p.vector at ps argument. For multiple subjects, the user can supply a matrix (or a row vector), indicating true parameters for each subject, separately on each row (via ps argument). This is the fixed-effect model. If the user wants to simulate from a random-effect (i.e., hierarchical) model, in which case p.prior must be supplied and ps will be ignored. Note in some cases, a random-effect model may fail to draw data from the model, because true parameters are drawn from p.prior and a specific model, like DDM, may has certain ranges from different parameters.

Usage

```
## 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 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 design subject x design cell combination.
seed	an integer specifying if and how the random number generator should be initialized.
nsub	number of subjects
prior	parameter priors. A list of distributions based on which the true parameters fro each subject are drawn. It is usually created by BuildPrior and will be saved as "p.prior" attribute.
ps	p.vector matrix. Each row represent a subject.
...	additional optional arguments.

Details

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.

Value

a data frame

Examples

```
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")
```

---

spdf *Simulated likelihood functions for RT models*

Description

CPU-based PDA

Usage

```
spdf(x, RT, n, h_in, debug = FALSE)
```

Arguments

x	empirical data
RT	simulated resposne times
n	number of model simulations
h_in	kernel bandwidth
debug	debugging?

Value

a vector

`StartNewsamples` *Initialize New Samples*

Description

These functions use prior distributions, either from `p.prior` or joinly from `p.prior` and `pp.prior` in the case of hierarchical models to generate over-dispersed initial parameter values.

Usage

```
StartNewsamples(nmc, data = NULL, prior = NULL, thin = 1,
  nchain = NULL, rp = 0.001)

RestartSamples(nmc, samples = NULL, thin = NULL, rp = 0.001,
  add = FALSE)

StartManynewsamples(nmc, data = NULL, prior = NULL, thin = 1,
  nchain = NULL, rp = 0.001)

RestartManysamples(nmc, samples = NULL, thin = NULL, rp = 0.001,
  add = FALSE)

StartNewHypersamples(nmc, data = NULL, prior = NULL, ppprior = NULL,
  thin = 1, rp = 0.001, nchain = NULL)

RestartHypersamples(nmc, samples = NULL, thin = NULL, rp = 0.001,
  add = FALSE)
```

Arguments

`nmc`	numbers of Monte Carlo samples / iterations.
`data`	a data model instance
`prior`	parameter prior distributions from `BuildPrior`.
`thin`	thinning length.
`nchain`	numbers of Markov chains. Default is 3 times the numbers of model parameters.
`rp`	DE-MCMC tuning parameter to generate random noise either from uniform or Gaussian distribution.
`samples`	a collection fo posterior samples.
`add`	add more samples onto an existing samples
`ppprior`	hyper parameter prior distributions from `BuildPrior`. This must be a set of location and scale hyper prior distributions.

Examples

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

## m1 is "model" class
class(m1)
## [1] "model"

pVec <- c(a = 1, v.f1 = 1, v.f2 = 1.5, z = .5, sz = .25, sv = .2, t0 = .15)
dat  <- simulate(m1, nsim = 1e2, ps = pVec)
str(dat)
## 'data.frame': 400 obs. of  4 variables:
## $ S : Factor w/ 2 levels "s1","s2": 1 1 1 1 1 1 1 1 1 1 ...
## $ F : Factor w/ 2 levels "f1","f2": 1 1 1 1 1 1 1 1 1 1 ...
## $ R : Factor w/ 2 levels "r1","r2": 1 1 1 2 1 1 1 1 2 1 ...
## $ RT: num  0.26 0.255 0.572 0.25 0.518 ...

dmi1 <- BuildDMI(dat, m1)
npar <- length(GetPNames(m1))

p.prior <- BuildPrior(
   dists = rep("tnorm", npar),
   p1    = c(a=2,  v.f1=2.5, v.f2=1.25, z=.5, sz=.3, sv=1,  t0=.3),
   p2    = c(a=.5, v.f1=.5,  v.f2=.35,  z=.1, sz=.1, sv=.3, t0=.05),
   lower = c(0,-5, -5, 0, 0, 0, 0),
   upper = c(5, 7,  7, 2, 2, 2, 2))

## Set up a new DMC sample with 16 iteration. The default thin is 1
sam0 <- StartNewsamples(nmc = 16, data = dmi1, prior = p.prior)
sam0$nmc
## [1] 16

sam1 <- RestartSamples(16, sam0)
sam1$nmc
## [1] 16
sam1 <- RestartSamples(16, sam1, add = TRUE)
sam1$nmc
## [1] 32

######################28
## Hierarchical      ##
######################28
model <- BuildModel(
        p.map     = list(A = "1", B = "R", t0 = "1", mean_v = c("D", "M"),
          sd_v = "M", st0 = "1"),
        match.map = list(M = list(s1 = 1, s2 = 2)),
```

```
        factors   = list(S = c("s1", "s2"), D = c("d1", "d2")),
        constants = c(sd_v.false = 1, st0 = 0),
        responses = c("r1", "r2"),
        type      = "norm")

## Population distribution, rate effect on F
pop.mean <- c(A=.4, B.r1=.6, B.r2=.8, t0=.3,
  mean_v.d1.true  = 1.5,
  mean_v.d2.true  = 1.0,
  mean_v.d1.false = 0,
  mean_v.d2.false = 0,  sd_v.true = .25)
pop.scale <-c(A=.1, B.r1=.1, B.r2=.1, t0=.05,
  mean_v.d1.true  =.2,
  mean_v.d2.true  =.2,
  mean_v.d1.false =.2,
  mean_v.d2.false =.2,  sd_v.true = .1)

pop.prior <- BuildPrior(
  dists = rep("tnorm", 9),
  p1 = pop.mean,
  p2 = pop.scale,
  lower = c(0,0,0,   .1, NA,NA,NA,NA, 0),
  upper = c(NA,NA,NA, 1, NA,NA,NA,NA, NA))

dat <- simulate(model, nsub = 6, nsim = 30, prior = pop.prior)
dmi <- BuildDMI(dat, model)
p.prior <- BuildPrior(
  dists = rep("tnorm", 9),
  p1   = pop.mean,
  p2   = pop.scale*5,
  lower=c(0,0,0,   .1, NA,NA,NA,NA, 0),
  upper=c(NA,NA,NA,NA, NA,NA,NA,NA, NA)
)

mu.prior <- BuildPrior(
  dists = rep("tnorm",  9),
  p1    = pop.mean,
  p2    = c(1,   1,  1,  1,   2,  2,  2, 2,  1),
  lower = c(0,   0,  0, .01, NA, NA, NA, NA, 0),
  upper = c(NA, NA, NA,  NA, NA, NA, NA, NA, NA)
)

sigma.prior <- BuildPrior(
  dists = rep("beta", length(p.prior)),
  p1    = c(A = 1, B.r1 = 1, B.r2 = 1, t0 = 1, mean_v.d1.true = 1,
    mean_v.d2.true = 1, mean_v.d1.false = 1, mean_v.d2.false = 1,
    sd_v.true = 1),
  p2    = rep(1, 9))

pp.prior <- list(mu.prior, sigma.prior)
hsam <- StartNewHypersamples(32, dmi, pop.prior, pp.prior)
```

`sumloglike` *Calculate Summed, Log-likelihood of a Cognitive Model*

Description

The function calculates log-likelihood for every trial. The input must be a data model instance.

Usage

```
sumloglike(pVec, pnames, allpar, parnames, model, type, dim1, dim2, dim3,
  n1idx, ise, cellidx, RT, matchcell, isr1, posdrift, nsim, bw, ncore,
  gpuid, debug)
```

Arguments

`pVec`	a parameter vector
`pnames`	a string vector storing the name of a parameter vector
`allpar`	all parameters
`parnames`	parameter names
`model`	a model specification
`type`	model type
`dim1`	first dimension of a model
`dim2`	second dimension of a model
`dim3`	third dimension of a model
`n1idx`	n1 order index
`ise`	an index vector storing if a cell is empty.
`cellidx`	cell index
`RT`	a RT vector
`matchcell`	an index vector storing is the cell is a match response
`isr1`	is r1 index
`posdrift`	a Boolean switch to enforce postive drift rate correction
`nsim`	number of simulation
`bw`	bandwidth
`ncore`	number of parallel core
`gpuid`	GPU card index on a multiple GPU machine
`debug`	whether to print debugging information for assuming drift rates are drawn from a normal distribution.

Value

a double scalar

Examples

```
m1 <- BuildModel(
  p.map     = list(a = "1", v = "1", z = "1", d = "1", sz = "1", sv = "1",
              t0 = "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)

## Set up a model-data instance
dat <- simulate(m1, 128, ps = p.vector)
dmi <- BuildDMI(dat, m1)
## sumloglike(p.vector, dmi)
## [1] 0.3796048
```

`summary.model` *Summarise posterior samples*

Description

This calls severn different variants of summary function to summarise posterior samples

Usage

```
## 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`	summarise from which iteration.
`end`	summarise to the end 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 indicating 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

```
## Not run:
est1 <- summary(hsam[[1]], FALSE)
est2 <- summary(hsam[[1]], FALSE, 1, 100)
est3 <- summary_one(hsam[[1]], 1, 100, c(.025, .5, .975), verbose = TRUE)
est4 <- summary_one(hsam[[1]], 1, 100, c(.025, .5, .975), verbose = F)

est5 <- summary_many(hsam, 1, 100, c(.025, .5, .975), FALSE)
est6 <- summary_many(hsam, 1, 100, c(.025, .5, .975), TRUE)
est7 <- summary(hsam)
est8 <- summary(hsam, verbose = TRUE)
est9 <- summary(hsam, verbose = FALSE)


hest1 <- summary_hyper(hsam, 1, 100, F, F, c(.025, .5, .975), 2, F)
hest2 <- summary_hyper(hsam, 1, 100, F, F, c(.05, .5, .9), 2, F)
hest3 <- summary(hsam, TRUE)

## End(Not run)
```

---

summary_mcmc_list *Summary statistic for posterior samples*

Description

Calculate summary statistics for pMCMC 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 to a factorial response x parameter matrix. The matrix is used by likelihood functions, assigning a trial to a cell for calculating probability densities.

Usage

```
TableParameters(x, cell, model, n1order = TRUE)
```

Arguments

x	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. use length(dimnames(model)) to check the upper bound.
model	a model ibhect
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 martix in natural order, which is used by simulate. By default n1.order is TRUE, the returned matrix, used by n1PDF-like functions.

Examples

```
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.f1.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 1.23 0.11   0
## 1.15 -0.1 0.87 0.26 0 1.23 0.11   0

##    A b  t0 mean_v sd_v st0
## 0.75 1 0.2    2.5    1   0
## 0.75 1 0.2    1.5    1   0
```

`theta2mcmclist` *Convert theta to a mcmc List*

Description

Extracts the parameter array (ie theta) from posterior samples of a partiipant 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 iteraton
`split`	whether to divide one MCMC sequence into two sequences.
`subchain`	boolean swith convert only a subset of chains
`nsubchain`	indicate the number of chains in the subset
`thin`	thinning lenght of the posterior samples

Details

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

Examples

```
## 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, .50, .50, .4, .4, .4, .4)
pop.scale <- c(.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(
  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)))
sigma.prior <- BuildPrior(
  dists = rep("beta", npar),
  p1    = rep(1, npar),
  p2    = rep(1, npar),
  upper = rep(2, npar))
names(sigma.prior) <- GetPNames(model)
pp.prior <- list(mu.prior, sigma.prior)

dat <- simulate(model, nsim = 30, nsub = 10, p.prior = pop.prior)
dmi <- BuildDMI(dat, model)
ps <- attr(dat, "parameters")
```

```
hsam <- run(StartNewHypersamples(1e2, dmi, p.prior, pp.prior, 1),
  pm = .1, hpm = .1, report = 20)

tmp1 <- theta2mcmclist(hsam[[1]])
tmp2 <- theta2mcmclist(hsam[[2]], start = 10, end = 90)
tmp3 <- theta2mcmclist(hsam[[3]], split = TRUE)
tmp4 <- theta2mcmclist(hsam[[4]], subchain = TRUE)
tmp5 <- theta2mcmclist(hsam[[5]], subchain = TRUE, nsubchain = 4)
tmp6 <- theta2mcmclist(hsam[[6]], thin = 2)

## End(Not run)
```

`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