Package 'rdecision'

May 21, 2020

Title Decision Analytic Modelling in Health Economics

Version 0.1.7

Description Classes and functions for modelling healthcare interventions using cohort models (decision trees, Markov models and extended Markov models). It draws on terminology and examples from Briggs, Claxton and Sculpher, ``Decision Modelling for Health Economic Evaluation", Oxford University Press, 2006.

Depends R (>= 3.1.0)

Imports R6,
rlang,
stats,
utils

Suggests rmarkdown,
knitr,
testthat

License GPL-3

LazyData true

Encoding UTF-8

RoxygenNote 7.1.0

VignetteBuilder knitr, rmarkdown

R topics documented:

`BetaModelVariable` *BetaModelVariable*

Description

An R6 class for a model variable with Beta function uncertainty

Details

A model variable for which the uncertainty in the point estimate can be modelled with a Beta distribution. The hyperparameters of the distribution are the shape ('alpha') and the shape ('beta') of the uncertainty distribution.

Super class

`rdecision::ModelVariable` -> `BetaModelVariable`

Methods

Public methods:

- `BetaModelVariable$new()`
- `BetaModelVariable$sample()`
- `BetaModelVariable$getDistribution()`
- `BetaModelVariable$getMean()`
- `BetaModelVariable$getSD()`
- `BetaModelVariable$getQuantile()`
- `BetaModelVariable$clone()`

Method `new()`**:** Create an object of class BetaModelVariable.

Usage:

```
BetaModelVariable$new(description, units, alpha, beta)
```

Arguments:

`description` A character string describing the variable.

`units` Units of the variable, as character string.

`alpha` parameter of the Beta distribution.

`beta` parameter of the Beta distribution.

`label` A character string label for the variable. It is advised to make this the same as the variable name which helps when tabulating model variables involving ExpressionModelVariables.

Returns: An object of class BetaModelVariable.

Method `sample()`**:** Set the value of the model variable from its uncertainty distribution. Nothing is returned; the sampled value is returned at the next call to 'value()'.

Usage:

```
BetaModelVariable$sample(expected = F)
```

Arguments:

`expected` Logical; if TRUE sets the value of the model variable returned at subsequent calls to 'value()' to be equal to the expectation of the variable. Default is FALSE.

Returns: Updated BetaModelVariable object.

Method `getDistribution()`**:** Accessor function for the name of the uncertainty distribution.

Usage:

```
BetaModelVariable$getDistribution()
```

Returns: Distribution name as character string.

Method `getMean()`**:** Return the expected value of the distribution.

Usage:

```
BetaModelVariable$getMean()
```

Returns: Expected value as a numeric value.

Method `getSD()`**:** Return the standard deviation of the distribution.

Usage:

```
BetaModelVariable$getSD()
```

Returns: Standard deviation as a numeric value

Method `getQuantile()`**:** Return the quantiles of the Gamma uncertainty distribution.

Usage:

```
BetaModelVariable$getQuantile(probs)
```

Arguments:

`probs` Vector of probabilities, in range [0,1].

Returns: Vector of quantiles.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
BetaModelVariable$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Andrew J. Sims <andrew.sims5@nhs.net>

ChanceNode *ChanceNode*

Description

An R6 class to represent a chance node in a decision tree.

Details

An R6 class to represent a chance node in a decision tree. The node is associated with at least two branches to other nodes, each of which has a conditional probability (the probability of following that branch given that the node has been reached) and a cost.

Super class

`rdecision::Node` -> `ChanceNode`

Methods

Public methods:

- `ChanceNode$new()`
- `ChanceNode$getModelVariables()`
- `ChanceNode$sampleModelVariables()`
- `ChanceNode$updateEdges()`
- `ChanceNode$clone()`

Method `new()`**:** Create a new ChanceNode object

Usage:

```
ChanceNode$new(children, edgelabels, costs, p, ptype = "auto")
```

Arguments:

`children` a list of $k \geq 2$ Nodes which will be the children of this node.

`edgelabels` a list of k character strings to label each edge (branch) leaving the 'ChanceNode', given in the same order as 'children'.

`costs` A list of k costs associated with each edge (branch) leaving the 'ChanceNode'. Each element may be of type 'numeric' or 'ModelVariable'; given in the same order as 'children'.

`p` a list of probabilities associated with k branches. There are four possible configurations of the list based on the `ptype` argument.

'numeric' k numeric values; the simplest case in which the probabilities are certain. Supplied values should add to unity and be given in the same order as 'children'.

'MV' At least one 'ModelVariable' and exactly one 'NA", and the remainder either 'numeric' or 'ModelVariable', given in the same order as 'children'. The single NA will be replaced on evaluation of the model variables by a value to ensure the sum of probabilities is unity. This option is not recommended, because there is a chance that during sampling, individual branch probabilities may be less than zero, or that the sum of the sampled model variable expressions may exceed 1.

'Beta' One 'BetaModelVariable' and one NA. Used for $k = 2$. The element defined as NA will be replaced by one minus the sampled value of the supplied beta distribution.

'Dirichlet' One DirichletModelVariable, with k parameters, given in the same order as 'children'.

'auto' Infer which of the previous options applies based on the types of elements of p

`ptype` a character string taking one of four possible values to define how the `p` argument is defined: 'numeric', 'MV', 'Beta' or 'Dirichlet'.

Returns: A new 'ChanceNode' object

Method `getModelVariables()`**:** Return the list of model variables associated with the node. The model variables may be associated with costs or probabilities.

Usage:

```
ChanceNode$getModelVariables()
```

Returns: List of model variables.

Method `sampleModelVariables()`**:** Sample all model variables in this node and update edges.

Usage:

```
ChanceNode$sampleModelVariables(expected = F)
```

Arguments:

`expected` If TRUE cause each model variable to return its expected value at the next call to 'value()'. If FALSE each model variable will return the sampled value. Default is FALSE.

Returns: An updated ChanceNode object.

Method `updateEdges()`**:** Update numerical values in edges.

Usage:

```
ChanceNode$updateEdges()
```

Returns: An updated object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ChanceNode$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Andrew Sims <andrew.sims5@nhs.net>

`ConstModelVariable` *ConstModelVariable*

Description

An R6 class for a constant in a model

Details

A ModelVariable with no uncertainty in its value. It has no distribution and there are no hyper-parameters. Its benefit over using a regular 'numeric' variable in a model is that it will appear in automatic tabulations of the model variables associated with a model and therefore be explicitly documented as a model input.

Super class

rdecision::ModelVariable -> ConstModelVariable

Methods

Public methods:

- ConstModelVariable$new()
- ConstModelVariable$sample()
- ConstModelVariable$getDistribution()
- ConstModelVariable$getMean()
- ConstModelVariable$getSD()
- ConstModelVariable$getQuantile()
- ConstModelVariable$clone()

Method `new()`**:** Create a new constant model variable

Usage:

```
ConstModelVariable$new(description, units, const)
```

Arguments:

`description` A character string description of the variable and its role in the model. This description will be used in a tabulation of the variables linked to a model.

`units` A character string description of the units, e.g. 'GBP', 'per year'.

`const` The constant numerical value of the object.

Returns: A new ModelVariable object.

Method `sample()`**:** Not applicable for a constant variable, no action is taken on sampling.

Usage:

```
ConstModelVariable$sample(expected = F)
```

Arguments:

`expected` Logical; ignored.

Returns: Updated ModelVariable object.

Method `getDistribution()`**:** Accessor function for the name of the uncertainty distribution.

Usage:

```
ConstModelVariable$getDistribution()
```

Returns: Distribution name as character string.

Method `getMean()`**:** Return the expected value of the distribution.

Usage:

```
ConstModelVariable$getMean()
```

Returns: Expected value as a numeric value.

Method `getSD()`**:** Return the standard deviation of the distribution.

Usage:

```
ConstModelVariable$getSD()
```

Returns: Standard deviation as a numeric value

Method `getQuantile()`**:** Quantiles of the uncertainty distribution; for a constant all quantiles are returned as the value of the constant.

Usage:

```
ConstModelVariable$getQuantile(probs)
```

Arguments:

`probs` Numeric vector of probabilities, each in range [0,1].

Returns: Vector of numeric values of the same length as 'probs'.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ConstModelVariable$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Andrew Sims <andrew.sims@newcastle.ac.uk>

`DecisionNode` *DecisionNode*

Description

An R6 class for a decision node in a decision tree

Details

A class to represent a decision node in a decision tree. The node is associated with one or more branches to child nodes.

Value

Updated DecisionNode object.

Super class

`rdecision::Node` -> `DecisionNode`

Methods

Public methods:

- `DecisionNode$new()`
- `DecisionNode$getModelVariables()`
- `DecisionNode$sampleModelVariables()`
- `DecisionNode$updateEdges()`
- `DecisionNode$updateTree()`
- `DecisionNode$evaluatePathways()`
- `DecisionNode$evaluateChoices()`

- DecisionNode$clone()

Method `new()`**:** Create a new decision node.

Usage:

```
DecisionNode$new(children, edgelabels, costs)
```

Arguments:

`children` A list of Nodes which are the children of this decision node.

`edgelabels` A vector of character stings containing the labels of each choice associated with the decision.

`costs` A list of values containing the costs associated with each choice. Each value can be a numeric variable, or a ModelVariable.

Returns: A new DecisionNode object

Method `getModelVariables()`**:** Return the list of model variables associated with the node. The model variables may be associated with costs or probabilities.

Usage:

```
DecisionNode$getModelVariables()
```

Returns: List of model variables.

Method `sampleModelVariables()`**:** Sample all model variables in this node and update edges.

Usage:

```
DecisionNode$sampleModelVariables(expected = FALSE)
```

Arguments:

`expected` If TRUE cause each model variable to return its expected value at the next call to 'value()'. If FALSE each model variable will return the sampled value. Default is FALSE.

Returns: An updated DecisionNode object.

Method `updateEdges()`**:** Update numerical values in edges.

Usage:

```
DecisionNode$updateEdges()
```

Returns: An updated Node object.

Method `updateTree()`**:** Update the tree by sampling model variables and then updating numerical edge values.

Usage:

```
DecisionNode$updateTree(expected = TRUE)
```

Arguments:

`expected` if TRUE set model variables to their expectation; otherwise sample from their uncertainty distributions.

Method `evaluatePathways()`**:** Evaluate a decision. Starting with this decision node, the function works though all possible paths and computes the probability, cost and utility of each.

Usage:

```
DecisionNode$evaluatePathways(expected = TRUE, uncorrelate = FALSE)
```

Arguments:

`expected` If TRUE, evaluate each model variable as its mean value, otherwise sample each one from their uncertainty distrbution.

`uncorrelate` If TRUE, resample and update the tree between the evaluation of each choice. This causes any model variables that are common to more than one choice to be resampled between choices, and removes correlation due to shared model variables. Other forms of correlation may not be removed.

Returns: A data frame with one row per path and columns organized as follows:

Choice The choice with which the path is associated.

Pathway The leaf node on which the pathway ends; normally the clinical outcome.

Probability The probability of traversing the pathway. The total probability of each choice should sum to unity; i.e. the sum of the Probability column should equal the number of branches leaving the decision node.

Cost The cost of traversing the pathway.

ExpectedCost Cost $*$ probability of traversing the pathway.

Utility The utility associated with the outcome.

ExpectedUtility Utility $*$ probability of traversing the pathway.

Method `evaluateChoices()`**:** Evaluate each choice. Starting with this decision node, the function works though all possible paths and computes the probability, cost and utility of each, then aggregates by choice.

Usage:

```
DecisionNode$evaluateChoices(expected = TRUE, uncorrelate = FALSE, N = 1)
```

Arguments:

`expected` If TRUE, evaluate each model variable as its mean value, otherwise sample each one from their uncertainty distrbution.

`uncorrelate` If TRUE, resample and update the tree between the evaluation of each choice. This causes any model variables that are common to more than one choice to be resampled between choices, and removes correlation due to shared model variables.

`N` Number of replicates. Intended for use with PSA (expected=F); use with expected=T will be repetitive and uninformative.

Returns: A data frame with one row per choice per run and columns organized as follows:

Run The run number

Choice The choice.

Cost Aggregate cost of the choice.

Utility Aggregate utility of the choice.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DecisionNode$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Andrew J. Sims <andrew.sims@newcastle.ac.uk>

des *Discrete event simulation solver of Markov models*

Description

des is solver function for the rdecision package for discrete event simulations. The function is based on the model described by Sonnenberg and Beck. This function solves a Markov state model using a Monte-Carlo method, based on random number generation.

Usage

```
des(
  nStates,
  nGroups = 0,
  nPatients,
  nCyclesPerYear,
  nCycles,
  state,
  group = NA,
  prevalence,
  Ip,
  Tp,
  Gp = NA,
  discount = 0,
  stub = NA,
  continue = F
)
```

Arguments

nStates	Number of states in Markov model
nGroups	Number of groups (default zero); states can be grouped together and a maximum cycle limit set on group occupancy.
nPatients	Number of patients to simulate
nCyclesPerYear	Number of cycles per year
nCycles	Number of time cycles to simulate
state	Data frame of length nStates with named fields (see: State)
group	Data frame of length nGroups with named fields (see: Group)
prevalence	Array of length nStates which contains values for the initial state occupancy (prevalence). All values should be in the interval [0,1] and the sum of the elements should be 1.
Ip	Matrix of ANNUAL rates of transition between states. The dimensions are (nStates,nStates) with each entry being the annual rate of transitions from the row state to the column state. Values are in the range [0,1]. Transitions to self (i.e. leading diagonal) should NOT be included.
Tp	Matrix of dimension (nStates,nStates) which contain time independent probabilities of transitions from time limited states to other states after time expiry. Values are in the interval [0,1].

Gp	Matrix of dimension (nGroups, nStates) which contain time independent probabilities of transitions from time limited groups to other states after time expiry. Values are in the interval [0,1]. It is an error if any of the transitions from a group to a state within that group are > 0.
discount	Annual discount rate for costs and benefits as a percentage figure, eg 3.5. This is applied to each time cycle with log correction to convert from annual rate to per-cycle rate if required. Default 0.00.
stub	File stub to be used for output files, eg 'mymarkov' would cause files called 'mymarkov-log.txt' etc to be created. Default NA means no files are written.
continue	If no, states are populated according to the given prevalence; if yes, the simulation restarts in its previous state by reading in the csv files, and the prevalences are ignored. If stub is NA (file output is suppressed) continue=T is disallowed.

Value

A list of 4 matrices, for state populations, costs, entries and utilities.

State

Data frame `state` should have the following fields: `name` string which describes state (used in log output) `hasCycleLimit` value TRUE if cycle limit applies (ie temporary/tunnel state); FALSE otherwise (ie absorbing or normal state) `cycleLimit` (integer) maximum number of cycles for which a single patient can occupy the state, if hasCycleLimit = TRUE. For temporary states use hasCycleLimit=TRUE and cycleLimit=0. `annualCost` cost of being in the state for 1 year `entryCost` one-off cost associated with entering the state `utility` (incremental) utility of being in the state for one year `group` group number to which this state belongs (0 if no group, >= 1 otherwise)

Group

Date frame `group` should have the following fields: `number` (integer) group number as an integer (referenced by state$group); NA if not defined. `name` string which describes the group name (in output) `hasCycleLimit` value TRUE if cycle limit applies to the group, FALSE otherwise. `cycleLimit` maximum number of cycles for which a single patient can occupy the group.

Note

The following files will be created by the function: 'stub'-log.txt: a text file listing inputs, progress and final results of the simulation. 'stub'-pop.csv: a comma separated value file containing the population of each state at the end of each cycle. 'stub'-psp.csv: a comma separated value file containing the cumulative cost associated with entry and occupancy of each state after each cycle. 'stub'-ent.csv: a comma separated value file containing the cumulative number of entries into each state, after each cycle. 'stub'-utl.csv: a comma separated value file containing the cumulative utlity of each state, after each cycle.

References

Sonnenberg FA and Beck JR, "Markov models in medical decision making: a practical guide", Medical Decision Making 1993;13:322-339.

Edge *Edge*

Description

An R6 class to represent an edge in a decision tree

Details

Edges are the formal term for paths linking nodes in a hierarchical tree. It is not intended that package users creating models should instantiate the 'Edge' class. Instead, it is included in the package as a convenience class used in the construction and traversal of decision trees by the package methods themselves. In this case objects of type 'Edge' are used to collect together information relating to the same edge, i.e. a label, a cost and a conditional probability.

Methods

Public methods:

- `Edge$new()`
- `Edge$getToNode()`
- `Edge$getLabel()`
- `Edge$getCost()`
- `Edge$setCost()`
- `Edge$getP()`
- `Edge$setP()`
- `Edge$clone()`

Method `new()`**:** Create an object of type 'Edge'.

Usage:

```
Edge$new(fromNode, toNode, label, cost = 0, p = 1)
```

Arguments:

`fromNode` Node nearest the root to which the edge connects.

`toNode` Node nearest the leaf to which the edge connects.

`label` Character string containing the edge label.

`cost` Cost associated with traversing the edge; ModelVariable.

`p` Probability of traversing the edge conditional on having reached 'fromNode'; ModelVariable.

Returns: A new 'Edge' object.

Method `getToNode()`**:** Access toNode.

Usage:

```
Edge$getToNode()
```

Returns: 'Node' to which the edge leads.

Method `getLabel()`**:** Access label.

Usage:

```
Edge$getLabel()
```

Returns: Label of the edge; character string.

Method `getCost()`**:** Access cost.

Usage:

```
Edge$getCost()
```

Returns: Cost associated with traversing the edge; numeric.

Method `setCost()`**:** Set the cost associated with traversing the edge.

Usage:

```
Edge$setCost(c)
```

Arguments:

`c` Cost as a numeric value.

Returns: Updated object.

Method `getP()`**:** Get the conditional probability of traversing the edge.

Usage:

```
Edge$getP()
```

Returns: Conditional probability; numeric.

Method `setP()`**:** Set conditional probability of traversing the edge

Usage:

```
Edge$setP(p)
```

Arguments:

`p` Conditional probability value; a numeric in the range [0,1].

Returns: Updated ‘edge‘ object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Edge$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Andrew J. Sims <andrew.sims5@nhs.net>

ExpressionModelVariable

ExpressionModelVariable

Description

An R6 class for a model variable constructed from an expression involving other model variables.

Details

A class to support expressions involving objects of base class ModelVariable, which itself behaves like a model variable. For example if `A` and `B` are variables with base class `ModelVariable` and `c` is a variable of type `numeric`, then it is not possible to write, for example, `x <-42*A/B + c`, because R cannot manipulate class variables using the same operators as regular variables. But such forms of expression may be desirable in constructing a model and this class provides a mechanism for doing so.

Super class

`rdecision::ModelVariable` -> `ExpressionModelVariable`

Methods

Public methods:

- `ExpressionModelVariable$new()`
- `ExpressionModelVariable$sample()`
- `ExpressionModelVariable$value()`
- `ExpressionModelVariable$getDistribution()`
- `ExpressionModelVariable$getMean()`
- `ExpressionModelVariable$getSD()`
- `ExpressionModelVariable$getQuantile()`
- `ExpressionModelVariable$getOperands()`
- `ExpressionModelVariable$clone()`

Method `new()`**:** Create a Model Variable formed from an expression involving other model variables.

Usage:

```
ExpressionModelVariable$new(description, units, expr, envir = globalenv())
```

Arguments:

`description` Name for the model variable expresssion. In a complex model it may help to tabulate how model variables are combined into costs, probablities and rates.

`units` Units in which the variable is expressed.

`expr` An R expression involving model variables which would be syntactically correct were each model variable to be replaced by numerical variables. Create either by 'quote(x+y)' or 'rlang::expr(x+y)'.

`envir` The environment in which the model variables live. Normally, and by default, this is the global environment. But if an object is created which refers to model variables created in a different environment it must be specified. If creating an object from within a function, for example, set 'envir=environment()' in the parameter list.

Returns: An object of type ExpressionModelVariable

Method `sample()`**:** Set the value of the model variable from its uncertainty distribution. Nothing is returned; the sampled value is returned at the next call to 'value()'.

Usage:

```
ExpressionModelVariable$sample(expected = F)
```

Arguments:

`expected` Logical; if TRUE, sets the value of the model variable returned at subsequent calls to 'value()' to be equal to the expectation of the variable. Default is FALSE.

Returns: Updated ExpressionModelVariable object.

Method `value()`**:** Evaluate the expression.

Usage:

```
ExpressionModelVariable$value()
```

Returns: Numerical value of the evaluated expression.

Method `getDistribution()`**:** Accessor function for the name of the expression model variable.

Usage:

```
ExpressionModelVariable$getDistribution()
```

Returns: Expression as a character string with all control characters having been removed.

Method `getMean()`**:** Return the expected value of the expression variable.

Usage:

```
ExpressionModelVariable$getMean()
```

Returns: Expected value as a numeric value.

Method `getSD()`**:** Return the standard deviation of the distribution.

Usage:

```
ExpressionModelVariable$getSD()
```

Returns: Standard deviation as a numeric value

Method `getQuantile()`**:** Return the quantiles by sampling the variable.

Usage:

```
ExpressionModelVariable$getQuantile(probs)
```

Arguments:

`probs` Vector of probabilities, in range [0,1].

Returns: Vector of quantiles.

Method `getOperands()`**:** Return a list of operands that are themselves ModelVariables given in the expression.

Usage:

```
ExpressionModelVariable$getOperands()
```

Returns: A list of model variables.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ExpressionModelVariable$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

For many expressions involving model variables there will be no closed form expressions for the standard deviation an the quantiles. Therefore they are obtained by simulation. Method 'getDistribution' returns the string representation of the expression used to create the model variable.

Author(s)

Andrew J. Sims <andrew.sims5@newcastle.ac.uk>

---

fprintf *A version of fprintf for R*

Description

`fprintf` writes a formatted string to the file open on connection `con`. It is intended to be equivalent to the Matlab and C function of the same name.

Usage

```
fprintf(con, fmt, ...)
```

Arguments

`con`	an R connection, opened with the open command or one its overloaded methods, such as file or url.
`fmt`	a format string, of the syntax accepted by the R function sprintf.
`...`	a list of arguments, matching those specified in the `fmt` argument,, to be written to the connection.

Value

The number of characters written to the connection.

See Also

sprintf

Examples

```
ofid <- file("marvin.txt", open='wt')
fprintf(ofid, "The answer is %i\n", 42)
close(ofid)
```

GammaModelVariable *GammaModelVariable*

Description

An R6 class for a model variable with Gamma function uncertainty

Details

A model variable for which the uncertainty in the point estimate can be modelled with a Gamma distribution. The hyperparameters of the distribution are the shape ('alpha') and the scale ('beta') of the uncertainty distribution. Note that this variable naming convention follos Briggs et al; 'beta' is more usually used to describe the rate parameter (reciprocal of scale.)

Super class

rdecision::ModelVariable -> GammaModelVariable

Methods

Public methods:

- GammaModelVariable$new()
- GammaModelVariable$sample()
- GammaModelVariable$getDistribution()
- GammaModelVariable$getMean()
- GammaModelVariable$getSD()
- GammaModelVariable$getQuantile()
- GammaModelVariable$clone()

Method `new()`**:** Create an object of class GammaModelVariable.

Usage:

```
GammaModelVariable$new(description, units, alpha, beta)
```

Arguments:

`description` A character string describing the variable.

`units` Units of the variable, as character string.

`alpha` shape parameter of the Gamma distribution.

`beta` scale parameter of the Gamma distribution.

Returns: An object of class GammaModelVariable.

Method `sample()`**:** Set the value of the model variable from its uncertainty distribution. Nothing is returned; the sampled value is returned at the next call to 'value()'.

Usage:

```
GammaModelVariable$sample(expected = FALSE)
```

Arguments:

`expected` Logical; if TRUE sets the value of the model variable returned at subsequent calls to 'value()' to be equal to the expectation of the variable. Default is FALSE.

Returns: Updated GammaModelVariable object.

Method `getDistribution()`: Accessor function for the name of the uncertainty distribution.

Usage:

```
GammaModelVariable$getDistribution()
```

Returns: Distribution name as character string.

Method `getMean()`: Return the expected value of the distribution.

Usage:

```
GammaModelVariable$getMean()
```

Returns: Expected value as a numeric value.

Method `getSD()`: Return the standard deviation of the distribution.

Usage:

```
GammaModelVariable$getSD()
```

Returns: Standard deviation as a numeric value

Method `getQuantile()`: Return the quantiles of the Gamma uncertainty distribution.

Usage:

```
GammaModelVariable$getQuantile(probs)
```

Arguments:

`probs` Vector of probabilities, in range [0,1].

Returns: Vector of quantiles.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
GammaModelVariable$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Andrew J. Sims <andrew.sims5@nhs.net>

`LeafNode` *LeafNode*

Description

An R6 class for a leaf node in a decision tree

Details

A LeafNode is synomymous with a clinical outcome, or end point of a decision tree. Paths through the tree must end on leaf nodes.

Super class

`rdecision::Node` -> `LeafNode`

Methods

Public methods:

- LeafNode$new()
- LeafNode$getName()
- LeafNode$getUtility()
- LeafNode$clone()

Method `new()`**:** Create a new 'LeafNode' object; synonymous with a clinical outcome.

Usage:

```
LeafNode$new(name, utility = 1)
```

Arguments:

`name` Character string; a label for the leaf node which is synonymous with the name of the root-to-leaf pathway

`utility` the preference or value that a user associates with a given health state (range 0 to 1).

Returns: A new 'LeafNode' object

Method `getName()`**:** Return the label of the leaf node; the name of the clinical outcome.

Usage:

```
LeafNode$getName()
```

Returns: Name of the clinical outcome; character string.

Method `getUtility()`**:** Return the utility associated with the clinical outcome.

Usage:

```
LeafNode$getUtility()
```

Returns: Utility (numeric value)

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LeafNode$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Andrew J. Sims <andrew.sims5@newcastle.ac.uk>

LogNormalModelVariable

LogNormalModelVariable

Description

An R6 class for a model variable with logNormal uncertainty

Details

A model variable for which the uncertainty in the point estimate can be modelled with a logNormal distribution. ProbOnto defines seven parametrizations of the log normal distribution. These are linked, allowing the parameters of any one to be derived from any other. All 7 parameterizations require two parameters; their meanings are as follows:

LN1 $p_1 = \mu$, $p_2 = \sigma$, where μ and σ are the mean and standard deviation, both on the log scale.

LN2 $p_1 = \mu$, $p_2 = v$, where μ and v are the mean and variance, both on the log scale.

LN3 $p_1 = m$, $p_2 = \sigma$, where m is the median on the natural scale and σ is the standard deviation on the log scale.

LN4 $p_1 = m$, $p_2 = c_v$, where m is the median on the natural scale and c_v is the coefficient of variation on the natural scale.

LN5 $p_1 = \mu$, $p_2 = \tau$, where μ is the mean on the log scale and τ is the precision on the log scale.

LN6 $p_1 = m$, $p_2 = \sigma_g$, where m is the median on the natural scale and σ_g is the geometric standard deviation on the natural scale.

LN7 $p_1 = \mu_N$, $p1 = \sigma_N$, where μ_N is the mean on the natural scale and σ_N is the standard deviation on the natural scale.

Super class

`rdecision::ModelVariable` -> `LogNormalModelVariable`

Methods

Public methods:

- `LogNormalModelVariable$new()`
- `LogNormalModelVariable$sample()`
- `LogNormalModelVariable$getDistribution()`
- `LogNormalModelVariable$getMean()`
- `LogNormalModelVariable$getSD()`
- `LogNormalModelVariable$getQuantile()`
- `LogNormalModelVariable$clone()`

Method `new()`**:** Create a model variable with log normal uncertainty.

Usage:

```
LogNormalModelVariable$new(description, units, p1, p2, parametrization = "LN1")
```

Arguments:

`description` A character string describing the variable.

`units` Units of the quantity; character string.

`p1` First hyperparameter, a measure of location. See 'Details'.

`p2` Second hyperparameter, a measure of spread. See 'Details'.

`parametrization` A character string taking one of the values 'LN1' (default) through 'LN7' (see 'Details').

Returns: A LogNormalModelVariable object.

Method `sample()`**:** Set the value of the model variable from its uncertainty distribution. Nothing is returned; the sampled value is returned at the next call to 'value()'.

Usage:

```
LogNormalModelVariable$sample(expected = F)
```

Arguments:

`expected` Logical; if TRUE sets the value of the model variable returned at subsequent calls to 'value()' to be equal to the expectation of the variable. Default is FALSE.

Returns: Updated NormalModelVariable object.

Method `getDistribution()`**:** Accessor function for the name of the uncertainty distribution.

Usage:

```
LogNormalModelVariable$getDistribution()
```

Returns: Distribution name as character string (LN1, LN2 etc).

Method `getMean()`**:** Return the expected value of the distribution.

Usage:

```
LogNormalModelVariable$getMean()
```

Returns: Expected value as a numeric value.

Method `getSD()`**:** Return the standard deviation of the distribution.

Usage:

```
LogNormalModelVariable$getSD()
```

Returns: Standard deviation as a numeric value

Method `getQuantile()`**:** Return the quantiles of the logNormal uncertainty distribution.

Usage:

```
LogNormalModelVariable$getQuantile(probs)
```

Arguments:

`probs` Vector of probabilities, in range [0,1].

Returns: Vector of quantiles.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LogNormalModelVariable$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Andrew J. Sims <andrew.sims@newcastle.ac.uk>

MarkovModel *MarkovModel*

Description

An R6 class for a Markov model

Details

A class to represent a complete Markov model.

Methods

Public methods:

- MarkovModel$new()
- MarkovModel$cycle()
- MarkovModel$cycles()
- MarkovModel$getStatenames()
- MarkovModel$setDiscount()
- MarkovModel$setPopulations()
- MarkovModel$setTransitions()
- MarkovModel$stateSummary()
- MarkovModel$transitionSummary()
- MarkovModel$clone()

Method `new()`**:** Creates a Markov model.

Usage:

```
MarkovModel$new(states, Ip, discount = 0, nCyclesPerYear = 1)
```

Arguments:

`states` a list of objects of type MarkovState

`Ip` Matrix of *annual* rates of transition between states. The dimensions are (nStates,nStates) with each entry being the annual rate of transitions from the row state to the column state. Values are in the range [0,1]. Transitions to self (i.e. leading diagonal) should be set to zero, and will be adjusted so that the sum of transitions from each row are zero. The matrix should have row names and column names which are the state names.

`discount` The annual discount rate (percentage).

`nCyclesPerYear` The number of cycles per year.

Returns: A MarkovModel object.

Method `cycle()`**:** Applies one cycle of the model, starting at zero.

Usage:

```
MarkovModel$cycle()
```

Returns: Calculated values, per state.

Method `cycles()`: Applies `nCycles` cycles of the model. The starting populations are redistributed through the transition probabilities and the state occupancy costs are calculated, using function `cycle`. The end populations are then fed back into the model for a further cycle and the process is repeated. For each cycle, the state populations and the aggregated occupancy costs are saved in one row of the returned dataframe, with the cycle number. If the cycle count for the model is zero when called, the first cycle evaluated will be cycle zero, i.e. the distribution of patients to starting states, which will include calculation of state entry costs.

Usage:

```
MarkovModel$cycles(nCycles = 2)
```

Arguments:

`nCycles` Number of cycles to run; default is 2.

Returns: Data frame with cycle results.

Method `getStatenames()`: Returns a character list of state names.

Usage:

```
MarkovModel$getStatenames()
```

Returns: List of the names of each state.

Method `setDiscount()`: Set annual discount rate

Usage:

```
MarkovModel$setDiscount(r)
```

Arguments:

`r` Annual discount rate, expressed as a percentage.

Returns: Updated MarkovModel object.

Method `setPopulations()`: Sets the occupancy of each state. Takes argument Calling this resets the cycle count for the model to zero.

Usage:

```
MarkovModel$setPopulations(populations)
```

Arguments:

`populations` A named vector of populations for the start of the state. The names should be the state names. Due to R's implementation of matrix algebra, `populations` must be a numeric type and is not restricted to being an integer.

Returns: Updated MarkovModel object. Sets the annual state transition rates.

Method `setTransitions()`:

Usage:

```
MarkovModel$setTransitions(Ip)
```

Arguments:

`Ip` A numeric square matrix of dimension equal to the number of states, with row and column names equal to Markov state names. Transition rates are from row state to column state.

Returns: Updated MarkovModel object

Method `stateSummary()`: Creates a state summary data frame suitable for printing with kable etc.

Usage:

```
MarkovModel$stateSummary()
```

Returns: A dataframe with details of all states.

Method `transitionSummary()`**:** Creates a table of the annual transition probabilities.

Usage:

```
MarkovModel$transitionSummary()
```

Returns: A data frame of annual transition probabilities

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MarkovModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Andrew J. Sims <andrew.sims5@nhs.net>

`MarkovState` *MarkovState*

Description

An R6 class for a state in a Markov model

Details

A class to represent a single state in a Markov model.

Methods

Public methods:

- `MarkovState$new()`
- `MarkovState$getName()`
- `MarkovState$hasCycleLimit()`
- `MarkovState$getCycleLimit()`
- `MarkovState$setAnnualCost()`
- `MarkovState$getAnnualCost()`
- `MarkovState$setEntryCost()`
- `MarkovState$getEntryCost()`
- `MarkovState$clone()`

Method `new()`**:** Create a Markov state object.

Usage:

```
MarkovState$new(
  name,
  annualCost = 0,
  entryCost = 0,
  hasCycleLimit = F,
  cycleLimit = NA
)
```

Arguments:

`name` The name of the state (character string).

`annualCost` The annual cost of state occupancy.

`entryCost` The cost to enter the state.

`hasCycleLimit` Whether the state has a cycle limit; logical.

`cycleLimit` The maximum length of stay (1 for cohort tunnel states); numeric.

`isTemporaryState` Logical; TRUE if the state is a tunnel state.

Returns: An object of type MarkovState.

Method `getName()`**:** Accessor function to retrieve the state name.

Usage:

```
MarkovState$getName()
```

Returns: State name.

Method `hasCycleLimit()`**:** Accessor function to identify if the state has a cycle limit.

Usage:

```
MarkovState$hasCycleLimit()
```

Returns: TRUE if temporary state; false otherwise.

Method `getCycleLimit()`**:** Accessor function to find the cycle limit.

Usage:

```
MarkovState$getCycleLimit()
```

Returns: The cycle limit; integer.

Method `setAnnualCost()`**:** Sets the annual cost of state occupancy.

Usage:

```
MarkovState$setAnnualCost(annualCost)
```

Arguments:

`annualCost` Annual cost of occupying the state; numeric.

Returns: Updated MarkovState object.

Method `getAnnualCost()`**:** Gets the annual cost of state occupancy.

Usage:

```
MarkovState$getAnnualCost()
```

Returns: Annual cost; numeric.

Method `setEntryCost()`**:** Sets the entry cost of state occupancy.

Usage:

```
MarkovState$setEntryCost(entryCost)
```

Arguments:

`entryCost` Cost to enter the state; numeric.

Returns: Updated MarkovState object.

Method `getEntryCost()`**:** Gets the entry cost of state occupancy.

Usage:

```
MarkovState$getEntryCost()
```

Returns: Entry cost; numeric.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MarkovState$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Andrew J. Sims <andrew.sims5@nhs.net>

`ModelVariable` *ModelVariable*

Description

An R6 class for a variable in an health economic model

Details

Base class for a variable used in a health economic model. The base class, which is not intended to be directly instantiated by model applications, wraps a numerical value which is used in calculations. The base class provides a framework for creating classes of model variables whose uncertainties are described by statistical distributions parametrized with hyperparameters.

Methods

Public methods:

- `ModelVariable$new()`
- `ModelVariable$get_environment()`
- `ModelVariable$isExpression()`
- `ModelVariable$sample()`
- `ModelVariable$value()`
- `ModelVariable$get_label()`
- `ModelVariable$set_label()`
- `ModelVariable$unset_label()`
- `ModelVariable$getDescription()`
- `ModelVariable$getUnits()`
- `ModelVariable$getDistribution()`

- ModelVariable$getOperands()
- ModelVariable$tabulate()
- ModelVariable$getMean()
- ModelVariable$getSD()
- ModelVariable$getQuantile()
- ModelVariable$r()
- ModelVariable$clone()

Method `new()`**:** Create an object of type 'ModelVariable'

Usage:

```
ModelVariable$new(description, units)
```

Arguments:

`description` A character string description of the variable and its role in the model. This description will be used in a tabulation of the variables linked to a model.

`units` A character string description of the units, e.g. 'GBP', 'per year'.

Returns: A new ModelVariable object.

Method `get_environment()`**:** Find the environment nearest the global environment in which this model variable or a reference to it, lives.

Usage:

```
ModelVariable$get_environment()
```

Returns: An environment. If not found, returns the empty environment.

Method `isExpression()`**:** Is this ModelVariable an expression?

Usage:

```
ModelVariable$isExpression()
```

Returns: TRUE if it inherits from ExpressionModelVariable, FALSE otherwise.

Method `sample()`**:** Set the value of the model variable from its uncertainty distribution. Nothing is returned; the sampled value is returned at the next call to 'value()'.

Usage:

```
ModelVariable$sample(expected = F)
```

Arguments:

`expected` Logical; if TRUE sets the value of the model variable returned at subsequent calls to 'value()' to be equal to the expectation of the variable. Default is FALSE.

Returns: Updated ModelVariable object.

Method `value()`**:** Return the current value of the model variable. This will be the expected value if the argument to the most recent call to 'sample' was TRUE or after creation of the object; otherwise it will return a value sampled from the uncertainty distribution.

Usage:

```
ModelVariable$value()
```

Returns: Numeric value of the model variable.

Method `get_label()`**:** Accessor function for the label. Unless 'set_label' has been called, this function will return the variable's own name. As far as possible, this will be the variable name used when the object was first created in the model, so that it aligns with the variable name used in ExpressionModelVariables and tabulations of variables used in models. But due to the nature of R's non-standard evaluation, this is not ensured. It will sometimes return the name of a reference to the original variable, if the original environment is not an ancestor of the calling environment of this function. In cases where references to model variables are routinely created, destroyed and passed around, it is advised to call this method from the environment of the original object as soon as possible after it is created. Subsequent calls will return the original name.

Usage:

```
ModelVariable$get_label()
```

Returns: Label of model variable as character string.

Method `set_label()`**:** Function to set the label. Normally this is not required because the label is set to the name given to the variable by the user. This function allows the default behaviour to be overridden.

Usage:

```
ModelVariable$set_label(label)
```

Arguments:

`label` The label to use.

Returns: Updated ModelVariable object

Method `unset_label()`**:** Function to unset the label. Causes the label to revert to its default, i.e. its variable name at the next call to 'get_label'.

Usage:

```
ModelVariable$unset_label()
```

Returns: Updated ModelVariable object

Method `getDescription()`**:** Accessor function for the description.

Usage:

```
ModelVariable$getDescription()
```

Returns: Description of model variable as character string.

Method `getUnits()`**:** Accessor function for units.

Usage:

```
ModelVariable$getUnits()
```

Returns: Description of units as character string.

Method `getDistribution()`**:** Accessor function for the name of the uncertainty distribution.

Usage:

```
ModelVariable$getDistribution()
```

Returns: Distribution name as character string.

Method `getOperands()`**:** Return a list of operands given in the expression used to form the expression. Only relevant for objects of inherited type ExpressionModelVariable, but defined for the base class for convenience to avoid type checking inside iterators.

Usage:

```
ModelVariable$getOperands()
```

Returns: A list of operands that are themselves ModelVariables. An empty list for non-expression model variables.

Method `tabulate()`**:** Tabulate the model variable and optionally include its operands.

Usage:

```
ModelVariable$tabulate(include.operands = FALSE)
```

Arguments:

`include.operands` If TRUE include the operands of this model variable in the table. Otherwise return a table with one row, describing this variable.

Returns: Data frame with one row per model variable, as follows:

Label The label given to the variable on creation.

Description As given at initialization.

Units Units of the variable.

Distribution Either the uncertainty distribution, if it is a regular model variable, or the expression used to create it, if it is an ExpressionModelVariable.

Mean Expected value.

SD Standard deviation.

Q2.5 p=0.025 quantile.

Q97.5 p=0.975 quantile.

Qhat Asterisk (*) if the quantiles and SD have been estimated by random sampling.

Method `getMean()`**:** Return the expected value of the distribution.

Usage:

```
ModelVariable$getMean()
```

Returns: Expected value as a numeric value.

Method `getSD()`**:** Return the standard deviation of the distribution.

Usage:

```
ModelVariable$getSD()
```

Returns: Standard deviation as a numeric value

Method `getQuantile()`**:** Find quantiles of the uncertainty distribution.

Usage:

```
ModelVariable$getQuantile(probs)
```

Arguments:

`probs` Numeric vector of probabilities, each in range [0,1].

Returns: Vector of numeric values of the same length as ‘probs‘.

Method `r()`**:** Draw random samples from the model variable. After returning the sample, the next call to ‘value()‘ will return the expected value.

Usage:

```
ModelVariable$r(n)
```

Arguments:

`n` Number of samples to return

Returns: Numeric vector of samples drawn at random.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ModelVariable$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Andrew Sims <andrew.sims@newcastle.ac.uk>

Node *Node*

Description

An R6 class to represent a node in a decision tree

Details

Base class to represent a single node in a decision tree. Non subclassed nodes are not expected to be created as model objects. Document Object Model (DOM) names are used for node methods as far as possibe.

Methods

Public methods:

- `Node$new()`
- `Node$hasChildNodes()`
- `Node$childNodes()`
- `Node$descendantNodes()`
- `Node$isSameNode()`
- `Node$nodeType()`
- `Node$getPathways()`
- `Node$getLabel()`
- `Node$getP()`
- `Node$getUtility()`
- `Node$getCost()`
- `Node$getModelVariables()`
- `Node$tabulateModelVariables()`
- `Node$sampleModelVariables()`
- `Node$updateEdges()`
- `Node$clone()`

Method `new()`**:** Create new Node object.

Usage:

```
Node$new()
```

Returns: A new Node object.

Method `hasChildNodes()`**:** Does the node have any child nodes? (DOM-style)

Usage:

```
Node$hasChildNodes()
```

Returns: TRUE if node has children, FALSE if not

Method `childNodes()`**:** Return list of child nodes (DOM-style)

Usage:

```
Node$childNodes()
```

Returns: list of child Nodes

Method `descendantNodes()`**:** Return list of descendent nodes.

Usage:

```
Node$descendantNodes()
```

Returns: List of descendent nodes, including self.

Method `isSameNode()`**:** Is this node the same as the argument? (DOM-style)

Usage:

```
Node$isSameNode(otherNode)
```

Arguments:

`otherNode` node to compare with this one

Returns: TRUE if 'otherNode' is also this one

Method `nodeType()`**:** node type (DOM-style)

Usage:

```
Node$nodeType()
```

Returns: Node class, as character string

Method `getPathways()`**:** Trace and list all pathways ending on leaf nodes which start with this node.

Usage:

```
Node$getPathways(choice = NA)
```

Arguments:

`choice` Name of choice. All pathways are returned if NA.

Returns: A list of Path objects. Each member of the list is a path from this node to a leaf, limited to those associated with choice, if defined.

Method `getLabel()`**:** Return label of edge which links to specified child node

Usage:

```
Node$getLabel(childNode)
```

Arguments:

`childNode` child node to which find label of linking edge

Returns: label as character string

Method `getP()`**:** Function to return the conditional probability of the edge which links to the specified child node

Usage:

```
Node$getP(childNode)
```

Arguments:

`childNode` child node to which to find probability of linking edge

Returns: numerical value of probability

Method `getUtility()`**:** function to return the utility associated with the node

Usage:

```
Node$getUtility()
```

Returns: Utility, numeric

Method `getCost()`**:** Function to return the cost of the edge which links to the specified child node

Usage:

```
Node$getCost(childNode)
```

Arguments:

`childNode` child node to identify edge with associated cost of traversal

Returns: Cost, numerical value

Method `getModelVariables()`**:** Function to return a list of model variables associated with this node.

Usage:

```
Node$getModelVariables()
```

Returns: List of model variables associated with this node.

Method `tabulateModelVariables()`**:** Tabulate all model variables associated with this node.

Usage:

```
Node$tabulateModelVariables(
  include.descendants = FALSE,
  include.operands = FALSE
)
```

Arguments:

`include.descendants` If TRUE, model variables associated with this node and its descendants are tabulated; otherwise only the ones that are associated with this node.

`include.operands` If TRUE, recursively add model variables which are included in expressions in ExpressionModelVariables. Default is FALSE.

Returns: Data frame with one row per model variable, as follows:

Label The label given to the variable on creation.

Description As given at initialization.

Units Units of the variable.

Distribution The uncertainty distribution or an expression.

Mean Expected value.

SD Standard deviation.

Q2.5 p=0.025 quantile.

Q97.5 p=0.975 quantile.

Qhat Asterisk if the quantiles and SD were estimated by random sampling.

Method `sampleModelVariables()`**:** Sample the model variables associated with the node.

Usage:

```
Node$sampleModelVariables(expected = FALSE)
```

Arguments:

`expected` if TRUE, use the expected value of the model variables in the node; otherwise sample from their uncertainty distributions.

Returns: Updated Node object

Method `updateEdges()`**:** Update the values on the edges associated with the node.

Usage:

```
Node$updateEdges()
```

Returns: Updated Node object

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Node$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Andrew Sims <andrew.sims5@nhs.net>

`NormalModelVariable` *NormalModelVariable*

Description

An R6 class for a model variable with Normal uncertainty

Details

A model variable for which the uncertainty in the point estimate can be modelled with a Normal distribution. The hyperparameters of the distribution are the mean (‘mu‘) and the standard deviation (‘sd‘) of the uncertainty distribution. The value of ‘mu‘ is the expected value of the variable.

Super class

`rdecision::ModelVariable` -> `NormalModelVariable`

Methods

Public methods:

- `NormalModelVariable$new()`
- `NormalModelVariable$sample()`
- `NormalModelVariable$getDistribution()`
- `NormalModelVariable$getMean()`
- `NormalModelVariable$getSD()`
- `NormalModelVariable$getQuantile()`
- `NormalModelVariable$clone()`

Method `new()`**:** Create a model variable with normal uncertainty.

Usage:

```
NormalModelVariable$new(description, units, mu, sigma)
```

Arguments:

`description` A character string describing the variable.

`units` Units of the quantity; character string.

`mu` Hyperparameter with mean of the Normal distribution for the uncertainty of the variable.

`sigma` Hyperparameter equal to the standard deviation of the normal distribution for the uncertainty of the variable.

Returns: A NormalModelVariable object.

Method `sample()`**:** Set the value of the model variable from its uncertainty distribution. Nothing is returned; the sampled value is returned at the next call to 'value()'.

Usage:

```
NormalModelVariable$sample(expected = F)
```

Arguments:

`expected` Logical; if TRUE sets the value of the model variable returned at subsequent calls to 'value()' to be equal to the expectation of the variable. Default is FALSE.

Returns: Updated NormalModelVariable object.

Method `getDistribution()`**:** Accessor function for the name of the uncertainty distribution.

Usage:

```
NormalModelVariable$getDistribution()
```

Returns: Distribution name as character string.

Method `getMean()`**:** Return the expected value of the distribution.

Usage:

```
NormalModelVariable$getMean()
```

Returns: Expected value as a numeric value.

Method `getSD()`**:** Return the standard deviation of the distribution.

Usage:

```
NormalModelVariable$getSD()
```

Returns: Standard deviation as a numeric value

Method `getQuantile()`**:** Return the quantiles of the Normal uncertainty distribution.

Usage:

```
NormalModelVariable$getQuantile(probs)
```

Arguments:

`probs` Vector of probabilities, in range [0,1].

Returns: Vector of quantiles.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
NormalModelVariable$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Andrew J. Sims <andrew.sims5@nhs.net>

Path *Path*

Description

An R6 class for pathway through a decision tree.

Details

A class to represent a traversal through a decision tree. Essentially a wrapper for an ordered list of Nodes.

Methods

Public methods:

- `Path$new()`
- `Path$getChoice()`
- `Path$getName()`
- `Path$getProbability()`
- `Path$getCost()`
- `Path$getUtility()`
- `Path$tabulate()`
- `Path$clone()`

Method `new()`**:** Create a path from a list of nodes. Must end on a leaf node.

Usage:

```
Path$new(nodes)
```

Arguments:

`nodes` A list of Nodes.

Method `getChoice()`**:** Returns the name of the choice arising from the first decision node of a list of nodes. In many decision trees, where the decision node is at the left, many leaf nodes (pathway names) will be associated with the same choice.

Usage:

```
Path$getChoice()
```

Returns: label of the choice; specifically the `edgelabel` field of the first decision node in `nodes`.

Method `getName()`**:** Returns the name of the pathway from the (final) leaf node of a list of nodes.

Usage:

```
Path$getName()
```

Returns: name of pathway; specifically the @codepathway field of the final leaf node in @codenodes.

Method `getProbability()`**:** Calculates the product of the conditional P values in the tree traversal.

Usage:
```
Path$getProbability()
```
Returns: Product of conditional probabilities.

Method `getCost()`**:** Calculates the sum of the pathway costs in the tree traversal.
Usage:
```
Path$getCost()
```
Returns: Sum of the pathway costs.

Method `getUtility()`**:** Calculates the sum of the pathway utilities in the tree traversal.
Usage:
```
Path$getUtility()
```
Returns: Sum of pathway utilities

Method `tabulate()`**:** Evaluate and tabulate the pathway.
Usage:
```
Path$tabulate()
```
Returns: Data frame with one row, as follows:
Choice The choice from which the pathway leads.
Pathway The end-point or outcome; i.e. label ofleaf node.
Probability Product of probabilities.
Cost Sum of cost.
Utility Sum of utility.

Method `clone()`**:** The objects of this class are cloneable with this method.
Usage:
```
Path$clone(deep = FALSE)
```
Arguments:
`deep` Whether to make a deep clone.

Note

Uses the first decision node in the list.

Assumes the leaf node is the last node in the list.

Assumes the nodes are supplied in order of traversal and that the final node is a leaf node.

Assumes that the final node is a leaf node.

Assumes that only the final leaf node has a utility value.

Author(s)

Andrew J. Sims <andrew.sims5@nhs.net>

`rdecision` *rdecision: Decision Analytic Modelling in Health Economics.*

Description

Classes and functions for modelling healthcare interventions using cohort models (decision trees, Markov models and extended Markov models). It draws on terminology from Briggs, Claxton and Sculpher, "Decision Modelling for Health Economic Evaluation", Oxford University Press, 2006.

Index