Package ‘decisionSupport’

March 24, 2015

Type Package

Title Quantitative Support of Decision Making under Uncertainty

Version 1.100.0.9000

Date 2015-03-24

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),
nleqslv (>= 2.6),
rriskDistributions (>= 2.0)

Suggests testthat (>= 0.9.1)

URL http://www.worldagroforestry.org/

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

R topics documented:

`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

x	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 (C)

License

Author(s)

Lutz G<c3><b6>hring <lutz.goehring@gmx.de>, Eike Luedeling (ICRAF) <E.Luedeling@cgiar.org>

Maintainer: Lutz G<c3><b6>hring <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 an Estimate Object*

Description

This function creates an object of class `estimate`. #ToDo: detailed description #ToDo: Implement characterization of distribution by mean and sd. Eventually, also by other quantiles.

Usage

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

Arguments

`...`	arguments that can be coerced to a data frame comprising the base of the estimate.
`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 type `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. `correlation_matrix` is a symmetric matrix with row and column names being the subset of the 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`

`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

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

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	character. Ouput 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.
...	Further parameters to be passed to write.csv.
estimate	Estimate object to write to file fileName.

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.
`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).
`prospectiveEstmate`	`estimate` object describing the prospective distribution of the input variables which could hypothetically achieved by collecting more information, viz. improving the measurement.

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")
```

`hist.mcSimulation` *Plot histogram of results of a Monte Carlo Simulation.*

Description

This function plots the histogram 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`.
`...`	Further arguments #ToDo

See Also

`mcSimulation`, `hist`

`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`

---

paramposnorm90ci — *Return parameters of positive normal random distribution based on the 90%-confidence interval.*

Description

This function calculates the distribution parameters from the 90%-confidence interval.

Usage

```
paramposnorm90ci(lower, upper, relativeTolerance = 0.05, method = "numeric")
```

Arguments

lower	`numeric`; lower bound of the 90% confidence intervall.
upper	`numeric`; upper bound of the 90% confidence intervall.
relativeTolerance	`numeric`; the relative tolerance level of deviation of the generated confidence interval from the specified interval.
method	The method to calculate the parameters. Default is `"numeric"`.

Details

#ToDo

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

Description

This function calculates the distribution parameters of a truncated normal distribution from an arbitrary confidence interval.

Usage

```
paramtnormci(p, ci, lowerTrunc = -Inf, upperTrunc = Inf,
  relativeTolerance = 0.05, method = "numeric")
```

Arguments

`p`	numeric vector; probabilities of upper and lower bound of the corresponding confidence interval.
`ci`	`numeric` vector; lower and upper bound of the confidence interval.
`lowerTrunc`	`numeric`; lower truncation point of the distribution.
`upperTrunc`	`numeric`; upper truncation point of the distribution.
`relativeTolerance`	`numeric`; the relative tolerance level of deviation of the generated confidence interval from the specified interval.
`method`	The method to calculate the parameters. Default is `"numeric"`.

Details

#ToDo

`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

---

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

---

r0_1norm90ci_numeric

Generate normal random numbers truncated to $[0, 1]$ based on the 90%-confidence interval.

Description

This function generatesnormal random numbers truncated to $[0, 1]$ based on the 90% confidence interval calculating the distribution parameter numerically from the 90%-confidence interval.

Usage

```
r0_1norm90ci_numeric(n, lower, upper, relativeTolerance = 0.05)
```

Arguments

`n`	Number of generated observations.
`lower`	`numeric`; lower bound of the 90% confidence intervall.
`upper`	`numeric`; upper bound of the 90% confidence intervall.
`relativeTolerance`	`numeric`; the relative tolerance level of deviation of the generated confidence interval from the specified interval.

Details

#ToDo

`random` *Generate random numbers for a certain probability distribution.*

Description

This function generates multivariate random numbers for general multivariate distributions.

Usage

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

Arguments

`rho`	Distribution to be randomly sampled.
`n`	Number of generated observations.
`method`	Particular method to be used for random number generation.
`...`	Optional arguments to be passed to the particular random number generating function.

`random.default` *Generate random numbers based on the first two moments of a certain probability distribution.*

Description

This function generates random numbers for general multivariate distributions that can be characterized by the joint first two moments, viz. the mean and covariance.

Usage

```
## Default S3 method:
random(rho = list(distribution_type, mean, sd), n, method,
  ...)
```

Arguments

`rho`	list; Distribution to be randomly sampled.
`n`	Number of generated observations
`method`	Particular method to be used for random number generation.
`...`	Optional arguments to be passed to the particular random number generating function.

`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", ...)
```

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.
`...`	Optional arguments to be passed to the particular random number generating function.

Details

Generation of uncorrelated components: Implementation: `random_estimate_1d`

Generation of correlated components: Implementation: `rmvnorm90ci_exact`

See Also

`estimate`

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_estimate_1d — *Generate univariate random numbers based on an estimate.*

Description

This function generates random numbers for general univariate distributions.

Usage

```
random_estimate_1d(rho, n, method = "calculate", ...)
```

Arguments

`rho`	`estimate` object; Univariate distribution to be randomly sampled.
`n`	Number of generated observations
`method`	Particular method to be used for random number generation.
`...`	Optional arguments to be passed to the particular random number generating function.

Details

`method` can be either `"calculate"` (the default) or `"fit"`.

The follwing table shows the available distributions and the implemented generation method:

Identification	**Distribution**	`method`
`const`	ToDo	`calculate`
`norm`	Normal distribution	`calculate`, `fit`
`posnorm`	ToDo	`calculate`
`0_1norm`	ToDo	`calculate`
`beta`	Beta distribution	`fit`
`cauchy`	ToDo	`fit`
`logis`	ToDo	`fit`
`t`	ToDo	`fit`

chisq	ToDo	fit
chisqnc	ToDo: implement?	fit
exp	ToDo	fit
f	ToDo	fit
gamma	ToDo	fit
lnorm	ToDo	fit
unif	ToDo	calculate, fit
weibull	ToDo	fit
triang	ToDo	fit
gompertz	ToDo	fit
pert	ToDo	fit
tnorm	Truncated normal distribution	fit

See Also

For method="calculate": rdist90ci_exact, rposnorm90ci_numeric and r0_1norm90ci_numeric; for method="fit": rdistq_fit

---

rdist90ci_exact *Generate univariate random numbers based on the 90%-confidence interval.*

Description

This function generates random numbers for general univariate distributions based on the 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.
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 as a character string:

Distribution encoding	Distribution
const	ToDo
norm	ToDo
pos_norm	ToDo

norm_0_1	ToDo
pois	ToDo
binom	ToDo
unif	ToDo
lnorm	ToDo
lnorm_lim2	ToDo

---

rdistq_fit *Generate univariate random numbers based on quantiles.*

Description

This function generates random numbers for a set of univariate distributions based on the distribution quantiles. Internally, this is achieved by fitting the distribution function to the given quantiles using rriskFitdist.perc.

Usage

```
rdistq_fit(distribution, n, percentiles = c(0.05, 0.5, 0.95), quantiles)
```

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.

Details

The follwing table shows the available distributions and their identification as a character string:

Identification	Distribution	Number of quantiles
norm	Normal distribution	>=2
beta	Beta distribution	ToDo
cauchy	ToDo	ToDo
logis	ToDo	ToDo
t	ToDo	ToDo
chisq	ToDo	ToDo
chisqnc	ToDo: implement?	ToDo
exp	ToDo	ToDo
f	ToDo	ToDo
gamma	ToDo	ToDo
lnorm	ToDo	ToDo
unif	ToDo	ToDo
weibull	ToDo	ToDo

triang	ToDo	ToDo
gompertz	ToDo	ToDo
pert	ToDo	ToDo
tnorm	Truncated normal distribution	ToDo

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.

Value

ToDo

`rmvnorm90ci_exact` *Generate normal distributed multivariate random numbers based on the 90%-confidence interval.*

Description

This function generates normal distributed multivariate random numbers based on the 90%-confidence interval.

Usage

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

Arguments

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

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`

rposnorm90ci_iter — *Generate positive normal random numbers based on the 90%-confidence interval.*

Description

This function generates positive normal random numbers based on the 90% confidence interval using an iteration algorithm.

Usage

```
rposnorm90ci_iter(n, lower, upper, relativeTolerance = 0.05, maxIter = 40)
```

Arguments

`n`	Number of generated observations.
`lower`	`numeric`; lower bound of the 90% confidence intervall.
`upper`	`numeric`; upper bound of the 90% confidence intervall.
`relativeTolerance`	`numeric`; the relative tolerance level of deviation of the generated confidence interval from the specified interval.
`maxIter`	`numeric`; maximum number of iterations.

Details

The generation of random numbers is repeated until the generated 90% - confidence interval is close enough to the desired value.

`rposnorm90ci_numeric` *Generate positive normal random numbers based on the 90%-confidence interval.*

Description

This function generates positive normal random numbers based on the 90% confidence interval calculating the distribution parameter numerically from the 90%-confidence interval.

Usage

```
rposnorm90ci_numeric(n, lower, upper, relativeTolerance = 0.05)
```

Arguments

`n`	Number of generated observations.
`lower`	`numeric`; lower bound of the 90% confidence intervall.
`upper`	`numeric`; upper bound of the 90% confidence intervall.
`relativeTolerance`	`numeric`; the relative tolerance level of deviation of the generated confidence interval from the specified interval.

Details

#ToDo

`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 #ToDo

Value

An object of class `summary.eviSimulation`.

See Also

`eviSimulation`, `print.summary.eviSimulation`

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

Description

summary.mcSimulation produces result summaries of the results of a Monte Carlo simulation obtained by the function mcSimulation.

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 #ToDo

Value

An object of class summary.mcSimulation.

See Also

mcSimulation, print.summary.mcSimulation

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 #ToDo

Value

An object of class `summary.welfareDecisionAnalysis`.

See Also

`welfareDecisionAnalysis`, `print.summary.welfareDecisionAnalysis`

`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(inputFileName, outputDirectory, modelFunction,
  NumberofSimulations, randomMethod = "calculate",
  functionSyntax = "globalNames", write_table = TRUE, indicators = FALSE,
  log_scales = FALSE, oldInputStandard = FALSE)
```

Arguments

`inputFileName`	Path to input csv file, which gives the input `estimate`.
`outputDirectory`	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`).

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

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