Package ‘DstarM’

May 18, 2018

Type Package
Title Analyze Two Choice Reaction Time Data with the D*M Method
Version 0.3.0
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, stats, ggplot2, Rcpp
LinkingTo Rcpp, RcppArmadillo
NeedsCompilation yes
ByteCompile TRUE
RoxygenNote 6.0.1
URL https://github.com/vandenman/DstarM
BugReports https://github.com/vandenman/DstarM/issues
Suggests testthat
Repository CRAN
Date/Publication 2018-05-18 09:28:10 UTC

R topics documented:

chisq .......................................................... 2
chisqFit ....................................................... 3
estCdf .......................................................... 5
estDstarM ...................................................... 6
estND ........................................................... 9
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

# Let's simulate a bunch of parameters and compare the three distance measures

```r
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密度(tt, x, boundary = 'upper'), tt = tt)
da = apply(parsMatA, 1, function(x, tt) Voss密度(tt, x, boundary = 'upper'), tt = tt)

# make plots of the densities
matplot(tt, dV, xlim = c(0, 5), main = 'Densities with different Boundary',
        col = rainbow(ncol(dV)), type = 'l', las = 1, bty = 'n',
        xlab = 'Time', ylab = 'Density')
legend('topright', lty = 1, bty = 'n', col = rainbow(ncol(dV)),
       legend = paste('a = ', parsMatA[, 1]))

matplot(tt, dA, xlim = c(0, 5), main = 'Densities with different Drift Speed',
        col = rainbow(ncol(dA)), type = 'l', las = 1, bty = 'n',
        xlab = 'Time', ylab = 'Density')
legend('topright', lty = 1, bty = 'n', col = rainbow(ncol(dA)),
       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]),
                     bhattacharyya(tt, dV[i], dV[i + 1]),
                     hellinger(tt, dV[i], dV[i + 1]))
  distMatA[i, ] = c(chisq(tt, dA[i], dA[i + 1]),
                     bhattacharyya(tt, dA[i], dA[i + 1]),
                     hellinger(tt, dA[i], dA[i + 1]))
}
```

# The three distance measures correlate highly for differences in Boundary
# The battacharyya distance measures does not correlate with the others
# when calculating differences in drift speed

cor(distMatA)
cor(distMatV)
```

chisqFit

### Calculate model fit

#### Description

Calculate model fit

#### Usage

```r
chisqFit(resObserved, data, DstarM = FALSE, tt = NULL, formula = NULL)
```
Arguments

resObserved either output from estObserved or a matrix containing custom densities to calculate the fitness for.

data A dataframe containing data.

dstarm Logical. Should the DstarM fit measure be calculated or the traditional fit measure?

tt time grid custom densities where calculated on. Should only be supplied if resObserved is a matrix containing custom densities

formula Optional formula argument, for when columns names in the data are different from those used to obtain the results.

Details

This function allows a user to manually calculate a chi-square goodness of fit measure for model densities. This is useful for comparing a traditional analysis and a D*M analysis. For completion, this function can also calculate a D*M fit measure. We do not recommend usage of the D*M measure. While the chi-square fit measure is identical to the value of the optimizer when fitting, the DstarM fit measure is not equal to that of a DstarM analysis. This is because this function calculates the DstarM fit measure on the complete distribution, not on the model distributions, as is done during the optimization.

Examples

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
allDat = simData(n = 3e3, pars = pars, tt = tt, pdfND = pdfND, return.pdf = TRUE)
truePdf = allDat$pdfUnnormalized
dat = allDat$dat
chisqFit(resObserved = truePdf, data = dat, tt = tt)
## Not run:
# estimate it
define restriction matrix
restr = matrix(1:5, 5, 3)
restr[2, 2:3] = 6:7 # allow drift rates to differ
# fix parameters for speed up
fixed = matrix(c("z1", "al / 2", "sz1", .5, "sv1", .5), 2, 3)
resD = estDstarM(data = dat, tt = tt, restr = restr, fixed = fixed,
   Optim = list(parallelType = 1))
resN = estND(resD, Optim = list(parallelType = 1))

resO = estObserved(resD, resN, data = dat)
resO$fit # proper fit

## End(Not run)
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)
x = 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 Estimation', 'DstarM Estimation'),
col = 1:3, lty = c(1, NA, NA), pch = c(NA, 1, 1), bty = 'n')
**Description**

Do a D*M analysis

**Usage**

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

**Arguments**

- **formula** A formula object of the form: `binary response ~ reaction time + condition1 * condition2 * ...`
- **data** A dataframe for looking up data specified in formula. For backwards compatability this can also be 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 stepper 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, stepper = 50, CR = .9, trace = 0)`.
- **DstarM** If TRUE a D*M analysis is done, otherwise the Chi square distance between data and model is minimized.
- **SE** A positive value, how many standard error to add to the variance to relax the variance restriction a bit.
estDstarM

oscPdf Logical, if TRUE check for oscillations in calculated densities and remove densities with oscillations.

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. Take care that densities of response categories within conditions are degenerate and therefore integrate to the proportion a category was observed (and not to 1).

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 itermmax equal zero.

funNdensity Function used to calculate densities. See details.

argsNdensity A names list containing additional arguments to be send to funNdensity.

funNdist Function used to calculate distances between densities. Defaults to a chi-square distance.

argsNdist A named list containing additional arguments to be send to funNdist.

verbose Numeric, should intermediate output be printed? Defaults to 1, higher values result in more progress output. Estimation will speed up if set to 0. If set to TRUE, Optim$trace will be forced to 0, hereby disabling the build in printing of DEoptim. To enable the printing of DEoptim, set verbose to 0 and specify Optim$trace. Optim. If set to 1, ETA refers to the expected maximum time until completion (when the iterations limit is reached).

useRcpp Logical, setting this to true will make the objective function use an Rcpp implementation of Voss.density with the distance function chisq. This gains speed at the cost of flexibility.

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.

funNdensity 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 funNdensity via the argument argsNdensity. 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.

For purposes of speed, the function can be run in parallel by providing the argument Optim = list(parallelType = 1). See DEoptim.control for details. Also, for Ratcliff models the objective function has been rewritten in Rcpp. This limits some functionality but does result in a faster estimation. Usage of Rcpp can be enabled via useRcpp = TRUE.
When verbose is set to 1, the ETA is an estimated of the time it takes to execute ALL iterations. Convergence can (and is usually) reached before then.

**Value**

Returns a list of class `DstarM.fitD` that contains:

- **Bestvals**: Named numeric vector. Contains the best parameter estimates.
- **fixed**: Numeric vector. Contains the best parameter estimates.
- **GlobalOptimizer**: List. All output from the call to `DEoptim`
- **Debug**: List. Contains the number of `DEoptim` iterations, the number of function evaluation of the objective function, and the maximum number of iterations.
- **note**: String. A possible note that is used for summary purposes
- **tt**: Numeric vector. Contains the time grid used.
- **g.hat**: Numeric matrix. Named columns represent the (possibly smoothed) densities of the data distribution of each condition-response pair.
- **modelDist**: Numeric matrix. Named columns represent the densities of the model evaluated at grid `tt` with parameters `Bestvals`.
- **n.condition**: Numeric scalar. The number of conditions
- **var.data**: Numeric vector. The variance of each condition-response pair. There are as many values as hypothesized nondecision densities.
- **var.m**: Numeric vector. The variance of the model distributions. There are as many values as hypothesized nondecision densities.
- **restr.mat**: Numeric matrix. Contains the restrictions used.
- **splits**: Numeric vector. Equal to the input argument with the same name.
- **n**: Numeric scalar. The total number of observations.
- **DstarM**: Logical. Equal to the input argument with the same name.
- **fun.density**: Function. Equal to the input argument with the same name.
- **fun.dist**: Function. Equal to the input argument with the same name.
- **h**: Scalar. Equal to the input argument with the same name.
- **args.density**: Named list. Equal to the input argument with the same name.
- **args.dist**: Named list. Equal to the input argument with the same name.

**Examples**

```r
# 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
```
estND = dbeta(tt, 10, 30)
# simulate data
data = simData(n = 3e3, pars = pars, tt = tt, pdfND = pdfND)
# define restriction matrix
restr = matrix(c(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 density*

**Description**

Estimate nondecision density

**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, useRcpp = TRUE)
```

**Arguments**

- **res**: an object of class D*M.
- **tt**: optional timegrid if the nondecision density is to be estimated 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 is 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 sublist 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` is numeric, should intermediate output be printed? Defaults to 1, higher values result in more progress output. Estimation will speed up if set to 0. If nonzero, `Optim$trace` will be forced to 0, hereby disabling the build in printing of `DEoptim`. To enable the printing of `DEoptim`, set `verbose` to 0 and specify `Optim$trace`.

`dist` is a matrix where columns represent nondecision distributions. If this argument is supplied then the objective function will be evaluated in these values.

`NDindex` is a vector containing indices of which nondecision distributions to estimate. If omitted, all nondecision distributions that complement the results in `res` are estimated.

`max` is a positive float which indicates the maximum height of the nondecision distribution. If estimated nondecision distributions appear chopped off 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.

`useRcpp` is logical, setting this to true will make use of an Rcpp implementation of the objective function. This gains speed at the cost of flexibility.

**Details**

When `verbose` is set to 1, the ETA is an estimated of the time it takes to execute ALL iterations. Convergence can (and is usually) reached before then.

**Examples**

```r
# 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('szl', .5, 'svl', .5), 2, 2)
```
## 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

```r
estObserved(resDecision, resND = NULL, data = NULL, interpolateND = FALSE, tt = NULL)
```

### Arguments

- `resDecision` output of `estDstarM`
- `resND` output of `estND`
- `data` Optional. If the data used to estimate the decision model is supplied additional fitmeasures are calculated.
- `interpolateND` Logical. If the decision model and nondecision model have been estimated on different time grids, should the rougher time grid be interpolated to match the smaller grid? If FALSE (the default) the decision model will be recalculated on the grid of the nondecision model. This tends to produce better fit values.
- `tt` Optional time grid to recalculate the model densities on. Unused in case of a DstarM analysis.

### Value

Returns a list of class DstarM.fitObs 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) ($ND$).

---

```r
## 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)
```
The number of parameters used in the decision model.

A numeric vector containing indices of any not observed condition-response pairs.

**Examples**

```r
# 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$dat
# define restriction matrix
restr = matrix(1:5, 5, 3)
restr[2, 2:3] = 6:7 # allow drift rates to differ
# fix variance parameters
fixed = matrix(c('sl1', .5, 'sv1', .5), 2, 2)
## Not run:
# Run D*M analysis
resD = estDstarM(dat = dat, tt = tt, restr = restr, fixed = fixed)
# Estimate nondecision density
resND = estND(resD)
# Estimate observed density
resObs = estObesed(resD, resND)
# plot histograms with overlayed densities per condition-response pair
plotObesed(resObs, data = dat, xlim = c(0, 1))
# plot estimated and true densities
plot(resObs, col = rep(zl:3, each = 2), xlim = 0:1)
matlines(tt, lst$pdfNormalized, col = rep(zl:3, each = 2), lty = 2)
## End(Not run)
```

---

**estQdf**

**Estimate quantiles of distribution**

**Description**

Estimate quantiles of distribution

**Usage**

`estQdf(p, x, cdf)`
getPdfs

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 \( q[i] = \text{uniroot}(x, \text{cdf} = \text{p[i]}) \). 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

```r
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
```

getPdfs

(Re)Calculate model densities with given parameters and time grid

Description

This function is a convenience function for calculating model pdfs for multiple sets of parameters at a specified time grid. If resDecision is supplied, the density function and any additional arguments for the density function will be extracted from that object. If pars is missing these will also be extracted from this object. This function is intended to recalculate model densities at a new time grid.

Usage

getPdfs(resDecision, tt, pars, DstarM = TRUE, fun.density = Voss.density, args.density = list())
*getSter*

**Arguments**

- `resDecision` output of `estDstarM`
- `tt` Time grid to calculate the model densities on.
- `pars` Model parameters, can be a matrix where every column is a set of parameters.
- `DstarM` Logical. Do the model pdfs also describe the nondecision distribution?
- `fun.density` density function to calculate pdfs from.
- `args.density` Additional arguments for `fun.density`

**Value**

A matrix containing model pdfs.

---

**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, data, formula = NULL)`

**Arguments**

- `res`  
  An object of class D*M.

- `data`  
  The data object used to create `res`.

- `formula`  
  Optional formula argument, for when columns names in the data are different from those used to obtain the results.

**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 subtracting 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.

---

**normalize**

*Normalize two pdfs*

**Description**

Normalize two pdfs

**Usage**

`normalize(x, tt, props = NULL)`
Arguments

- **x**: Probability density function(s) evaluated at grid `x`. Input should be either a vector or matrix. If input is a matrix, each column represents a single pdf.
- **tt**: a numeric grid defined in `x`.
- **props**: the value each density should integrate to.

Examples

```r
tt <- seq(0, 9, length.out = 1e4)
# 2 proper densities
x1 <- cbind(dexp(tt, .5), dexp(tt, 2))
# still 2 proper densities
x2 <- normalize(10*x1, tt)
# 2 densities that integrate to .5
x3 <- normalize(x1, tt, props = c(.5, .5))
# plot the results
matplot(tt, cbind(x1, x2, x3), type = "l", ylab = "density",
        col = rep(1:3, each = 2), lty = rep(1:2, 3), las = 1, bty = "n")
legend("topright", legend = rep(paste0("x", 1:3), each = 2),
        col = rep(1:3, each = 2), lty = rep(1:2, 3), bty = "n")
```

---

**obsQuantiles**

```
Calculate model fit
```

Description

This function is nothing but a wrapper for `quantile`.

Usage

```r
obsQuantiles(data, probs = seq(0, 1, 0.01), what = "cr")
```

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.
- **probs**: vector of probabilities for which the corresponding values should be called
- **what**: Character. 'cr' if the quantiles are to be calculated per condition-response pair, 'c' if the quantiles are to be calculated per condition, and 'r' if the quantiles are to be calculated per response.
plotObserved

Examples

```
  tt = seq(0, 5, .01)
  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)
  probs = seq(0, 1, .01)
  q = obsQuantiles(data, probs = probs)
  matplot(probs, q, type = 'l', las = 1, bty = 'n')
```

plotObserved

Plot quantiles of data against model implied quantiles.

Description

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

Usage

```
plotObserved(resObserved, data, what = c("cr", "c", "r"), layout = NULL,
              main = NULL, linesArgs = list(), ggplot = FALSE, prob = seq(0, 1,
              .01), probType = 3, ...)
```

Arguments

- `resObserved`: output of `estObserved`
- `data`: The dataset used to estimate the model.
- `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`: Deprecated and ignored.
- `prob`: Should a qqplot of observed vs model implied quantiles be plotted? By default, it is `seq(0, 1, .01)`, the probabilities between 0 and 1 to compare the model implied quantiles to the observed quantiles. If this argument is `NULL`, then a histogram overlayed with model implied densities will be plotted. Internally, `estQdf` is used for generating quantiles.
- `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$dat
# define restriction matrix
restr = matrix(1:3, 3)
restr[2, 2:3] = 6:7 # allow drift rates to differ
# fix variance parameters
fixed = matrix(c('sz', .5, 'sv1', .5), 2, 2)
## Not run:
# Run D*M analysis
resD = estDstarM(dat = dat, tt = tt, restr = restr, fixed = fixed)
# Estimate nondecision density
resND = estND(resD)
# Estimate observed density
resObs = estObserved(resD, resND)
# plot histograms with overlayed densities per condition-response pair
plotObserved(resObs, data = dat, 
xlim = c(0, 1))
# plot estimated and true densities
plot(resObs, col = rep(1:3, each = 2), xlim = 0:1)
matlines(tt, lst$pdfNormalized, col = rep(1:3, each = 2), lty = 2)
# other uses of plotObserved
plotObserved(resObs = resObs, data = dat, what = 'cr', xlim = c(0, 1))
plotObserved(resObs = resObs, data = dat, what = 'c', xlim = c(0, 1))
plotObserved(resObs = resObs, data = dat, what = 'r', xlim = c(0, 1))

## End(Not run)
rtDescriptives

Descriptives of reaction time data

Usage

rtDescriptives(formula = NULL, data, plot = TRUE, verbose = TRUE)

Arguments

formula  A formula object of the form: binary response ~ reaction time + condition1 * condition2

data  A dataframe for looking up data specified in formula. For backwards compatability this can also be 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.

plot  Logical, should a density plot of all condition-response pairs be made?

verbose  Logical, should a table of counts and proportions be printed?

Details

This function and rthist are helper functions to inspect raw data.

Value

Invisibly returns an object of class 'D*M'. It's first element is table and contains raw counts and proportions for condition response pairs, conditions, and responses. It's second element plot contains a ggplot object.

Examples

tt <- seq(0, 5, .01)
pars <- matrix(.5, 5, 2)
pars[1, ] <- 1
pars[2, ] <- c(0, 2)
dat <- simData(n = 3e3, pars = pars, tt = tt, pdfND = dbeta(tt, 10, 30))
x <- rtDescriptives(data = dat)

print(x$table, what = 'cr')
print(x$table, what = 'c')
print(x$table, what = 'r')
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))
Simulate data from a given density function via multinomial sampling

**Description**

Simulate data from a given density function via multinomial sampling.

**Usage**

```r
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 correspond 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 response option lower or upper, and a column named `condition` indicating which condition a trial 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

\[
\begin{align*}
tt &= \text{seq}(0, 5, .01) \\
pdfND &= \text{dbeta}(tt, 10, 30) \\
n &= 100 \\
pars &= c(1, 2, .5, .5, .5) \\
dat &= \text{simData}(n, pars, tt, pdfND) \\
\text{head(dat)}
\end{align*}
\]

Description

Test fun.density with lower and upper bounds

Usage

\[
\text{testFun(fun.density, lower, upper, args = list())}
\]

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 occurred, otherwise returns an error message

Examples

\[
\begin{align*}
\text{lower} &= c(.5, -6, .1, 0, 0) \\
\text{upper} &= c(2, 6, .99, .99, 10) \\
\text{args} &= \text{list}(t = \text{seq}(0, 5, .01), \text{pars} = \text{lower}, \text{boundary} = \text{'lower'}, \text{DstarM} = \text{TRUE}) \\
\text{testFun(fun.density = Voss.density, lower = lower, upper = upper, args = args)} \\
& \quad \# \text{TRUE}
\end{align*}
\]
upgradeDstarM

Upgrade a DstarM object for backwards compatability

Description
Upgrade a DstarM object for backwards compatability

Usage
upgradeDstarM(x)

Arguments
x an object of class D*M or DstarM.

Value
An object of class DstarM.fitD, DstarM.fitND, or DstarM.fitObs.

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 ddiffusion without error handling can be found in the source code (commented out to pass R check). Note that for in Voss.density if DstarM == FALSE nondecision parameters are implemented manually and might differ from from how they are implemented in other packages. The parameter t0 specifies the mean of a uniform distribution and st0 specifies the relative size of this uniform distribution. To obtain the lower and upper range of the uniform distribution calculate a = t0 - t0*st0, and b = t0 + t0*st0.

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

approxfun, 13
battacharyya(chisq), 2
chisq, 2
chisqFit, 3
ddiffusion, 24
DEoptim, 8
DEoptim.control, 6, 7, 10
dlBA, 24
dwiener, 24
ecdf, 5
estCdf, 5, 13
estDstarM, 6, 11, 14
estNO, 9, 11
estObserved, 4, 11, 17
estQdf, 12, 17
getPdfs, 13
getSter, 14
getTer, 15
hellinger(chisq), 2
LBA.density(Voss.density), 23
lines, 17
normalize, 15
obsQuantiles, 16
plotObserved, 17
quantile, 16
rtDescriptives, 19, 20
rtHist, 19, 20
simData, 21
testFun, 7, 22, 24
upgradedDstarM, 23
Voss.density, 23
Wiener.density(Voss.density), 23