Package 'DstarM'

August 29, 2016

Type Package

Title Analyze Two Choice Reaction Time Data with the D*M Method

Version 0.1.0

Author Don van den Bergh, Stijn Verdonck, Francis Tuerlinckx

Maintainer Don van den Bergh <donvdbergh@hotmail.com>

Description A collection of functions to estimate parameters of a diffusion model via a D*M analysis. Build in models are: the Ratcliff diffusion model, the RWiener diffusion model, and Linear Ballistic Accumulator models. Custom models functions can be specified as long as they have a density function.

License GPL (>= 2)

LazyData TRUE

Imports DEoptim, RWiener, rtdists, methods, stats, ggplot2

ByteCompile TRUE

RoxygenNote 5.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2016-08-29 08:18:32

R topics documented:

`byParticipant` *Helper function to do DstarM analyses by participant.*

Description

A helper function to do run DstarM functions per participant. Supported functions are: `estDstarM`, `estND`, and `estObserved`.

Usage

```
byParticipant(data, resD, resND, argsEstDstarM = list(), argsEstND = list(),
  argsEstObserved = list(), uniqueArgs = FALSE)
```

Arguments

`data`	data.frame with a $id column. For other requirements, see `estDstarM`.
`resD`	list of decision models (all of class DstarM).
`resND`	list of nondecision models (all of class DstarM).
`argsEstDstarM`	Additional arguments for `estDstarM`.
`argsEstND`	Additional arguments for `estND`.
`argsEstObserved`	Additional arguments for `estObserved`.
`uniqueArgs`	Logical, are the additional arguments specified using one list for all participants or does the list of additional arguments contain lists with participant specific arguments?

Examples

```
## Not run:
library(DstarM)
tt = seq(0, 5, .1)
pars = c(.8, 2, .5, .5, .5, # condition 1
         .8, 3, .5, .5, .5, # condition 2
         .8, 4, .5, .5, .5) # condition 3
pdfND = dbeta(tt, 10, 30)
# simulate data for 2 participants
n = 1.2e3
data = rbind(simData(n = n, pars = pars, tt = tt, pdfND = pdfND),
             simData(n = n, pars = pars, tt = tt, pdfND = pdfND))
# add participant column
```

```
data$pp = rep(1:2, each = n)
# define restriction matrix
restr = matrix(1:5, 5, 3)
restr[2, -1] = 6:7 # allow drift rates to differ
# fix variance parameters
fixed = matrix(c('sz1', .5, 'sv1', .5), 2, 2)
## Run D*M analysis by pp
resDpp = byParticipant(data = data,
                       argsEstDstarM = list(tt = tt,
                                            restr = restr,
                                            fixed = fixed,
                                            Optim = list(parallelType = 1)))
coef(resDpp)
plot(resDpp)
resNDpp = byParticipant(resD = resDpp,
                        argsEstND = list(Optim = list(parallelType = 1)))
plot(resNDpp, xlim = 0:1)
lines(tt, pdfND, col = 3, type = 'b')
resObsPp = byParticipant(resD = resDpp,
                         resND = resNDpp)
plot(resObsPp)

## End(Not run)
```

`calcIC` *Calculate Information Criteria*

Description

Calculate information criteria for `D*M` models. This function can be called with the resObserved argument or the resDecision and, in case of D*M analyses, also the nondecision analysis. If resObserved is not supplied a call to `estObserved` will be made.

Usage

```
calcIC(resObserved, resDecision, resND, data, npar)
```

Arguments

`resObserved`	an object of class `D*M`, specifically output of the function `estObserved`.
`resDecision`	an object of class `D*M`, specifically output of the function `estDstarM`.
`resND`	an object of class `D*M`, specifically output of the function `estND`.
`data`	The data used to estimate `resDecision`.
`npar`	The number of parameters estimated.

Details

Calculates several information criteria.

Calculate information criteria for D*M models. The results should be interpreted carefully! D*M models are **not** estimated by maximizing a likelihood, and therefore more complex models do not always necessarily have a higher likelihood than less complex models. Also, adding parameters to the decision model might improve the fit of the decision model, but can worsen the fit of the nondecision model. As a result the total fit decreases when adding unneeded parameters. This is shown in the example. Caution is adviced when interpreting likelihood based information criteria.

Value

a list (S3 object of class 'D*M') that contains:

`AIC`	Akaike Information Criterion.
`AICc`	AIC with a correction for finite sample sizes.
`BIC`	Bayesian Information Criterion, also known as Schwarz criterion.
`npar`	Number of parameters.
`logLik`	The natural log of the likelihood.

Examples

```
rm(list = ls())
set.seed(42)
# Simulate data, fit 3 models, compare models.
# simulate data with three stimuli of different difficulty.
# this implies different drift rates across conditions.
# define a time grid. A more reasonable stepsize is .01; this is just for speed.
tt = seq(0, 5, .1)
pars = c(.8, 2, .5, .5, .5, # condition 1
        .8, 3, .5, .5, .5, # condition 2
        .8, 4, .5, .5, .5) # condition 3
pdfND = dbeta(tt, 10, 30)
# simulate data
dat = simData(n = 3e3, pars = pars, tt = tt, pdfND = pdfND)
# define restriction matrices
restr1 = restr2 = restr3 = matrix(1:5, 5, 3)
## restr1 allows nothing to differ over conditions
restr2[2, 2:3] = 6:7 # allow drift rates to differ
restr3[1:3, 2:3] = 6:11 # allow all decision model parameters to differ
# fix variance parameters
fixed = matrix(c('sz1', .5, 'sv1', .5), 2, 2)
## Not run:
# Run D*M analysis
resD1 = estDstarM(dat = dat, tt = tt, restr = restr1, fixed = fixed, Optim = list(parallelType = 1))
resD2 = estDstarM(dat = dat, tt = tt, restr = restr2, fixed = fixed, Optim = list(parallelType = 1))
resD3 = estDstarM(dat = dat, tt = tt, restr = restr3, fixed = fixed, Optim = list(parallelType = 1))
# Estimate nondecision density - The warnings can be ignored.
# Quantiles for nondecision densities cannot be calculated due to the big time grid (by .1).
resND1 = estND(resD1, Optim = list(parallelType = 1))
```

```
resND2 = estND(resD2, Optim = list(parallelType = 1))
resND3 = estND(resD3, Optim = list(parallelType = 1))
# Estimate observed densities
resObs1 = estObserved(resD1, resND1)
resObs2 = estObserved(resD2, resND2)
resObs3 = estObserved(resD3, resND3)
# Compare optimizer fitness
lstObs = list(resObs1, resObs2, resObs3)
# fit$total is the sum of fit$Decision and
# fit$ND. Model 3 has the lowest fit
sapply(lstObs, function(x) x$fit$total)
# model 3 has lowest Decision fit due to overfitting
sapply(lstObs, function(x) x$fit$Decision)
# However, model 3 has worse nondecision fit compared to model 2 due to overfitting!
sapply(lstObs, function(x) x$fit$ND)
# Calculate information criteria
IC1 = calcIC(resObs1, resD1, data = dat)
IC2 = calcIC(resObs2, resD2, data = dat)
IC3 = calcIC(resObs3, resD3, data = dat)
sapply(list(IC1, IC2, IC3), function(x) x$logLik)
# Do likelihood ratio tests
anova(IC1, IC2, IC3) # unorthodox output, model 2 is best?
which.min(c(IC1$AIC, IC2$AIC, IC3$AIC)) # model 2 is best
which.min(c(IC1$BIC, IC2$BIC, IC3$BIC)) # model 2 is best
# We can do the same for traditional analyses
# likelihood based methods do work for traditional analyses
# define restriction matrices
restrC1 = restrC2 = restrC3 = matrix(1:7, 7, 3)
# restr1 allows nothing to differ over conditions
restrC2[2, 2:3] = 8:9 # allow drift rates to differ
restrC3[c(1:2, 4), 2:3] = 8:13 # allow all decision model parameters to differ
# Do traditional analyses
resC1 = estDstarM(dat = dat, tt = tt, restr = restrC1, fixed = fixed,
DstarM = FALSE, Optim = list(parallelType = 1))
resC2 = estDstarM(dat = dat, tt = tt, restr = restrC2, fixed = fixed,
DstarM = FALSE, Optim = list(parallelType = 1))
resC3 = estDstarM(dat = dat, tt = tt, restr = restrC3, fixed = fixed,
DstarM = FALSE, Optim = list(parallelType = 1))
# resObs does not need to be calculated now since resC* contains the full model
# Estimate observed densities
resObsC1 = estObserved(resC1)
resObsC2 = estObserved(resC2)
resObsC3 = estObserved(resC3)
# Compare optimizer fitness
lstObsC = list(resObsC1, resObsC2, resObsC3)
sapply(lstObsC, function(x) x$fit$total)
# Calculate information criteria
ICC1 = calcIC(resObserved = resObsC1, data = dat)
ICC2 = calcIC(resDecision = resC2, data = dat) # both input methods are possible
ICC3 = calcIC(resObserved = resObsC3, data = dat)
sapply(list(ICC1, ICC2, ICC3), function(x) x$logLik)
# Do likelihood ratio tests
# correct model is retrieved by all methods
```

```
anova(ICC1, ICC2, ICC3)
which.min(c(ICC1$AIC, ICC2$AIC, ICC3$AIC))
which.min(c(ICC1$BIC, ICC2$BIC, ICC3$BIC))

## End(Not run)
```

---

chisq *Calculates the distance between two probability densities.*

Description

Calculates the distance between two probability densities.

Usage

```
chisq(tt, a, b)

battacharyya(tt, a, b)

hellinger(tt, a, b)
```

Arguments

tt	the time grid on which the densities are evaluated.
a	a vector with values of the first density.
b	a vector with values of the second density.

Value

The distance between densities a and b.

Examples

```
# Lets simulate a bunch of parameters and compare the three distance measures.

tt = seq(0, 5, .001)
parsMatV = cbind(.8, seq(0, 5, .5), .5, .5, .5) # differ only in drift speed
parsMatA = cbind(seq(.5, 2, .15), 2, .5, .5, .5)# differ only in boundary
# calculate densities for all these parameters
dV = apply(parsMatV, 1, function(x, tt) Voss.density(tt, x, boundary = 'upper'), tt = tt)
dA = apply(parsMatA, 1, function(x, tt) Voss.density(tt, x, boundary = 'upper'), tt = tt)
# make plots of the densities
matplot(tt, dA, xlim = c(0, .6), main = 'Densities with different Boundary',
        col = rainbow(ncol(dA)),type = 'l', lty = 1, las = 1, bty = 'n',
        xlab = 'Time', ylab = 'Density')
legend('topright', lty = 1, bty = 'n', col = rainbow(ncol(dA)),
       legend = paste('a = ', parsMatA[, 1]))
matplot(tt, dV, xlim = c(0, .6), main = 'Densities with different Drift Speed',
        col = rainbow(ncol(dV)), type = 'l', lty = 1, las = 1, bty = 'n',
```

```
        xlab = 'Time', ylab = 'Density')
legend('topright', lty = 1, bty = 'n', col = rainbow(ncol(dV)),
       legend = paste('v = ',parsMatV[, 2]))
# empty matrices for data storage
distMatV = matrix(NA, nrow = ncol(dV) - 1, ncol = 3,
                  dimnames = list(NULL, c('Chisq', 'Bhattacharyya', 'Hellinger')))
distMatA = matrix(NA, nrow = ncol(dA) - 1, ncol = 3,
                  dimnames = list(NULL, c('Chisq', 'Bhattacharyya', 'Hellinger')))
# calculate distances between densities in column i and i + 1.
# this is done using three different distance measures
for (i in 1:(ncol(dA) - 1)) {
  distMatV[i, ] = c(chisq(tt, dV[, i], dV[, i + 1]),
                    battacharyya(tt, dV[, i], dV[, i + 1]),
                    hellinger(tt, dV[, i], dV[, i + 1]))
  distMatA[i, ] = c(chisq(tt, dA[, i], dA[, i + 1]),
                    battacharyya(tt, dA[, i], dA[, i + 1]),
                    hellinger(tt, dA[, i], dA[, i + 1]))
}
# The three distance measures correlate highly for differences in Boundary
cor(distMatA)
# The battacharyya distance measures does not correlate with the others
# when calculating differences in drift speed
cor(distMatV)
```

`estCdf` *Estimate cumulative distribution for D*M models*

Description

Estimate cumulative distribution for D*M models

Usage

```
estCdf(x)
```

Arguments

`x`	Any density function to calculate a cumulative distribution for. The code is designed for input of class `DstarM` but other input is also accepted. Other input can be either a matrix where columns represent densities or a single vector representing a density.

Details

Cumulative distributions functions are calculated by: `cumsum(x) / sum(x)`. This method works well enough for our purposes. The example below shows that the `ecdf` functions seems to work slightly better. However, this estimates a cdf from raw data and does not transform a pdf into a cdf and is therefore not useful for D*M models.

Value

Cumulative density function(s). If the input was a matrix, a matrix of cumulative density functions is returned.

Examples

```
x = rnorm(1000)
xx = seq(-5, 5, .1)
approx1 = stats::ecdf(x)(xx)
approx2 = estCdf(dnorm(xx, mean(x), sd(x)))
trueCdf = pnorm(xx)
matplot(xx, cbind(trueCdf, approx1, approx2), type = c('l', 'p', 'p'),
        lty = 1, col = 1:3, pch = 1, bty = 'n', las = 1, ylab = 'Prob')
legend('topleft', legend = c('True Cdf', 'Stats Estatimation', 'DstarM Estimation'),
       col = 1:3, lty = c(1, NA, NA), pch = c(NA, 1, 1), bty = 'n')
```

---

estDstarM *Do a D*M analysis*

Description

Do a D*M analysis

Usage

```
estDstarM(data, tt, restr = NULL, fixed = list(), lower, upper,
  Optim = list(), DstarM = TRUE, SE = 0, oscPdf = TRUE,
  splits = rep.int(0, (ncondition)), forceRestriction = TRUE, mg = NULL,
  h = 1, pars, fun.density = Voss.density, args.density = list(),
  fun.dist = chisq, args.dist = list(tt = tt), verbose = TRUE)
```

Arguments

`data`	A dataframe with: a column named `rt` containing response times in ms, a column named `response` containing at most 2 response options, and an optional column named `condition` containing a numeric index as to which conditions observations belong.
`tt`	A time grid on which the density function will be evaluated. Should be larger than the highest observed reaction time.
`restr`	A restriction matrix where each column depicts one condition. The number of rows should match the number of parameters (and be equal to the length of lower). The contents of `restr` should be numbers, identical numbers means that these parameters (either within or between condition) will be constrained. Different numbers means parameters will not be constrained.
`fixed`	A matrix that allows for fixing parameters to certain values.

lower	Should be a vector containing lower bounds for each parameter. Has a default if fun.density == Voss.density.
upper	Should be a vector containing upper bounds for each parameter. Has a default if fun.density == Voss.density.
Optim	a named list with identical arguments to DEoptim.control. In addition, if verbose == TRUE Optim$steptol can be a vector, i.e. c(200, 50, 10) means: Do 200 iterations then check for convergence, do 50 iterations then check for convergence, check every 10 iterations for convergence until itermax is reached. Defaults to Optim = list(reltol = 1e-6, itermax = 1e3,steptol = 50, CR = .9, trace
DstarM	If TRUE a D*M analysis is done, otherwise the Chi square distance between data and model is minimized.
SE	positive value, how many standard error to add to the variance to relax the variance restriction a bit.
oscPdf	Logical, if TRUE check for oscillations in calculated densities and remove densities with oscilations.
splits	Numeric vector determining which conditions have an equal nondecision density. Identical values in two positions indicate that the conditions corresponding to the indices of those values have an identical nondecision distribution.
forceRestriction	if TRUE the variance restriction is enforced.
mg	Supply a data density, usefull if a uniform kernel approximation does not suffice.
h	bandwidth of a uniform kernel used to generate data based densities.
pars	Optional parameter vector to supply if one wishes to evaluate the objective function in a given parameter vector. Only used if itermax equal zero.
fun.density	Function used to calculate densities. See details.
args.density	A names list containing additional arguments to be send to fun.density.
fun.dist	Function used to calculate distances between densities. Defaults to a chi-square distance.
args.dist	A named list containing additional arguments to be send to fun.dist.
verbose	Logical, should intermediate output be printed? Defaults to TRUE. Estimation will speed up if set to FALSE. If set to TRUE, Optim$trace will be forced to 0, hereby disabling the build in printing of DEoptim. To enable the printing of DEoptim, set verbose to FALSE and specify trace in Optim.

Details

Response options will be alphabetically sorted and the first response option will be treated as the 'lower' option. This means that if the observed proportion of the first response options is higher, the drift speed will most likely be negative.

fun.density allows a user to specify a custom density function. This function must (at least) take the following arguments: t: a vector specifying at which time points to calculate the density pars: a parameter vector boundary: character 'upper' or 'lower' specifying for which response option the density will be calculated. DstarM: Logical, if TRUE the density should not describe the nondecision density, if FALSE it should describe the nondecision density. Any additional arguments

can be passed to `fun.density` via the argument `args.density`. If one intends to use a custom density function it is recommended to test the function first with `testFun`. When specifying a custom density function it is probably also necessary to change the lower and upper bounds of the parameter space.

Value

Returns an object of class `'D*M'`, which is a named list.

Examples

```
# simulate data with three stimuli of different difficulty.
# this implies different drift rates across conditions.
# define a time grid. A more reasonable stepsize is .01; this is just for speed.
tt = seq(0, 5, .1)
pars = c(.8, 2, .5, .5, .5, # condition 1
        .8, 3, .5, .5, .5,  # condition 2
        .8, 4, .5, .5, .5)  # condition 3
pdfND = dbeta(tt, 10, 30)
# simulate data
data = simData(n = 3e3, pars = pars, tt = tt, pdfND = pdfND)
# define restriction matrix
restr = matrix(1:5, 5, 3)
restr[2, 2:3] = 6:7 # allow drift rates to differ
# fix variance parameters
fixed = matrix(c('sz1', .5, 'sv1', .5), 2, 2)
## Not run:
# Run D*M analysis
res = estDstarM(data = data, tt = tt, restr = restr, fixed = fixed)
coef(res)
summary(res)

## End(Not run)
```

`estND` *Estimate nondecision denstiy*

Description

Estimate nondecision denstiy

Usage

```
estND(res, tt = NULL, data = NULL, h = res$h, zp = 5, upper.bound = 1,
  lower.bound = 0, Optim = list(), verbose = TRUE, dist = NULL, NDindex,
  max = 100)
```

Arguments

res	an object of class D*M.
tt	optional timegrid if the nondecision density is to be esimtated at a different grid than the model density.
data	if tt is specified then the original dataset must be supplied too.
h	Optional smoothing parameter to be used when estimating the nondecision model on a different time grid than the decision model. If omitted, the smoothing parameter of the decision model is used.
zp	Zero padding the estimated nondecision density by this amount to avoid numerical artefacts.
upper.bound	An upper bound for the nondecision density. Defaults to one. Lowering this bound can increase estimation speed, at the cost of assuming that the density of the nondecision distribution is zero past this value.
lower.bound	A lower bound for the nondecision density. Defaults to zero. Increasing this bound can increase estimation speed, at the cost of assuming that the density of the nondecision distribution is zero past this value.
Optim	a named list with identical arguments to DEoptim.control. In addition, if verbose == TRUE Optim$steptol can be a vector, i.e. c(200, 50, 10) means: Do 200 iterations then check for convergence, do 50 iterations then check for convergence, check every 10 iterations for convergence until itermax is reached. If there are multiple nondecision distributions to estimate, one can supply different estimation parameters for every nondecision distribution by supplying Optim as a list of lists. Every sublists then corresponds to parameters for one nondecision distribution and should consist of arguments for DEoptim.control. Defaults to Optim = list(reltol = 1e-6, itermax = 1e4, steptol = 200, CR = .9, trace = 0).
verbose	Logical, should the function return text output on the current status of the estimation procedure. As the estimation can take 30+ minutes this defaults to yes. If FALSE, the analysis will be somewhat faster.
dist	A matrix where columns represent nondecision distributions. If this argument is supplied then the objective function will be evaluated in these values.
NDindex	A vector containing indices of which nondecision distributions to estimate. If omitted, all nondecision distributions that complement the results in res are estimated.
max	A positive float which indicates the maximum height of the nondecision distribution. If estimated nondecision distributions appear chopped of or have a lot of values at this max value it is recommended to re-estimate the nondecision distributions with a higher max value. Increasing the max value without reason will increase the size of the parameter space and slow the estimation procedure.

Examples

```
# simulate data with three stimuli of different difficulty.
# this implies different drift rates across conditions.
# define a time grid. A more reasonable stepsize is .01; this is just for speed.
tt = seq(0, 5, .1)
```

```
pars = c(.8, 2, .5, .5, .5, # condition 1
         .8, 3, .5, .5, .5, # condition 2
         .8, 4, .5, .5, .5) # condition 3
pdfND = dbeta(tt, 10, 30)
# simulate data
dat = simData(n = 3e5, pars = pars, tt = tt, pdfND = pdfND)
# define restriction matrix
restr = matrix(1:5, 5, 3)
restr[2, 2:3] = 6:7 # allow drift rates to differ
# fix variance parameters
fixed = matrix(c('sz1', .5, 'sv1', .5), 2, 2)
## Not run:
# Run D*M analysis
res = estDstarM(data = dat, tt = tt, restr = restr, fixed = fixed)
# Estimate nondecision density
resND = estND(res)
plot(resND)
lines(tt, pdfND, type = 'b', col = 2)

## End(Not run)
```

`estObserved` *Estimate observed data density*

Description

Estimates the density of the observed data by convoluting the estimated decision distributions with the estimated nondecision distributions. If a traditional analysis was run the argument resND can be omitted.

Usage

```
estObserved(resDecision, resND)
```

Arguments

`resDecision`	output of `estDstarM`.
`resND`	output of `estND`.

Value

a list (S3 object of class 'DstarM') that contains:

`obsNorm`	A matrix containing normalized densities of each condition response pair.
`obs`	A matrix containing unnormalized densities of each condition response pair.
`tt`	The time grid used.
`fit`	A list containing the values of the objective function for the total model ($total), for the decision model ($Decision) and for the nondecision distribution(s) ($ND).

npar	The number of parameters used in the decision model.
obsIdx	A numeric vector containing indices of any not observed condition-response pairs.

Examples

```
# simulate data with three stimuli of different difficulty.
# this implies different drift rates across conditions.
# define a time grid. A more reasonable stepsize is .01; this is just for speed.
tt = seq(0, 5, .1)
pars = c(.8, 2, .5, .5, .5, # condition 1
        .8, 3, .5, .5, .5, # condition 2
        .8, 4, .5, .5, .5) # condition 3
pdfND = dbeta(tt, 10, 30)
# simulate data
lst = simData(n = 3e5, pars = pars, tt = tt, pdfND = pdfND, return.pdf = TRUE)
dat = lst$dat
# define restriction matrix
restr = matrix(1:5, 5, 3)
restr[2, 2:3] = 6:7 # allow drift rates to differ
# fix variance parameters
fixed = matrix(c('sz1', .5, 'sv1', .5), 2, 2)
## Not run:
# Run D*M analysis
resD = estDstarM(dat = dat, tt = tt, restr = restr, fixed = fixed)
# Estimate nondecision density
resND = estND(resD)
# Estimate observed density
resObs = estObserved(resD, resND)
# plot histograms with overlayed
# densities per condition-response pair
plotObserved(resObserved = resObs, data = dat,
            xlim = c(0, 1))
# plot estimated and true densities
plot(resObs, col = rep(1:3, each = 2), xlim = 0:1)
matlines(tt, lst$pdfNormalized, col = rep(1:3, each = 2), lty = 2)

## End(Not run)
```

---

estQdf *Estimate quantiles of distribution*

Description

Estimate quantiles of distribution

Usage

```
estQdf(p, x, cdf)
```

Arguments

p	A vector of probabilities.
x	The x-axis values corresponding to the cumulative distribution function.
cdf	A cumulative distributions function, i.e. output of estCdf.

Details

Quantiles are obtained in the following manner. For p = 0 and p = 1, the minimum and maximum of x is used. For other probabilities the quantiles are obtained via `q[i] = uniroot(x, cdf - p[i])$root`. Y values are interpolated via approxfun.

Value

Quantiles of cumulative distribution function(s). If the input was a matrix of cumulative distributions functions, a matrix of quantiles is returned.

Examples

```
x = seq(-9, 9, .1) # x-grid
d = dnorm(x) # density functions
p = seq(0, 1, .2) # probabilities of interest
cEst = estCdf(d) # estimate cumulative distribution functions
qEst = estQdf(p = p, x = x, cdf = cEst) # estimate quantiles
plot(x, cEst, bty = 'n', las = 1, type = 'l', ylab = 'Probability') # plot cdf
abline(h = p, v = qEst, col = 1:6, lty = 2) # add lines for p and for obtained quantiles
points(x = qEst, y = p, pch = 18, col = 1:6, cex = 1.75) # add points for intersections
```

---

getSter *Estimate variance of nondecision density*

Description

Estimate variance of nondecision density

Usage

```
getSter(res)
```

Arguments

res	An object of class D*M.

Details

The object res can either be output from estDstarM or output from estND. if the former is supplied, getSter attempts to calculate the variance of the nondecision distribution by subtracting the variance of the model distribution from the variance of the data distribution. If the latter is supplied, the variance is calculated by integrating the nondecision distribution.

getTer *Calculate Mean of the nondecision distribution.*

Description

Calculate Mean of the nondecision distribution.

Usage

```
getTer(res, dat)
```

Arguments

res	An object of class D*M.
dat	The data object used to create res.

Details

The object res can either be output from estDstarM or output from estND. If the former is supplied it is also necessary to supply the data used for the estimation. The mean will then be estimated by substracting the mean of the model densities from the mean of the data density. If the latter is supplied than this is not required; the mean will be calculated by integrating the nondecision distribution.

Value

A vector containing estimates for the mean of the nondecision densities.

plotObserved *Plot histogram of data with overlayed model predicted density.*

Description

Plots histograms for each condition-response pair/ condition/ response with overlayed estimated densities.

Usage

```
plotObserved(resObserved, data, what = "cr", layout = NULL, main = NULL,
  linesArgs = list(), ggplot = FALSE, prob = NULL, probType = 3, ...)
```

Arguments

resObserved	output of estObserved.
data	The dataset used to estimate the model.
what	What to plot. Can be 'cr' for 'condition-response pairs, 'c' for condition, and 'r' for response.
layout	An optional layout matrix.
main	an optional vector containing names for each plot.
linesArgs	A list containing named arguments to be passed to lines.
ggplot	Logical, should ggplot2 be used instead of base R graphics? If set to TRUE, some arguments from linesArgs and ... will be ignored (but can be added to plots manually).
prob	Should a qqplot of observed vs model implied quantiles be plotted? If NULL (the default) then a histogram overlayed with model implied densities will be plotted. Otherwise, this argument should be a vector of probabilities to be passed to estQdf.
probType	A numeric value defining several plotting options. 0 does nothing, 1 removes the 0% quantile, 2 removes the 100% quantile and 3 removes both the 0% and 100% quantile.
...	Further arguments to be passed to hist.

Details

Keep in mind when using what = 'c' or what = 'r' pdfs are simply averaged, not weighted to the number of observed responses.

Value

if ggplot is FALSE invisible(), otherwise a list

Examples

```
# simulate data with three stimuli of different difficulty.
# this implies different drift rates across conditions.
# define a time grid. A more reasonable stepsize is .01; this is just for speed.
tt = seq(0, 5, .1)
pars = c(.8, 2, .5, .5, .5, # condition 1
         .8, 3, .5, .5, .5, # condition 2
         .8, 4, .5, .5, .5) # condition 3
pdfND = dbeta(tt, 10, 30)
# simulate data
lst = simData(n = 3e5, pars = pars, tt = tt, pdfND = pdfND, return.pdf = TRUE)
dat = lst$dat
# define restriction matrix
restr = matrix(1:5, 5, 3)
restr[2, 2:3] = 6:7 # allow drift rates to differ
# fix variance parameters
fixed = matrix(c('sz1', .5, 'sv1', .5), 2, 2)
```

```
## Not run:
# Run D*M analysis
resD = estDstarM(dat = dat, tt = tt, restr = restr, fixed = fixed)
# Estimate nondecision density
resND = estND(resD)
# Estimate observed density
resObs = estObserved(resD, resND)
# plot histograms with overlayed
# densities per condition-response pair
plotObserved(resObserved = resObs, data = dat,
            xlim = c(0, 1))
# plot estimated and true densities
plot(resObs, col = rep(1:3, each = 2), xlim = 0:1)
matlines(tt, lst$pdfNormalized, col = rep(1:3, each = 2), lty = 2)
# other uses of plotObserved
plotObserved(resObserved = resObs, data = dat, what = 'cr', xlim = c(0, 1))
plotObserved(resObserved = resObs, data = dat, what = 'c', xlim = c(0, 1))
plotObserved(resObserved = resObs, data = dat, what = 'r', xlim = c(0, 1))

## End(Not run)
```

`rtDescriptives` *Descriptives of reaction time data*

Description

Descriptives of reaction time data

Usage

```
rtDescriptives(data)
```

Arguments

`data`	A reaction time dataset. Must be a dataframe with $rt, $condition and $response.

Details

This function and `rtHist` are helper functions to inspect raw data.

Value

An object of class 'D*M', containing raw counts and proportions for condition response pairs, conditions, and responses.

Examples

```
tt = seq(0, 5, .01)
dat = simData(n = 3e5, pars = rep(.5, 5), tt = tt, pdfND = dbeta(tt, 10, 30))
x = rtDescriptives(dat)
x
print(x, what = 'cr')
print(x, what = 'c')
print(x, what = 'r')
print(x, digits = 20)
```

`rtHist` *Make histograms of reaction time data*

Description

Make histograms of reaction time data

Usage

```
rtHist(data, what = "cr", layout = NULL, nms = NULL, ggplot = FALSE,
  ...)
```

Arguments

`data`	A reaction time dataset. Must be a dataframe with $rt, $condition and $response.
`what`	@param what What to plot. Can be 'cr' for 'condition-response pairs, 'c' for condition, and 'r' for response.
`layout`	An optional layout.
`nms`	An optional vector of names for each plot. If omitted the names will be based on the contents of `data$condition` and/or `data$response`.
`ggplot`	ggplot Logical, should ggplot2 be used instead of base R graphics? If set to TRUE, some arguments from `linesArgs` and `...` will be ignored (but can be added to plots manually).
`...`	Arguments to be passed to `hist`

Details

This function and `rtDescriptives` are helper functions to inspect raw data.

Value

invisible()

Examples

```
tt = seq(0, 5, .01)
dat = simData(n = 3e4, pars = rep(.5, 5), tt = tt, pdfND = dbeta(tt, 10, 30))
rtHist(dat, breaks = tt, xlim = c(0, 1))
```

simData *Simulate data from a given densitiy function via multinomial sampling*

Description

Simulate data from a given densitiy function via multinomial sampling

Usage

```
simData(n, pars, tt, pdfND, fun.density = Voss.density,
  args.density = list(prec = 3), npars = 5, return.pdf = FALSE)
```

Arguments

n	Number of observations to be sampled
pars	Parameter values for the density function to be evaluated with. `length(pars)` must be a multiple of npars.
tt	time grid on which the density function will be evaluated. Responses not in this time grid cannot appear.
pdfND	either a vector of length tt specifying the nondecision density for all condition-response pairs, or a matrix where columns corresponds to the nondecision densities of condition-response pairs.
fun.density	Density function to use.
args.density	Additional arguments to be passed to `fun.density`, aside from `tt`, `pars`, and a boundary argument ('upper' or 'lower')
npars	Number of parameters `fun.density` must be evaluated with. If `length(pars) > npars` each `npars` values in `pars` will be seen as the parameter values of a condition.
return.pdf	Logical, if TRUE `genData` returns a list containing the probability density function used and the data, if FALSE `genData` returns a dataframe with simulated data.

Details

Simulate data via multinomial sampling. The response options to sample from should be provided in `tt`. The number of conditions is defined as `length(pars) / npars`.

Value

A sorted dataframe where rows represent trials. It contains: a column named rt containing reaction times in seconds, a column named response containing either reponse option lower or upper, and a column named condition indicating which condition a trials belongs to. If `return.pdf` is TRUE it returns a list where the first element is the sorted dataframe, the second through the fifth elements are lists that contain densities used for simulating data.

Examples

```
tt = seq(0, 5, .01)
pdfND = dbeta(tt, 10, 30)
n = 100
pars = c(1, 2, .5, .5, .5)
dat = simData(n, pars, tt, pdfND)
head(dat)
```

`testFun` *Test fun.density with lower and upper bounds*

Description

Test fun.density with lower and upper bounds

Usage

```
testFun(fun.density, lower, upper, args = list())
```

Arguments

`fun.density`	A density function to be evaluated.
`lower`	Lower bounds of the parameter space with which `fun.density` can be evaluated.
`upper`	Upper bounds of the parameter space with which `fun.density` can be evaluated.
`args`	Additional arguments for fun.density.

Details

A function that is called whenever a nondefault density function is passed to `DstarM`. It does some rough error checking.

Value

Returns TRUE if no errors occured, otherwise returns an error message

Examples

```
lower = c(.5, -6, .1, 0, 0)
upper = c(2, 6, .99, .99, 10)
args = list(t = seq(0, 5, .01), pars = lower, boundary = 'lower',
DstarM = TRUE)
testFun(fun.density = Voss.density, lower = lower, upper = upper,
args = args)
# TRUE
```

`Voss.density` *Calculate model density for a given set of parameters*

Description

Calculate model density for a given set of parameters

Usage

```
Voss.density(t, pars, boundary, DstarM = TRUE, prec = 3)

LBA.density(t, pars, boundary, DstarM = TRUE, ...)

Wiener.density(t, pars, boundary, DstarM)
```

Arguments

`t`	Time grid for density to be calculated on.
`pars`	Parameter vector where (if `DstarM == TRUE`) the first index contains the boundary parameter, the second contains the drift speed, the third contains the relative starting point, the fourth contains a proportion of the maximum size of the variance on the relative starting point, the fifth contains the standard deviation of the drift speed. if `DstarM == FALSE` then third index of `pars` contains the Ter, the fifth the drift speed, the the sixth contains a proportion of the maximum size of the variance on the relative starting point, the fifth contains the standard deviation of the drift speed, and the seventh contains a proportion of the maximum variance of the Ter.
`boundary`	For which response option will the density be calculated? Either 'upper' or 'lower'.
`DstarM`	Logical, see `pars`.
`prec`	Precision with which the density is calculated. Corresponds roughly to the number of decimals accurately calculated.
`...`	Other arguments, see `dLBA`

Details

These functions are examples of what `fun.density` should look like. `Voss.density` is an adaptation of `ddiffusion`, `LBA.density` is an adaptation of `dLBA`, and `wiener.density` is an adaptation of `dwiener`. To improve speed one can remove error handling. Normally error handling is useful, however because differential evolution can result in an incredible number of function evaluations (more than 10.000) it is recommended to omit error handling in custom density functions. `estDstarM` will apply some internal error checks (see `testFun`) on the density functions before starting differential evolution. A version of `ddifusion` without error handling can be found in the source code (commented out to pass R check).

Value

A numeric vector of length `length(t)` containing a density.

Examples

```
t = seq(0, .75, .01)
V.pars = c(1, 2, .5, .5, .5)
L.pars = c(1, .5, 2, 1, 1, 1)
W.pars = V.pars[1:3]
V1 = Voss.density(t = t, pars = V.pars, boundary = 'upper', DstarM = TRUE)
V2 = Voss.density(t = t, pars = V.pars, boundary = 'lower', DstarM = TRUE)
L1 = LBA.density(t = t, pars = L.pars, boundary = 'upper', DstarM = TRUE)
L2 = LBA.density(t = t, pars = L.pars, boundary = 'lower', DstarM = TRUE)
W1 = Wiener.density(t = t, pars = W.pars, boundary = 'upper', DstarM = TRUE)
W2 = Wiener.density(t = t, pars = W.pars, boundary = 'lower', DstarM = TRUE)
densities = cbind(V1, V2, L1, L2, W1, W2)
matplot(t, densities, type = 'b', ylab = 'Density', lty = 1, las = 1, bty = 'n',
        col = rep(1:3, each = 2), pch = c(0, 15, 1, 16, 2, 17), cex = .8,
        main = 'Model densities')
legend('topright', legend = c('Voss', 'LBA', 'RWiener'), lty = 1,
       pch = 15:17, col = 1:3, bty = 'n')
```

Index