Package ‘decisionSupport’

April 13, 2015

Type Package

Title Quantitative Support of Decision Making under Uncertainty

Version 1.100.0.9005

Date 2015-04-13

Description Supporting the quantitative analysis of binary welfare based decision making processes using Monte Carlo simulations. Decision support is given on two levels: (i) The actual decision level is to choose between two alternatives under probabilistic uncertainty. This package calculates the optimal decision based on maximizing expected welfare. (ii) The meta decision level is to allocate resources to reduce the uncertainty in the underlying decision problem, i.e to increase the current information to improve the actual decision making process. This problem is dealt with using the Value of Information Analysis (VIA). The Expected Value of Information (EVI) for arbitrary prospective estimates can be calculated as well as Individual and Clustered Expected Value of Perfect Information (EVPI). The probabilistic calculations are done via Monte Carlo simulations. This Monte Carlo functionality can be used on its own.

License GPL-3

Depends R (>= 3.1.3)

Imports msm (>= 1.5),
mvtnorm (>= 1.0.2),
stats (>= 3.1.3)

Suggests eha (>= 2.4.2),
mc2d (>= 0.1.15),
nleqslv (>= 2.6),
rriskDistributions (>= 2.0),
testthat (>= 0.9.1),
knitr

URL http://www.worldagroforestry.org/

Encoding UTF-8

Classification/JEL I38, O16, O21, O22, O23

Collate 'rmvnorm90ci_exact.R'
'rdistq_fit.R'
'random.R'
'paramtnormci_numeric.R'
'paramtnormci_fit.R'
'rtnorm90ci.R'
'rdist90ci_exact.R'
'estimate1d.R'
'estimate.R'
'mcSimulation.R'
'welfareDecisionAnalysis.R'
'eviSimulation.R'
'individualEvpiSimulation.R'
'estimate_read_csv_old.R'
'uncertaintyAnalysis.R'
'decisionSupport-package.R'
'globalNames2data.frameNames.R'

VignetteBuilder knitr

R topics documented:

as.data.frame.mcSimulation 3
corMat 4
corMat.estimate 4
decisionSupport 5
estimate 7
estimate1d 8
estimate_read_csv 9
estimate_read_csv_old 11
estimate_write_csv 12
eviSimulation 13
globalNames2data.frameNames 17
hist.mcSimulation 18
individualEvpiSimulation 19
mcSimulation 21
names.estimate 23
paramtnormci_fit 24
paramtnormci_numeric 25
print.mcSimulation 26
print.summary.eviSimulation 27
print.summary.mcSimulation 27
print.summary.welfareDecisionAnalysis 28
random 28
random.estimate 30
random.estimate1d 31
rdist90ci_exact 33
rdistq_fit 34

rmvnorm90ci_exact 36
row.names.estimate 36
rtnorm90ci 37
sort.summary.eviSimulation 38
summary.eviSimulation 39
summary.mcSimulation 39
summary.welfareDecisionAnalysis 40
uncertaintyAnalysis 41
welfareDecisionAnalysis 42

Index **45**

`as.data.frame.mcSimulation` *Coerce to a Data Frame.*

Description

Functions to check if an object is a data frame, or coerce it if possible.

Usage

```
## S3 method for class 'mcSimulation'
as.data.frame(x, row.names = NULL, optional = FALSE,
  ..., stringsAsFactors = default.stringsAsFactors())
```

Arguments

`x`	An object of class `mcSimulation`.
`row.names`	`NULL` or a character vector giving the row names for the data frame. Missing values are not allowed.
`optional`	logical. If `TRUE`, setting row names and converting column names (to syntactic names: see `make.names`) is optional.
`...`	additional arguments to be passed to or from methods.
`stringsAsFactors`	logical: should the character vector be converted to a factor?

See Also

`as.data.frame`

corMat *Return the Correlation Matrix of x.*

Description

Return the correlation matrix of x.

Usage

```
corMat(rho)
```

Arguments

`rho`	a distribution.

corMat.estimate *Return the correlation matrix of an `estimate` object.*

Description

This function returns the full correlation matrix of an `estimate` object.

Usage

```
## S3 method for class 'estimate'
corMat(rho)
```

Arguments

`rho`	an `estimate` object.

See Also

`estimate`, `row.names.estimate`, `names.estimate` `decisionSupport` *Quantitative Support of Decision Making under Uncertainty.*

Description

The `decisionSupport` package supports the quantitative analysis of welfare based decision making processes using Monte Carlo simulations. This is an important part of the Applied Information Economics (AIE) approach developed in Hubbard (2014). These decision making processes can be categorized into two levels of decision making:

1. The actual problem of interest of a policy maker which we call the *underlying welfare based decision* on how to influence an ecological-economic system based on a particular information on the system available to the decision maker and
2. the *meta decision* on how to allocate resources to reduce the uncertainty in the underlying decision problem, i.e to increase the current information to improve the underlying decision making process.

The first problem, i.e. the underlying problem, is the problem of choosing the decision which maximizes expected welfare. The welfare function can be interpreted as a von Neumann-Morgentstern utility function. Whereas, the second problem, i.e. the meta decision problem, is dealt with using the *Value of Information Analysis (VIA)*. Value of Information Analyis seeks to assign a value to a certain reduction in uncertainty or, equivalently, increase in information. Uncertainty is dealt with in a probabilistic manner. Probabilities are transformed via Monte Carlo simulations.

Details

The functionality of this package is subdivided into three main parts: (i) the welfare based analysis of the underlying decision, (ii) the meta decision of reducing uncertainty and (iii) the Monte Carlo simulation for the transformation of probabilities and calculation of expectation values. Furthermore, there is a wrapper function around these three parts which aims at providing an easy-to-use interface.

Welfare based Analysis of the Underlying Decision Problem:

Welfare Decision Analysis: Implementation: `welfareDecisionAnalysis`

Utility Functions: Implementation: ToDo

The Meta Decision of Reducing Uncertainty: The meta decision of how to allocate resources for uncertainty reduction can be analyzed with this package in two different ways: via (i) Expected Value of Information Analysis or (ii) via Partial Least Squares (PLS) analysis and Variable Importance in Projection (VIP).

Expected Value of Information (EVI): Implementation: `eviSimulation`, `individualEvpiSimulation`

Partial Least Squares (PLS) analysis and Variable Importance in Projection (VIP): Implementation: ToDo

Solving the Practical Problem of Calculating Expectation Values by Monte Carlo Simulation:

Estimates: Implementation: `estimate`

Multivariate Ranom Number Generation: Implementation: `random.estimate`

Monte Carlo Simulation: Implementation: `mcSimulation`

Uncertainty Analysis: A wrapper function: Implementation: `uncertaintyAnalysis`

Package Options

ToDo

Copyright ©

License

Author(s)

Lutz Göhring <`lutz.goehring@gmx.de`>, Eike Luedeling (ICRAF) <`E.Luedeling@cgiar.org`>

Maintainer: Lutz Göhring <`lutz.goehring@gmx.de`>

References

Hubbard, Douglas W., *How to Measure Anything? - Finding the Value of "Intangibles" in Business*, John Wiley & Sons, Hoboken, New Jersey, 2014, 3rd Ed, `http://www.howtomeasureanything.com/`.

Hugh Gravelle and Ray Rees, *Microeconomics*, Pearson Education Limited, 3rd edition, 2004.

See Also

`welfareDecisionAnalysis`, `eviSimulation`, `mcSimulation` `estimate` *Create a multivariate estimate object*

Description

This function creates an object of class `estimate`. It extends the univariate estimate `estimate1d` to the multivariate case. This includes the description of correlations between the different variables.

Usage

```
estimate(..., correlation_matrix = NULL)
```

Arguments

`...` arguments that can be coerced to a data frame comprising the base of the estimate. Mandatory columns are `distribution`, `lower` and `upper`.

`correlation_matrix` `numeric matrix`: containing the correlations of the variables.

Details

The parameters in `...` provide the base information of an estimate.

The structure of the estimate base information (mandatory): Mandatory columns:

Column name	R-type	Explanation
`distribution`	`character`	Distribution types
`variable`	`character`	Variable names

Value

An object of class `estimate` which is a list whith components `base` and `correlation_matrix`:

`$base` is a `data.frame` with mandatory column `distribution`. The `row.names` are the names of the variables. Each row has the properties of an `estimate1d`.

`$correlation_matrix` is a symmetric matrix with row and column names being the subset oft he variables supplied in `base` which are correlated. Its elements are the corresponding correlations.

See Also

`row.names.estimate`, `names.estimate`, `corMat`, `estimate_read_csv`, `estimate_write_csv`, `random.estimate`, `estimate1d` `estimate1d` *Create a 1-dimensional estimate object.*

Description

`estimate1d` creates an object of class `estimate1d`. A one dimensional estimate is at minimum characterized by the type of a univariate parametric distribution, the 5% - and 95% quantiles. Optionally, the median can be supplied.

`as.estimate1d` tries to transform an object to class `estimate1d`.

Usage

```
estimate1d(distribution, lower, upper, ...)

as.estimate1d(x, ...)
```

Arguments

`distribution`	`character`; A character string that defines the type of the univariate parametric distribution.
`lower`	`numeric`; lower bound of the 90% confidence intervall, i.e the 5%-quantile of this estimate.
`upper`	`numeric`; upper bound of the 90% confidence intervall, i.e the 95%-quantile of this estimate.
`...`	arguments that can be coerced to a list comprising further elements of the 1-d estimate (for details cf. below). Each element must be atomic and of length 1 (1-d property).
`x`	an object to be transformed to class `estimate1d`.

Details

It must hold that `lower <= upper`.

The structure of the input arguments:

Mandatory input elements::

Argument	R-type	Explanation
`distribution`	`character`	Distribution type of the estimate
`lower`	`numeric`	5%-quantile of the estimate
`upper`	`numeric`	95%-quantile of the estimate

Optional input elements:: The optional parameters in `...` provide additional characteristics of the 1-d estimate. Frequent optional elements are:

Argument	R-type	Explanation
`variable`	`character`	Variable name

`median`	cf. below	50%-quantile of the estimate
`method`	`character`	Method for calculation of distribution parameters

The `median`*:* If supplied as input, `median` can be either `NULL`, `numeric` or the character string `"mean"`. If it is `NA` it is set to `NULL`; if it equals `"mean"` it is set to `mean(c(lower, upper))`; if it is `numeric` it must hold that `lower <= median <= upper`. In case that no element `median` is provided, the default is `median=NULL`.

Value

An object of class `estimate1d` and `list` with at least (!) the elements:

Element	**R-type**	**Explanation**
`distribution`	`character`	Distribution type of the estimate
`lower`	`numeric`	5%-quantile of the estimate
`median`	`numeric` or `NULL`	50%-quantile of the estimate
`upper`	`numeric`	95%-quantile of the estimate

Note that the `median` is a mandatory element of an `estimate1d`, although it is not necessary as input. If `median` is numeric it holds that: `lower <= median <= upper`. In any case an `estimate1d` object has the property `lower <= upper`.

See Also

`random.estimate1d`

`estimate_read_csv` *Read an Estimate from CSV - File.*

Description

This function reads an `estimate` from the specified csv files. In this context, an estimate of a variable is defined by its distribution type, its 90%-confidence interval `[lower,upper]` and its correlation to other variables. #ToDo: Implement characterization of distribution by mean and sd. Eventually, also by other quantiles.

Usage

```
estimate_read_csv(fileName, strip.white = TRUE, ...)
```

Arguments

`fileName` Name of the file containing the base information of the estimate that should be read.

`strip.white` logical. Allows the stripping of leading and trailing white space from unquoted character fields (numeric fields are always stripped). See `scan` for further details (including the exact meaning of 'white space'), remembering that the columns may include the row names.

...	Further parameters to be passed to read.csv.

Details

An estimate might consists of uncorrelated and correlated variables. This is reflected in the input file structure, which is described in the following.

Value

An object of type estimate.

CSV input file structures

The estimate is read from one or two csv files: the basic csv file which is mandatory and the correlation csv file which is optional. The basic csv file contains the definition of the distribution of all variables ignoring potential correlations. The correlation csv file only defines correlations.

The structure of the basic input file (mandatory): File name structure: `<basic-filename>.csv`
Mandatory columns:

Column name	R-type	Explanation
`lower`	`numeric`	ToDo
`upper`	`numeric`	ToDo
`distribution`	`character`	ToDo
`variable`	`character`	ToDo

Optional columns:

Column name	R-type	Explanation
`description`	`character`	ToDo
`median`	`numeric`	ToDo
`start`	`integer`	ToDo
`end`	`integer`	ToDo
`indicator`	`logical`	ToDo

Columns without names are ignored. Rows where the `variable` field is empty are also dropped.

The structure of the correlation file (optional): File name structure: `<basic-filename>_cor.csv`
Columns and rows are named by the corresponding variables. Only those variables need to be present which are correlated with others. The element `["rowname","columnname"]` contains the correlation between the variables `rowname` and `columnname`. Uncorrelated elements can be left empty, i.e. as `NA`, or defined as `0`. The element `["name","name"]` has to be set to `1`. The matrix must be given in symmetric form.

See Also

estimate_write_csv, read.csv, estimate

estimate_read_csv_old *Read an Estimate from CSV - File (depreciated standard).*

Description

This function reads an `estimate` from the specified csv files. In this context, an estimate of a variable is defined by its distribution type, its 90%-confidence interval `[lower,upper]` and its correlation to other variables. #ToDo: Implement characterization of distribution by mean and sd. Eventually, also by other quantiles.

Usage

```
estimate_read_csv_old(fileName, strip.white = TRUE, ...)
```

Arguments

`fileName`	Name of the file containing the base information of the estimate that should be read.
`strip.white`	logical. Allows the stripping of leading and trailing white space from unquoted character fields (numeric fields are always stripped). See `scan` for further details (including the exact meaning of 'white space'), remembering that the columns may include the row names.
`...`	Further parameters to be passed to `read.csv`.

Details

An estimate might consists of uncorrelated and correlated variables. This is reflected in the input file structure, which is described in the following.

Value

An object of type `estimate`.

CSV input file structures

The estimate is read from one or two csv files: the basic csv file which is mandatory and the correlation csv file which is optional. The basic csv file contains the definition of the distribution of all variables ignoring potential correlations. The correlation csv file only defines correlations.

The structure of the basic input file (mandatory): File name structure: `<basic-filename>.csv`
Mandatory columns:

Column name	R-type	Explanation
`lower`	`numeric`	ToDo
`upper`	`numeric`	ToDo
`distribution`	`character`	ToDo
`variable`	`character`	ToDo

Optional columns:

Column name	R-type	Explanation
description	character	ToDo
median	numeric	ToDo
start	integer	ToDo
end	integer	ToDo
indicator	logical	ToDo

Columns without names are ignored. Rows where the `variable` field is empty are also dropped.

The structure of the correlation file (optional): File name structure: `<basic-filename>.csv_correlations.csv` #ToDo

See Also

`estimate_read_csv`, `read.csv`, `estimate`

`estimate_write_csv` *Write an Estimate to CSV - File.*

Description

This function writes an `estimate` to the specified csv file(s).

Usage

```
estimate_write_csv(estimate, fileName, varNamesAsColumn = TRUE,
  quote = FALSE, ...)
```

Arguments

`estimate` Estimate object to write to file `fileName`.

`fileName` `character`. Output file name which must end with `.csv`.

`varNamesAsColumn` `logical`; If `TRUE` the variable names will be written as a separate column, otherwise as row names.

`quote` a `logical` value (TRUE or FALSE) or a numeric vector. If TRUE, any character or factor columns will be surrounded by double quotes. If a numeric vector, its elements are taken as the indices of columns to quote. In both cases, row and column names are quoted if they are written. If FALSE, nothing is quoted. Parameter is passed on to `write.csv`.

`...` Further parameters to be passed to `write.csv`.

Value

An object of type `estimate`.

See Also

estimate_read_csv, estimate, write.csv

---

eviSimulation *Expected Value of Information (EVI) Simulation.*

Description

The Expected Value of Information (EVI) is calculated based on a Monte Carlo simulation of the values of two different decision alternatives.

Usage

```
eviSimulation(model, currentEstimate, prospectiveEstimate, numberOfSimulations,
  functionSyntax = "data.frameNames")
```

Arguments

model	either a function or a list with two functions: `list(p1,p2)`. In the first case the function is the net benefit of project approval vs. the status quo. In the second case the element `p1` is the function valuing the first project and the element `p2` valueing the second project.
currentEstimate	estimate object describing the distribution of the input variables as currently estmated.
prospectiveEstimate	estimate object describing the prospective distribution of the input variables which could hypothetically achieved by collecting more information, viz. improving the measurement.
numberOfSimulations	integer; number of simulations to be used in the underlying Monte Carlo analysis
functionSyntax	function character; function syntax used in the model function(s).

Details

This principle is along the line described in Hubbard (2014). The Expected Value of Information is the decrease in the EOL for an information improvement from the current estimate (I_current) to a better prospective (or hypothetical) information (I_prospective): EVI := EOL(I_current) - EOL(I_prospective). Thus, the EVI depends on the model for valueing a decision, the current information, i.e. the current estimate, and the specification of a hypothetical improvement in information, i.e. a prospective estimate.

Value

An object of class `eviSimulation` with the following elements:

current	welfareDecisionAnalysis object for `currentEstimate`
prospective	welfareDecisionAnalysis object for `prospectiveEstimate`
evi	Expected Value of Information (EVI) of gained by the prospective estimate w.r.t. the current estimate

See Also

welfareDecisionAnalysis, mcSimulation, estimate

Examples

```
#############################################################
# Example 1 Only one prospective estimate:
#############################################################
numberOfSimulations=10000
# Create the estimate object:
variable=c("revenue","costs")
distribution=c("posnorm","posnorm")
lower=c(10000,  5000)
upper=c(100000, 50000)
currentEstimate<-estimate(variable, distribution, lower, upper)
prospectiveEstimate<-currentEstimate
revenueConst<-mean(c(currentEstimate$base["revenue","lower"],
                     currentEstimate$base["revenue","upper"]))
prospectiveEstimate$base["revenue",]<-data.frame(distribution="const",
                                                 lower=revenueConst,
                                                 upper=revenueConst,
                                                 row.names="revenue",
                                                 stringsAsFactors=FALSE)
# (a) Define the model function without name for the return value:
profit<-function(x){
x$revenue-x$costs
}

# Calculate the Expected Value of Information:
eviSimulationResult<-eviSimulation(model=profit,
                                   currentEstimate=currentEstimate,
                                   prospectiveEstimate=prospectiveEstimate,
                                   numberOfSimulations=numberOfSimulations,
                                   functionSyntax="data.frameNames")
# Show the simulation results:
print(summary(eviSimulationResult))
#############################################################
# (b) Define the model function with a name for the return value:
profit<-function(x){
list(Profit=x$revenue-x$costs)
}
# Calculate the Expected Value of Information:
eviSimulationResult<-eviSimulation(model=profit,
                                   currentEstimate=currentEstimate,
                                   prospectiveEstimate=prospectiveEstimate,
                                   numberOfSimulations=numberOfSimulations,
                                   functionSyntax="data.frameNames")
# Show the simulation results:
print(summary((eviSimulationResult)))
#############################################################
# (c) Two decision variables:
decisionModel<-function(x){
```

```
 list(Profit=x$revenue-x$costs,
      Costs=-x$costs)
}
# Calculate the Expected Value of Information:
eviSimulationResult<-eviSimulation(model=decisionModel,
                                   currentEstimate=currentEstimate,
                                   prospectiveEstimate=prospectiveEstimate,
                                   numberOfSimulations=numberOfSimulations,
                                   functionSyntax="data.frameNames")
# Show the simulation results:
print(summary((eviSimulationResult)))
#############################################################
# Example 2 A list of prospective estimates:
#############################################################
numberOfSimulations=10000
#  Define the model function with a name for the return value:
profit<-function(x){
 list(Profit=x$revenue-x$costs)
}
# Create the estimate object:
variable=c("revenue","costs")
distribution=c("posnorm","posnorm")
lower=c(10000,  5000)
upper=c(100000, 50000)
currentEstimate<-estimate(variable, distribution, lower, upper)
perfectInformationRevenue<-currentEstimate
revenueConst<-mean(c(currentEstimate$base["revenue","lower"],
                     currentEstimate$base["revenue","upper"]))
perfectInformationRevenue$base["revenue",]<-data.frame(distribution="const",
                                                       lower=revenueConst,
                                                       upper=revenueConst,
                                                       row.names="revenue",
                                                       stringsAsFactors=FALSE)
# (a) A list with one element
prospectiveEstimate<-list(perfectInformationRevenue=perfectInformationRevenue)
# Calculate the Expected Value of Information:
eviSimulationResult<-eviSimulation(model=profit,
                                   currentEstimate=currentEstimate,
                                   prospectiveEstimate=prospectiveEstimate,
                                   numberOfSimulations=numberOfSimulations,
                                   functionSyntax="data.frameNames")
# Show the simulation results:
print(summary(eviSimulationResult))
#############################################################
# (b) A list with two elements
perfectInformationCosts<-currentEstimate
costsConst<-mean(c(currentEstimate$base["costs","lower"],
                   currentEstimate$base["costs","upper"]))
perfectInformationCosts$base["costs",]<-data.frame(distribution="const",
                                                   lower=costsConst,
                                                   upper=costsConst,
                                                   row.names="costs",
                                                   stringsAsFactors=FALSE)
```

```
prospectiveEstimate<-list(perfectInformationRevenue=perfectInformationRevenue,
                          perfectInformationCosts=perfectInformationCosts)
# Calculate the Expected Value of Information:
eviSimulationResult<-eviSimulation(model=profit,
                                   currentEstimate=currentEstimate,
                                   prospectiveEstimate=prospectiveEstimate,
                                   numberOfSimulations=numberOfSimulations,
                                   functionSyntax="data.frameNames")
# Show the simulation results:
print(summary(eviSimulationResult))
#############################################################
# Example 3 A list of prospective estimates and two decision variables:
#############################################################
numberOfSimulations=10000
# Create the current estimate object:
variable=c("revenue","costs")
distribution=c("posnorm","posnorm")
lower=c(10000,  5000)
upper=c(100000, 50000)
currentEstimate<-estimate(variable, distribution, lower, upper)
# Create a list of two prospective estimates:
perfectInformationRevenue<-currentEstimate
revenueConst<-mean(c(currentEstimate$base["revenue","lower"],
                     currentEstimate$base["revenue","upper"]))
perfectInformationRevenue$base["revenue",]<-data.frame(distribution="const",
                                                       lower=revenueConst,
                                                       upper=revenueConst,
                                                       row.names="revenue",
                                                       stringsAsFactors=FALSE)
perfectInformationCosts<-currentEstimate
costsConst<-mean(c(currentEstimate$base["costs","lower"],currentEstimate$base["costs","upper"]))
perfectInformationCosts$base["costs",]<-data.frame(distribution="const",
                                                   lower=costsConst,
                                                   upper=costsConst,
                                                   row.names="costs",
                                                   stringsAsFactors=FALSE)
prospectiveEstimate<-list(perfectInformationRevenue=perfectInformationRevenue,
                          perfectInformationCosts=perfectInformationCosts)
# Define the model function with two decision variables:
decisionModel<-function(x){
 list(Profit=x$revenue-x$costs,
      Costs=-x$costs)
}
# Calculate the Expected Value of Information:
eviSimulationResult<-eviSimulation(model=decisionModel,
                                   currentEstimate=currentEstimate,
                                   prospectiveEstimate=prospectiveEstimate,
                                   numberOfSimulations=numberOfSimulations,
                                   functionSyntax="data.frameNames")
# Show the simulation results:
print(sort(summary(eviSimulationResult)),decreasing=TRUE,along="Profit")
```

globalNames2data.frameNames

Transform model function variable names: global to data.frame names.

Description

The variable names of a function are transformed from global variable names to data.frame names of the form `x$<globalName>`.

Usage

```
globalNames2data.frameNames(modelFunction, globalNames)
```

Arguments

`modelFunction`	a function which body contains global variables. The function must not contain any arguments.
`globalNames`	a `character` vector containing the names of the global variables that shall be transformed.

Details

The input function must be of the form:

```
modelFunction<-function(){
  ...
  <expression with variable1>
  ...
}
```

Value

The transformed function which is of the form:

```
function(x){
  ...
  <expression with x$variable1>
  ...
}
```

Warning

If there are local functions within the function `modelFunction` defined, which arguments have identical names to any of the `globalNames` the function fails!

See Also

mcSimulation, estimate

Examples

```
profit1<-function(){
   list(Profit=revenue-costs)
 }
 profit2<-globalNames2data.frameNames(modelFunction=profit1, globalNames=c("revenue", "costs"))
 print(profit2)
 is.function(profit2)
 profit2(data.frame("revenue"=10,"costs"=2))
```

---

hist.mcSimulation — *Plot Histogram of results of a Monte Carlo Simulation*

Description

This function plots the histograms of the results of mcSimulation.

Usage

```
## S3 method for class 'mcSimulation'
hist(x, breaks = 100, col = NULL, xlab = NULL,
  main = paste("Histogram of ", xlab), ..., colorQuantile = c("GREY",
  "YELLOW", "ORANGE", "DARK GREEN", "ORANGE", "YELLOW", "GREY"),
  colorProbability = c(1, 0.95, 0.75, 0.55, 0.45, 0.25, 0.05),
  resultName = NULL)
```

Arguments

`x` An object of class `mcSimulation`.

`breaks` one of:

- a vector giving the breakpoints between histogram cells,
- a function to compute the vector of breakpoints,
- a single number giving the number of cells for the histogram,
- a character string naming an algorithm to compute the number of cells (see 'Details'),
- a function to compute the number of cells.

In the last three cases the number is a suggestion only; the breakpoints will be set to `pretty` values. If `breaks` is a function, the `x` vector is supplied to it as the only argument.

`col` a colour to be used to fill the bars. The default of `NULL` yields unfilled bars.

`xlab` `character`; x label of the histogram. If it is not provided, i.e. equals `NULL` the name of the chosen variable by argument `resultName` is used.

main — character; main title of the histogram.

... — Further arguments to be passed to hist.

colorQuantile — character vector encoding the color of the quantiles defined in argument colorProbability.

colorProbability — numeric vector; defines the quantiles that shall be distinguished by the colors chosen in argument colorQuantile. Must be of the same length as colorQuantile.

resultName — character; indicating the name of the component of the simulation function (model_function) which results histogram shall be generated. If model_function is single valued, no name needs to be supplied. Otherwise, one valid name has to be specified. Defaults to NULL.

Value

an object of class "histogram". For details see hist.

See Also

mcSimulation, hist. For a list of colors available in R see colors.

---

individualEvpiSimulation *Individual Expected Value of Perfect Information Simulation*

Description

The Individual Expected Value of Perfect Information (Individual EVPI) is calculated based on a Monte Carlo simulation of the values of two different decision alternatives.

Usage

```
individualEvpiSimulation(model, currentEstimate,
  perfectProspectiveNames = row.names(currentEstimate),
  perfectProspectiveValues = colMeans(random(rho = currentEstimate, n =
  numberOfSimulations)[, perfectProspectiveNames]), numberOfSimulations,
  functionSyntax = "data.frameNames")
```

Arguments

model — either a function or a list with two functions: list(p1,p2). In the first case the function is the net benefit of project approval vs. the status quo. In the second case the element p1 is the function valuing the first project and the element p2 valueing the second project.

currentEstimate — estimate object describing the distribution of the input variables as currently estmated.

`perfectProspectiveNames`
: character vector; input variable names that are assumed to be known perfectly with prospective information.

`perfectProspectiveValues`
: numeric vector of the same length as `perfectProspectiveNames` with the corresponding values assumed to be known perfectly.

`numberOfSimulations`
: integer; number of simulations to be used in the underlying Monte Carlo analysis

`functionSyntax`
: function character; function syntax used in the model function(s).

Details

This principle is along the line described in Hubbard (2014). The Expected Value of Information is the decrease in the EOL for an information improvement from the current estimate (I_current) to a better prospective (or hypothetical) information (I_prospective): EVI := EOL(I_current) - EOL(I_prospective). If one variables under I_prospective is assumed to be known with certainty the EVI is called the Individual Expected Value of Perfect Information (Individual EVPI). More precisely, if one assumes under I_prospective to perfectly know (x_1, ..., x_k) to equal (a_1, ..., a_k) then one can specify the notation as Individual EVPI[x_i = a_i]. Summarizing, the Individual EVPI depends on the model for valueing a decision, the current information, i.e. the current estimate, and the specification of the variable that is assumed to be known with certainty, viz. the improvement in information, i.e. a prospective estimate.

Value

An object of class `eviSimulation` with the following elements:

`current`	`welfareDecisionAnalysis` object for `currentEstimate`
`prospective`	`welfareDecisionAnalysis` object for `prospectiveEstimate`
`evi`	Expected Value of Information (EVI) of gained by the prospective estimate w.r.t. the current estimate

See Also

`eviSimulation`, `welfareDecisionAnalysis`, `mcSimulation`, `estimate`

Examples

```
# Number of simulations:
n=100000
# Create the current estimate from text:
estimateText<-"variable,  distribution, lower, upper
               revenue1,  posnorm,      100,   1000
               revenue2,  posnorm,      50,    2000
               costs1,    posnorm,      50,    2000
               costs2,    posnorm,      100,   1000"
currentEstimate<-estimate(read.csv(header=TRUE, text=estimateText,
                          strip.white=TRUE, stringsAsFactors=FALSE))
# The model function:
profitModel <- function(x){
 list(Profit=x$revenue1 + x$revenue2 - x$costs1 - x$costs2)
```

```
}
# Calculate the Individual EVPI:
individualEvpiResult<-individualEvpiSimulation(model=profitModel,
                                               currentEstimate=currentEstimate,
                                               numberOfSimulations=n,
                                               functionSyntax="data.frameNames")
# Show the simulation results:
print(sort(summary(individualEvpiResult)),decreasing=TRUE,along="Profit")
```

`mcSimulation` *Perform a Monte Carlo Simulation.*

Description

This method solves the following problem. Given a multivariate random variable $x = (x_1, \ldots, x_k)$ with joint probability distribution P, i.e.

$$x \sim P.$$

Then the continuous function

$$f : R^k \to R^l, y = f(x)$$

defines another random variable with distribution

$$y \sim f(P).$$

Given a probability density ρ of x that defines P the problem is the determination of the probability density ϕ that defines $f(P)$. This method samples the probability density ϕ of y by Monte Carlo simulation.

Usage

```
mcSimulation(estimate, model_function, ..., numberOfSimulations,
  randomMethod = "calculate", functionSyntax = "data.frameNames")
```

Arguments

`estimate`	Filename or estimate object representing the joint probability distribution of the input variables.
`model_function`	A numeric function; The function that describes the value of a certain project.
`...`	Optional arguments of `model_function`.
`numberOfSimulations`	The number of Monte Carlo simulations to be run.
`randomMethod`	`character`. The method to be used to sample the distribution representing the input estimate.
`functionSyntax`	`character`. The syntax which has to be used to implement the model function. Possible values are `globalNames`, `data.frameNames` or `matrixNames`. Details are given below.

Details

If `functionSyntax="globalNames"`, the variable names used in the definition of `model_function` have to be defined globally. `model_function` has to be of the form `function(x,varnames)`. If `functionSyntax="data.frameNames"`, the model function is constructed, e.g. like this: `profit<-function(x){ x[["revenue"]]-x[["costs"]] }` or like this: `profit<-function(x){ x$revenue-x$costs }` If `functionSyntax="matrixNames"`, the model function is constructed, e.g. like this: `profit<-function(x){ x[,"revenue"]-x[,"costs"] }`

Value

An object of class `mcSimulation`.

phi	an l-variate probability distribution
x	a dataframe containing the sampled $x-$ values
y	a dataframe containing the simulated $y-$ values

See Also

print.mcSimulation, summary.mcSimulation, hist.mcSimulation, estimate, random.estimate

Examples

```
#############################################################
 # Example 1 (Creating the estimate from the command line):
 #############################################################
 # Create the estimate object:
 variable=c("revenue","costs")
 distribution=c("norm","norm")
 lower=c(10000,  5000)
 upper=c(100000, 50000)
 costBenefitEstimate<-estimate(variable, distribution, lower, upper)
 # (a) Define the model function without name for the return value:
 profit1<-function(x){
   x$revenue-x$costs
 }
 # Perform the Monte Carlo simulation:
 predictionProfit1<-mcSimulation( estimate=costBenefitEstimate,
                                 model_function=profit1,
                                 numberOfSimulations=100000,
                                 functionSyntax="data.frameNames")
 # Show the simulation results:
 print(summary(predictionProfit1))
 hist(predictionProfit1,xlab="Profit")
 #############################################################
 # (b) Define the model function with a name for the return value:
 profit1<-function(x){
   list(Profit=x$revenue-x$costs)
 }
 # Perform the Monte Carlo simulation:
 predictionProfit1<-mcSimulation( estimate=costBenefitEstimate,
```

```
                                model_function=profit1,
                                numberOfSimulations=100000,
                                functionSyntax="data.frameNames")
# Show the simulation results:
print(summary(predictionProfit1, classicView=TRUE))
hist(predictionProfit1)
#########################################################
# (c) Using global names in the model function syntax
# (CAVE: currently slow!):
profit1<-function(){
  list(Profit=revenue-costs)
}
# Perform the Monte Carlo simulation:
predictionProfit1<-mcSimulation( estimate=costBenefitEstimate,
                                model_function=profit1,
                                numberOfSimulations=10000,
                                functionSyntax="globalNames")
# Show the simulation results:
print(summary(predictionProfit1, probs=c(0.05,0.50,0.95)))
hist(predictionProfit1)

#############################################################
# Example 2(Reading the estimate from file):
#############################################################
# Define the model function:
profit2<-function(x){
  Profit<-x[["sales"]]*(x[["productprice"]] - x[["costprice"]])
  list(Profit=Profit)
}
# Read the estimate of sales, productprice and costprice from file:
inputFileName=system.file("extdata","profit-4.csv",package="decisionSupport")
parameterEstimate<-estimate_read_csv(fileName=inputFileName)
print(parameterEstimate)
# Perform the Monte Carlo simulation:
predictionProfit2<-mcSimulation( estimate=parameterEstimate,
                                model_function=profit2,
                                numberOfSimulations=100000,
                                functionSyntax="data.frameNames")
# Show the simulation results:
print(summary(predictionProfit2))
hist(predictionProfit2)
```

---

names.estimate *Return the column names of an* estimate *object.*

Description

This function returns the column names of an estimate object which is identical to names(x$base).

Usage

```
## S3 method for class 'estimate'
names(x)
```

Arguments

x	an estimate object.

See Also

estimate, row.names.estimate, corMat.estimate

---

paramtnormci_fit   *Fit parameters of truncated normal distribution based on a confidence interval.*

Description

This function fits the distribution parameters, i.e. mean and sd, of a truncated normal distribution from an arbitrary confidence interval and, facultatively, the median.

Usage

```
paramtnormci_fit(p, ci, median = mean(ci), lowerTrunc = -Inf,
  upperTrunc = Inf, relativeTolerance = 0.05, fitMethod = "Nelder-Mead",
  ...)
```

Arguments

p	numeric 2-dimensional vector; probabilities of upper and lower bound of the corresponding confidence interval.
ci	numeric 2-dimensional vector; lower, i.e ci[[1]], and upper bound, i.e ci[[2]], of the confidence interval.
median	if NULL: truncated normal is fitted only to lower and upper value of the confidence interval; if numeric: truncated normal is fitted on the confidence interval and the median simultaneously. For details cf. below.
lowerTrunc	numeric; lower truncation point of the distribution (>= -Inf).
upperTrunc	numeric; upper truncation point of the distribution (<= Inf).
relativeTolerance	numeric; the relative tolerance level of deviation of the generated probability levels from the specified confidence interval. If the relative deviation is greater than relativeTolerance a warning is given.
fitMethod	optimization method used in constrOptim.
...	further parameters to be passed to constrOptim.

Details

For details of the truncated normal distribution see `tnorm`.

The cummulative distribution of a truncated normal $F_{\mu,\sigma}$(x) gives the probability that a sampled value is less than x. This is equivalent to saying that for the vector of quantiles $q = (q_{p_1}, \ldots, q_{p_k})$ at the corresponding probabilities $p = (p_1, \ldots, p_k)$ it holds that

$$p_i = F_{\mu,\sigma}(q_{p_i}),\ i = 1, \ldots, k$$

In the case of arbitrary postulated quantiles this system of equations might not have a solution in μ and σ. A least squares fit leads to an approximate solution:

$$\sum_{i=1}^{k}(p_i - F_{\mu,\sigma}(q_{p_i}))^2 = \min$$

defines the parameters μ and σ of the underlying normal distribution. This method solves this minimization problem for two cases:

1. `ci[[1]] < median < ci[[2]]`: The parameters are fitted on the lower and upper value of the confidence interval and the median, formally:
   $k = 3$
   p_1=`p[[1]]`, p_2=`0.5` and p_3=`p[[2]]`;
   q_{p_1}=`ci[[1]]`, $q_{0.5}$=`median` and q_{p_3}=`ci[[2]]`
2. `median=NULL`: The parameters are fitted on the lower and upper value of the confidence interval only, formally:
   $k = 2$
   p_1=`p[[1]]`, p_2=`p[[2]]`;
   q_{p_1}=`ci[[1]]`, q_{p_2}=`ci[[2]]`

The `(p[[2]]-p[[1]])` - confidence interval must be symmetric in the sense that `p[[1]] + p[[2]] = 1`.

Value

A list with elements `mean` and `sd`, i.e. the parameters of the underlying normal distribution.

See Also

`tnorm`, `constrOptim`

`paramtnormci_numeric` *Return parameters of truncated normal distribution based on a confidence interval.*

Description

This function calculates the distribution parameters, i.e. `mean` and `sd`, of a truncated normal distribution from an arbitrary confidence interval.

Usage

```
paramtnormci_numeric(p, ci, lowerTrunc = -Inf, upperTrunc = Inf,
  relativeTolerance = 0.05, rootMethod = "probability", ...)
```

Arguments

`p`	`numeric` 2-dimensional vector; probabilities of lower and upper bound of the corresponding confidence interval.
`ci`	`numeric` 2-dimensional vector; lower, i.e `ci[[1]]`, and upper bound, i.e `ci[[2]]`, of the confidence interval.
`lowerTrunc`	`numeric`; lower truncation point of the distribution (>= `-Inf`).
`upperTrunc`	`numeric`; upper truncation point of the distribution (<= `Inf`).
`relativeTolerance`	`numeric`; the relative tolerance level of deviation of the generated confidence interval from the specified interval. If this deviation is greater than `relativeTolerance` a warning is given.
`rootMethod`	`character`; if `="probability"` the equation defining the parameters `mean` and `sd` is the difference between calculated and given probabilities of the confidence interval; if `="quantile"` the equation defining the parameters is the difference between calculated and given upper and lower value of the confidence interval.
`...`	Further parameters passed to `nleqslv`.

Details

For details of the truncated normal distribution see `tnorm`.

Value

A list with elements `mean` and `sd`, i.e. the parameters of the underlying normal distribution.

See Also

`tnorm`, `nleqslv`

`print.mcSimulation` *Print Basic Results from Monte Carlo Simulation.*

Description

This function prints basic results from Monte Carlo simulation and returns it invisible.

Usage

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

Arguments

`x`	An object of class `mcSimulation`.
`...`	Further arguments #ToDo

See Also

`mcSimulation`

`print.summary.eviSimulation`
Print the Summarized EVI Simulation Results.

Description

This function prints the summary of of `eviSimulation` obtained by `summary.eviSimulation`.

Usage

```
## S3 method for class 'summary.eviSimulation'
print(x, ...)
```

Arguments

`x`	An object of class `summary.eviSimulation`.
`...`	Further arguments #ToDo

See Also

`eviSimulation`

`print.summary.mcSimulation`
Print the Summary of a Monte Carlo Simulation.

Description

This function prints the summary of of `mcSimulation` obtained by `summary.mcSimulation`.

Usage

```
## S3 method for class 'summary.mcSimulation'
print(x, ...)
```

Arguments

x An object of class mcSimulation.

... Further arguments #ToDo

See Also

mcSimulation, summary.mcSimulation

---

print.summary.welfareDecisionAnalysis

Print the Summarized Decsion Analysis Results..

Description

This function prints the summary of of welfareDecisionAnalysis obtained by summary.welfareDecisionAnalysis.

Usage

```
## S3 method for class 'summary.welfareDecisionAnalysis'
print(x, ...)
```

Arguments

x An object of class summary.welfareDecisionAnalysis.

... Further arguments #ToDo

See Also

welfareDecisionAnalysis

---

random

Quantiles based generic random number generation.

Description

This function generates random numbers for parametric distributions, parameters of which are determined by given quantiles.

The default method generates univariate random numbers specified by arbitrary quantiles.

Usage

```
random(rho, n, method, relativeTolerance, ...)

## Default S3 method:
random(rho = list(distribution = "norm", probabilities =
  c(0.05, 0.95), quantiles = c(-qnorm(0.95), qnorm(0.95))), n, method = "fit",
  relativeTolerance = 0.05, ...)
```

Arguments

- `rho` — `list` Distribution to be randomly sampled. The list elements are `distribution`, `probabilities` and `quantiles`. For details cf. below.
- `n` — `integer` Number of observations to be generated
- `method` — `character` Particular method to be used for random number generation. Currently only method `rdistq_fit{fit}` is implemented which is the default.
- `relativeTolerance` — `numeric`; the relative tolerance level of deviation of the generated confidence interval from the specified interval. If this deviation is greater than `relativeTolerance` a warning is given.
- `...` — Optional arguments to be passed to the particular random number generating function, i.e. `rdistq_fit`.

Details

The distribution family is determined by `rho[["distribution"]]`. For the possibilities cf. `rdistq_fit`.

`rho[["probabilities"]]` and `[[rho"quantiles"]]` are numeric vectors of the same length. The first defines the probabilites of the quantiles, the second defines the quantiles values which determine the parametric distribution.

Value

A numeric vector of length `n` containing the generated random numbers.

Methods (by class)

- `default`: Univariate random number generation.

See Also

`rdistq_fit`

Examples

```
x<-random(n=10000)
 hist(x,breaks=100)
 mean(x)
 sd(x)

 rho<-list(distribution="norm",
           probabilities=c(0.05,0.4,0.8),
           quantiles=c(-4, 20, 100))
 x<-random(rho=rho, n=10000, tolConv=0.01)
 hist(x,breaks=100)
 quantile(x,p=rho[["probabilities"]])
```

random.estimate *Generate Random Numbers for an Estimate.*

Description

This function generates random numbers for general multivariate distributions that are defined as an estimate.

Usage

```
## S3 method for class 'estimate'
random(rho, n, method = "calculate",
  relativeTolerance = 0.05, ...)
```

Arguments

rho	estimate object; Multivariate distribution to be randomly sampled.
n	Number of generated observations
method	Particular method to be used for random number generation.
relativeTolerance	numeric; the relative tolerance level of deviation of the generated confidence interval from the specified interval. If this deviation is greater than relativeTolerance a warning is given.
...	Optional arguments to be passed to the particular random number generating function.

Details

Generation of uncorrelated components: Implementation: random.estimate1d

Generation of correlated components: Implementation: rmvnorm90ci_exact

See Also

estimate, random.estimate1d, random

Examples

```
variable=c("revenue","costs")
distribution=c("norm","norm")
 lower=c(10000,  5000)
 upper=c(100000, 50000)
 estimateObject<-estimate(variable, distribution, lower, upper)
 x<-random(rho=estimateObject, n=10000)
 apply(X=x, MARGIN=2, FUN=quantile, probs=c(0.05, 0.95))
 cor(x)
 colnames(x)
```

```
summary(x)
hist(x[,"revenue"])
hist(x[,"costs"])
```

`random.estimate1d` *Generate univariate random numbers defined by a 1-d estimate.*

Description

This function generates random numbers for univariate parametric distributions, which parameters are determined by a one dimensional estimate (`estimate1d`).

Usage

```
## S3 method for class 'estimate1d'
random(rho, n, method = "calculate",
  relativeTolerance = 0.05, ...)
```

Arguments

`rho`	`estimate1d`: Univariate distribution to be randomly sampled.
`n`	`integer`: Number of observations to be generated
`method`	`character`: Particular method to be used for random number generation. It can be either `"calculate"` (the default) or `"fit"`. Details below.
`relativeTolerance`	`numeric`: the relative tolerance level of deviation of the generated confidence interval from the specified interval. If this deviation is greater than `relativeTolerance` a warning is given.
`...`	Optional arguments to be passed to the particular random number generating function (cf. below).

Details

`rho[["distribution"]]`**:** The follwing table shows the available distributions and the implemented generation method:

`rho[["distribution"]]`	**Distribution Name**	`method`
`"const"`	Deterministic case	not applicable
`"norm"`	Normal	`calculate`, `fit`
`"posnorm"`	Positive normal	`calculate`, `fit`
`"tnorm_0_1"`	0-1-truncated normal	`calculate`, `fit`
`"beta"`	Beta	`fit`
`"cauchy"`	Cauchy	`fit`
`"logis"`	Logistic	`fit`
`"t"`	Student t	`fit`
`"chisq"`	Central Chi-Squared	`fit`
`"chisqnc"`	Non-central Chi-Squared	`fit`

"exp"	Exponential	fit
"f"	Central F	fit
"gamma"	Gamma with scale=1/rate	fit
"lnorm"	Log Normal	calculate, fit
"unif"	Uniform	calculate, fit
"weibull"	Weibull	fit
"triang"	Triangular	fit
"gompertz"	Gompertz	fit
"pert"	(Modified) PERT	fit

For `distribution="const"` the argument `method` is obsolete, as a constant is neither fitted nor calculated.

`rho[["method"]]` If supplied, i.e. `!is.null(rho[["method"]])`, this value overwrites the function argument `method`.

`method` This parameter defines, how the parameters of the distribution to be sample are derived from `rho[["lower"]]`, `rho[["upper"]]` and possibly `rho[["median"]]`. Possibilities are `"calculate"` (the default) or `"fit"`:

`method="calculate"` The parameters are calculated if possible using the exact (analytical) formula or, otherwise, numerically. This calculation of the distribution parameters is independent of `rho[["median"]]` being supplied or not. For the implemented distributions, it only depends on `rho[["lower"]]` and `rho[["upper"]]`. However, if it is supplied, i.e. `is.numeric(rho[["median"]])`, a check is performed, if the relative deviation of the generated median from`rho[["median"]]` is greater than `relativeTolerance`. In this case a warning is given.

`method="fit"` The parameters are obtained by fitting the distribution on the supplied quantiles. Given that `rho[["median"]]==NULL` the distribution is fitted only to `lower` and `upper` and a warning is given; due to the used numerical procedure, the calculated parameters might define a distribution which strongly deviates from the intended one. There is larger control on the shape of the distribution to be generated by supplying the estimate of the median. If `is.numeric(rho[["median"]])` the distribution is fitted to `lower`, `upper` and `median`.

`...` For passing further parameters to the function which generates the random numbers, cf. the above table and follow the link in the column `method`.

See Also

`estimate1d`; For `method="calculate"`: `rdist90ci_exact`; for `method="fit"`: `rdistq_fit`; for both methods: `rposnorm90ci` and `rtnorm_0_1_90ci`. For the default method: `random`.

Examples

```
# Generate log normal distributed random numbers:
x<-random(estimate1d("lnorm",50,100), n=100000)
quantile(x, probs=c(0.05, 0.95))
hist(x, breaks=100)
```

`rdist90ci_exact` *90%-confidence interval based univariate random number generation (by exact parameter calculation).*

Description

This function generates random numbers for a set of univariate parametric distributions from given 90% confidence interval. Internally, this is achieved by exact, i.e. analytic, calculation of the parameters for the individual distribution from the given 90% confidence interval.

Usage

```
rdist90ci_exact(distribution, n, lower, upper)
```

Arguments

`distribution`	`character`; A character string that defines the univariate distribution to be randomly sampled. For possible options cf. section Details.
`n`	Number of generated observations.
`lower`	`numeric`; lower bound of the 90% confidence intervall.
`upper`	`numeric`; upper bound of the 90% confidence intervall.

Details

The follwing table shows the available distributions and their identification (option: `distribution`) as a character string:

`distribution`	**Distribution Name**	**Requirements**
`"const"`	Deterministic case	`lower == upper`
`"norm"`	Normal	`lower < upper`
`"lnorm"`	Log Normal	`0 < lower < upper`
`"unif"`	Uniform	`lower < upper`

Parameter formulae: We use the notation: l=`lower` and u=`upper`; Φ is the cummulative distribution function of the standard normal distribution and Φ^{-1} its inverse, which is the quantile function of the standard normal distribution.

`distribution="norm"`: The formulae for μ and σ, viz. the mean and standard deviation, respectively, of the normal distribution are $\mu = \frac{l+u}{2}$ and $\sigma = \frac{\mu-l}{\Phi^{-1}(0.95)}$.

`distribution="unif"`: For the minimum a and maximum b of the uniform distribution $U_{[a,b]}$ it holds that $a = l - 0.05(u-l)$ and $b = u + 0.05(u-l)$.

`distribution="lnorm"`: The density of the log normal distribution is $f(x) = \frac{1}{\sqrt{2\pi}\sigma x}\exp(-\frac{(\ln(x)-\mu)^2}{2\sigma^2})$ for $x > 0$ and $f(x) = 0$ otherwise. Its parameters are determined by the confidence interval via $\mu = \frac{\ln(l)+\ln(u)}{2}$ and $\sigma = \frac{1}{\Phi^{-1}(0.95)}(\mu - \ln(l))$. Note the correspondence to the formula for the normal distribution.

Value

A numeric vector of length n with the sampled values according to the chosen distribution.

In case of distribution="const", viz. the deterministic case, the function returns: rep(lower, n).

Examples

```
# Generate uniformly distributed random numbers:
lower=3
upper=6
hist(r<-rdist90ci_exact(distribution="unif", n=10000, lower=lower, upper=upper),breaks=100)
print(quantile(x=r, probs=c(0.05,0.95)))
print(summary(r))

# Generate log normal distributed random numbers:
hist(r<-rdist90ci_exact(distribution="lnorm", n=10000, lower=lower, upper=upper),breaks=100)
print(quantile(x=r, probs=c(0.05,0.95)))
print(summary(r))
```

---

rdistq_fit — *Quantiles based univariate random number generation (by parameter fitting).*

Description

This function generates random numbers for a set of univariate parametric distributions from given quantiles. Internally, this is achieved by fitting the distribution function to the given quantiles.

Usage

```
rdistq_fit(distribution, n, percentiles = c(0.05, 0.5, 0.95), quantiles,
  relativeTolerance = 0.05, tolConv = 0.001, fit.weights = rep(1,
  length(percentiles)), verbosity = 1)
```

Arguments

distribution	A character string that defines the univariate distribution to be randomly sampled.
n	Number of generated observations.
percentiles	Numeric vector giving the percentiles.
quantiles	Numeric vector giving the quantiles.
relativeTolerance	numeric; the relative tolerance level of deviation of the generated individual percentiles from the specified percentiles. If any deviation is greater than relativeTolerance a warning is given.
tolConv	positive numerical value, the absolute convergence tolerance for reaching zero by fitting distributions get.norm.par will be shown.

`fit.weights` numerical vector of the same length as a probabilities vector `p` containing positive values for weighting quantiles. By default all quantiles will be weighted by 1.

`verbosity` `integer`; if `0` the function is silent; the larger the value the more verbose is the output information.

Details

The follwing table shows the available distributions and their identification (option: `distribution`) as a character string:

`distribution`	**Distribution Name**	`length(quantiles)`	**Necessary Package**
"norm"	Normal	>=2	
"beta"	Beta	>=2	
"cauchy"	Cauchy	>=2	
"logis"	Logistic	>=2	
"t"	Student t	>=1	
"chisq"	Central Chi-Squared	>=1	
"chisqnc"	Non-central Chi-Squared	>=2	
"exp"	Exponential	>=1	
"f"	Central F	>=2	
"gamma"	Gamma with `scale=1/rate`	>=2	
"lnorm"	Log Normal	>=2	
"unif"	Uniform	==2	
"weibull"	Weibull	>=2	
"triang"	Triangular	>=3	mc2d
"gompertz"	Gompertz	>=2	eha
"pert"	(Modified) PERT	>=4	mc2d
"tnorm"	Truncated Normal	>=4	msm

`percentiles` and `quantiles` must be of the same length. `percentiles` must be `>=0` and `<=1`.

The default for `percentiles` is 0.05, 0.5 and 0.95, so for the default, the quantiles argument should be a vector with 3 elements. If this is to be longer, the percentiles argument has to be adjusted to match the length of quantiles.

The fitting of the distribution parameters is done using `rriskFitdist.perc`.

Value

A numeric vector of length `n` with the sampled values according to the chosen distribution.

See Also

`rriskFitdist.perc`

Examples

```
# Fit a log normal distribution to 3 quantiles:
percentiles<-c(0.05, 0.5, 0.95)
```

```
quantiles=c(1,3,15)
hist(r<-rdistq_fit(distribution="lnorm", n=10000, quantiles=quantiles),breaks=100)
print(quantile(x=r, probs=percentiles))
```

`rmvnorm90ci_exact` *90%-confidence interval multivariate normal random number generation.*

Description

This function generates normal distributed multivariate random numbers which parameters are determined by the 90%-confidence interval. The calculation of `mean` and `sd` is exact.

Usage

```
rmvnorm90ci_exact(n, lower, upper, correlationMatrix)
```

Arguments

`n`	`integer` Number of observations to be generated.
`lower`	`numeric` vector; lower bound of the 90% confidence intervall.
`upper`	`numeric` vector; upper bound of the 90% confidence intervall.
`correlationMatrix`	`numeric` symmetric matrix which is the correlation matrix of the multivariate normal distribution. In particular, all diagonal elements must be equal to 1.

See Also

`random`, `rmvnorm`

`row.names.estimate` *Return the variable names of an* `estimate` *object.*

Description

This function returns the variable names of an `estimate` object which is identical to `row.names(x$base)`.

Usage

```
## S3 method for class 'estimate'
row.names(x)
```

Arguments

`x`	an `estimate` object.

See Also

`estimate`, `names.estimate`, `corMat.estimate` rtnorm90ci *90%-confidence interval based truncated normal random number generation.*

Description

rtnorm90ci generates truncated normal random numbers based on the 90% confidence interval calculating the distribution parameter numerically from the 90%-confidence interval or via a fit on the 90%-confidence interval. The fit might include the median or not.

rposnorm90ci generates positive normal random numbers based on the 90% confidence interval. It is a wrapper function for rtnorm90ci.

rtnorm_0_1_90ci generates normal random numbers truncated to $[0, 1]$ based on the 90% confidence interval. It is a wrapper function for rtnorm90ci.

Usage

```
rtnorm90ci(n, ci, median = mean(ci), lowerTrunc = -Inf, upperTrunc = Inf,
  method = "numeric", relativeTolerance = 0.05, ...)

rposnorm90ci(n, lower, median = mean(c(lower, upper)), upper,
  method = "numeric", relativeTolerance = 0.05, ...)

rtnorm_0_1_90ci(n, lower, median = mean(c(lower, upper)), upper,
  method = "numeric", relativeTolerance = 0.05, ...)
```

Arguments

n	Number of generated observations.
ci	numeric 2-dimensional vector; lower, i.e ci[[1]], and upper bound, i.e ci[[2]], of the 90%-confidence interval.
median	if NULL: truncated normal is fitted only to lower and upper value of the confidence interval; if numeric: truncated normal is fitted on the confidence interval and the median simultaneously. For details cf. below. This option is only relevant if method="fit".
lowerTrunc	numeric; lower truncation point of the distribution (>= -Inf).
upperTrunc	numeric; upper truncation point of the distribution (<= Inf).
method	method used to determine the parameters of the truncated normal; possible methods are "numeric" (the default) and "fit".
relativeTolerance	numeric; the relative tolerance level of deviation of the generated confidence interval from the specified interval. If this deviation is greater than relativeTolerance a warning is given.
...	further parameters to be passed to paramtnormci_numeric or paramtnormci_fit, respectively.
lower	numeric; lower bound of the 90% confidence intervall.
upper	numeric; upper bound of the 90% confidence intervall.

Details

method="numeric" is implemented by paramtnormci_numeric and method="fit" by paramtnormci_fit.

Positive normal random number generation: a positive normal distribution is a truncated normal distribution with lower truncation point equal to zero and upper truncation is infinity. rposnorm90ci implements this as a wrapper function for rtnorm90ci(n, c(lower,upper), median, lowerTrunc=0, upperTrunc=Inf,

0-1-(truncated) normal random number generation: a 0-1-normal distribution is a truncated normal distribution with lower truncation point equal to zero and upper truncation equal to 1. rtnorm_0_1_90ci implements this as a wrapper function for rtnorm90ci(n, c(lower,upper), median, lowerTrunc=0, upperTrunc=1, m

See Also

For the implementation of method="numeric": paramtnormci_numeric; for the implementation of method="fit": paramtnormci_fit.

---

sort.summary.eviSimulation *Sort Summarized EVI Simulation Results..*

Description

Sort summarized EVI simulation results according to their EVI.

Usage

```
## S3 method for class 'summary.eviSimulation'
sort(x, decreasing = TRUE, ...,
  along = row.names(x$summary$evi)[[1]])
```

Arguments

x	An object of class summary.eviSimulation.
decreasing	logical; if the evi should be sorted in decreasing order.
...	Further arguments #ToDo
along	character; the name of the valuation variable along which evi should be sorted.

Value

An object of class summary.eviSimulation.

See Also

eviSimulation, summary.eviSimulation, sort

summary.eviSimulation *Summarize EVI Simulation Results*

Description

summary.eviSimulation produces result summaries of the results of Expected Value of Information (EVI) simulation obtained by the function `eviSimulation`.

Usage

```
## S3 method for class 'eviSimulation'
summary(object, ..., digits = max(3,
  getOption("digits") - 3))
```

Arguments

`object`	An object of class `eviSimulation`.
`...`	Further arguments passed to `summary.welfareDecisionAnalysis`.
`digits`	how many significant digits are to be used for numeric and complex `x`. The default, `NULL`, uses `getOption("digits")`. This is a suggestion: enough decimal places will be used so that the smallest (in magnitude) number has this many significant digits, and also to satisfy `nsmall`. (For the interpretation for complex numbers see `signif`.)

Value

An object of class `summary.eviSimulation`.

See Also

`eviSimulation`, `print.summary.eviSimulation`, `summary.welfareDecisionAnalysis`

summary.mcSimulation *Summarize Results from Monte Carlo Simulation.*

Description

A summary of the results of a Monte Carlo simulation obtained by the function `mcSimulation` is produced.

Usage

```
## S3 method for class 'mcSimulation'
summary(object, ..., digits = max(3,
  getOption("digits") - 3), variables.y = names(object$y), variables.x = if
  (classicView) names(object$x), classicView = FALSE, probs = c(0, 0.1,
  0.25, 0.5, 0.75, 0.9, 1))
```

Arguments

`object`	An object of class `mcSimulation`.
`...`	Further arguments passed to `summary.data.frame` (`classicView=TRUE`) or `format` (`classicView=FALSE`).
`digits`	how many significant digits are to be used for numeric and complex `x`. The default, `NULL`, uses `getOption("digits")`. This is a suggestion: enough decimal places will be used so that the smallest (in magnitude) number has this many significant digits, and also to satisfy `nsmall`. (For the interpretation for complex numbers see `signif`.)
`variables.y`	`character` or `character` vector; Names of the components of the simulation function (`model_function`) which results shall be displayed. Defaults to all components.
`variables.x`	`character` or `character` vector; Names of the components of the input variables to the simulation function, i.e. the names of the variables in the input `estimate` which random sampling results shall be displayed. Defaults to all components.
`classicView`	`logical`; if `TRUE` the results are summarized using summary.data.frame, if `FALSE` further output is produced and the quantile information can be chosen. Cf. section Value and argument `probs`. Default is `FALSE`.
`probs`	`numeric` vector of quantiles that shall be displayed if `classicView=FALSE`.

Value

An object of class `summary.mcSimulation`.

`chance_loss`
`chance_zero`
`chance_gain`

See Also

`mcSimulation`, `print.summary.mcSimulation`, `summary.data.frame`

`summary.welfareDecisionAnalysis`

Summarize Decsion Analysis Results.

Description

summary.welfareDecisionAnalysis produces result summaries of the results of decision analysis simulation obtained by the function `welfareDecisionAnalysis`.

Usage

```
## S3 method for class 'welfareDecisionAnalysis'
summary(object, ..., digits = max(3,
  getOption("digits") - 3))
```

Arguments

`object`	An object of class `welfareDecisionAnalysis`.
`...`	Further arguments passed to `format`.
`digits`	how many significant digits are to be used for numeric and complex `x`. The default, `NULL`, uses `getOption("digits")`. This is a suggestion: enough decimal places will be used so that the smallest (in magnitude) number has this many significant digits, and also to satisfy `nsmall`. (For the interpretation for complex numbers see `signif`.)

Value

An object of class `summary.welfareDecisionAnalysis`.

See Also

`welfareDecisionAnalysis`, `print.summary.welfareDecisionAnalysis`, `format`

`uncertaintyAnalysis` *Uncertainty Analysis Wrapper Function.*

Description

This function performs a Monte Carlo simulation from input files and analyses the results via Partial Least Squares Regression (PLSR) and calculates the Variable Importance on Projection (VIP). Results are safed as plots.

Usage

```
uncertaintyAnalysis(inputFilePath, outputPath, modelFunction,
  numberOfSimulations, randomMethod = "calculate",
  functionSyntax = "globalNames", write_table = TRUE, indicators = FALSE,
  log_scales = FALSE, oldInputStandard = FALSE, verbosity = 1)
```

Arguments

`inputFilePath`	Path to input csv file, which gives the input `estimate`.
`outputPath`	Path were the result plots and tables are safed.
`modelFunction`	The model function.
`numberOfSimulations`	The number of Monte Carlo simulations to be performed.

randomMethod	ToDo
functionSyntax	ToDo
write_table	logical; If the full Monte Carlo simulation results and PLSR results should be written to file.
indicators	logical; If indicator variables should be respected specially.
log_scales	logical; If the scales in the pls plots should be logarithmic.
oldInputStandard	logical; If the old input standard should be used (estimate_read_csv_old).
verbosity	integer; if 0 the function is silent; the larger the value the more verbose is output information.

See Also

mcSimulation, estimate, estimate_read_csv

---

welfareDecisionAnalysis

Analysis of the Underlying Welfare Based Decision Problem

Description

The optimal choice between two different opportunities is calculated. This decision is based on minimizing the Expected Net Loss (ENL).

Usage

```
welfareDecisionAnalysis(estimate, model, numberOfSimulations,
  functionSyntax = "data.frameNames")
```

Arguments

estimate	estimate object describing the distribution of the input variables.
model	either a function or a list with two functions: list(p1,p2). In the first case the function is the net benefit of project approval vs. the status quo. In the second case the element p1 is the function valuing the first project and the element p2 valueing the second project.
numberOfSimulations	integer; number of simulations to be used in the underlying Monte Carlo analysis
functionSyntax	function character; function syntax used in the model function(s).

Details

This principle is along the line described in Hubbard (2014). The Expected Opportunity Loss (EOL) is defined as the Expected Net Loss (ENL) for the best decision. The best decision minimises the ENL. The EOL is always conditional on the available information (I): EOL=EOL(I). Here, the available information is the supplied estimate. One can show that in the case of two alternatives, minimization of EOL is equivalent to maximization of the Expected Net Benefit.

Value

An object of class `welfareDecisionAnalysis` with the following elements:

`enbPa`	Expected Net Loss (ENL) in case of project approval (PA)
`enbSq`	Expected Net Loss (ENL) in case of status quo (SQ)
`eol`	Expected Oportunity Loss (EOL)
`optimalChoice`	The optimal choice, i.e. either project approval (PA) or the status quo (SQ)

See Also

`mcSimulation`, `estimate`, `summary.welfareDecisionAnalysis`

Examples

```
#############################################################
# Example 1 (Creating the estimate from the command line):
#############################################################
# Create the estimate object:
variable=c("revenue","costs")
distribution=c("posnorm","posnorm")
lower=c(10000,  5000)
upper=c(100000, 50000)
costBenefitEstimate<-estimate(variable, distribution, lower, upper)
# (a) Define the model function without name for the return value:
profit<-function(x){
 x$revenue-x$costs
}
# Perform the decision analysis:
myAnalysis<-welfareDecisionAnalysis(estimate=costBenefitEstimate,
                                    model=profit,
                                    numberOfSimulations=100000,
                                    functionSyntax="data.frameNames")
# Show the analysis results:
print(summary((myAnalysis)))
#############################################################
# (b) Define the model function with a name for the return value:
profit<-function(x){
 list(Profit=x$revenue-x$costs)
}
# Perform the decision analysis:
myAnalysis<-welfareDecisionAnalysis(estimate=costBenefitEstimate,
                                    model=profit,
                                    numberOfSimulations=100000,
                                    functionSyntax="data.frameNames")
# Show the analysis results:
print(summary((myAnalysis)))
#############################################################
# (c) Two decsion variables:
decisionModel<-function(x){
 list(Profit=x$revenue-x$costs,
   Costs=-x$costs)
}
```

```
# Perform the decision analysis:
myAnalysis<-welfareDecisionAnalysis(estimate=costBenefitEstimate,
                                    model=decisionModel,
                                    numberOfSimulations=100000,
                                    functionSyntax="data.frameNames")
# Show the analysis results:
print(summary((myAnalysis)))
```

Index

(Modified) PERT, *32*, *35*
0-1-truncated normal, *31*

as.data.frame, *3*
as.data.frame.mcSimulation, 3
as.estimate1d *(*estimate1d*)*, 8

Beta, *31*, *35*

calculate, *31, 32*
Cauchy, *31*, *35*
Central Chi-Squared, *31*, *35*
Central F, *32*, *35*
colors, *19*
constrOptim, *24, 25*
corMat, 4, *7*
corMat.estimate, 4, *24*, *36*

data.frame, *7*
decisionSupport, 5
decisionSupport-package
    *(*decisionSupport*)*, 5

estimate, *4*, *6*, 7, *9–14*, *18–20*, *22–24*, *30*, *36*, *41–43*
estimate1d, *7*, 8, *31, 32*
estimate_read_csv, *7*, 9, *12, 13*, *42*
estimate_read_csv_old, 11, *42*
estimate_write_csv, *7*, *10*, 12
eviSimulation, *5, 6*, 13, *20*, *27*, *38, 39*
Exponential, *32*, *35*

fit, *31, 32*
format, *40, 41*

Gamma, *32*, *35*
getOption, *39–41*
globalNames2data.frameNames, 17
Gompertz, *32*, *35*

hist, *19*
hist.mcSimulation, 18, *22*

individualEvpiSimulation, *5*, 19

Log Normal, *32, 33*, *35*
Logistic, *31*, *35*

make.names, *3*
mcSimulation, *6*, *14*, *18–20*, 21, *27, 28*, *39, 40*, *42, 43*

names.estimate, *4*, *7*, 23, *36*
nleqslv, *26*
Non-central Chi-Squared, *31*, *35*
Normal, *31*, *33*, *35*

paramtnormci_fit, 24, *37, 38*
paramtnormci_numeric, 25, *37, 38*
Positive normal, *31*
pretty, *18*
print.mcSimulation, *22*, 26
print.summary.eviSimulation, 27, *39*
print.summary.mcSimulation, 27, *40*
print.summary.welfareDecisionAnalysis, 28, *41*

random, 28, *30*, *32*, *36*
random.estimate, *6, 7*, *22*, 30
random.estimate1d, *9*, *30*, 31
rdist90ci_exact, *32*, 33
rdistq_fit, *29*, *32*, 34
read.csv, *10–12*
rmvnorm, *36*
rmvnorm90ci_exact, *30*, 36
row.names, *7*
row.names.estimate, *4*, *7*, *24*, 36
rposnorm90ci, *32*
rposnorm90ci *(*rtnorm90ci*)*, 37
rriskFitdist.perc, *35*
rtnorm90ci, 37
rtnorm_0_1_90ci, *32*

rtnorm_0_1_90ci (rtnorm90ci), 37

scan, *9*, *11*
signif, *39–41*
sort, *38*
sort.summary.eviSimulation, 38
Student t, *31*, *35*
summary.data.frame, *40*
summary.eviSimulation, *27*, *38*, 39
summary.mcSimulation, *22*, *27, 28*, 39
summary.welfareDecisionAnalysis, *28*, *39*, 40, *43*

tnorm, *25, 26*
Triangular, *32*, *35*
Truncated Normal, *35*

uncertaintyAnalysis, *6*, 41
Uniform, *32, 33*, *35*

Weibull, *32*, *35*
welfareDecisionAnalysis, *5, 6*, *13, 14*, *20*, *28*, *40, 41*, 42
write.csv, *12, 13*