

# Package ‘MoTBFs’

July 22, 2025

**Type** Package

**Title** Learning Hybrid Bayesian Networks using Mixtures of Truncated Basis Functions

**Version** 1.4.2

**Description** Learning, manipulation and evaluation of mixtures of truncated basis functions (MoTBFs), which include mixtures of polynomials (MOPs) and mixtures of truncated exponentials (MTEs). MoTBFs are a flexible framework for modelling hybrid Bayesian networks (I. Pérez-Bernabé, A. Salmerón, H. Langseth (2015) <[doi:10.1007/978-3-319-20807-7\\_36](https://doi.org/10.1007/978-3-319-20807-7_36)>; H. Langseth, T.D. Nielsen, I. Pérez-Bernabé, A. Salmerón (2014) <[doi:10.1016/j.ijar.2013.09.012](https://doi.org/10.1016/j.ijar.2013.09.012)>; I. Pérez-Bernabé, A. Fernández, R. Rumí, A. Salmerón (2016) <[doi:10.1007/s10618-015-0429-7](https://doi.org/10.1007/s10618-015-0429-7)>). The package provides functionality for learning univariate, multivariate and conditional densities, with the possibility of incorporating prior knowledge. Structural learning of hybrid Bayesian networks is also provided. A set of useful tools is provided, including plotting, printing and likelihood evaluation. This package makes use of S3 objects, with two new classes called 'motbf' and 'jointmotbf'.

**Depends** R (>= 3.2.0)

**Imports** quadprog, lpSolve, bnlearn, methods, ggmm, Matrix

**Encoding** UTF-8

**License** LGPL-3

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2025-07-22 11:01:49 UTC

**RoxygenNote** 7.2.3

**Author** Inmaculada Pérez-Bernabé [aut],  
Antonio Salmerón [aut],  
Thomas D. Nielsen [aut],  
Ana D. Maldonado [aut, cre]

**Maintainer** Ana D. Maldonado <[ana.d.maldonado@ual.es](mailto:ana.d.maldonado@ual.es)>

## Contents

as.function.jointmotbf . . . . .	3
as.function.motbf . . . . .	4
asMOPString . . . . .	5
asMTEString . . . . .	6
BICMoTBF . . . . .	7
BICMultiFunctions . . . . .	8
Class-JointMoTBF . . . . .	9
Class-MoTBF . . . . .	10
clean . . . . .	11
coef.jointmotbf . . . . .	11
coef.mop . . . . .	12
coef.motbf . . . . .	13
coef.mte . . . . .	15
coefExpJointCDF . . . . .	16
conditionalmotbf.learning . . . . .	16
dataMining . . . . .	19
derivMOP . . . . .	21
derivMoTBF . . . . .	22
derivMTE . . . . .	23
dimensionFunction . . . . .	24
discreteStatesFromBN . . . . .	25
ecoli . . . . .	26
evalJointFunction . . . . .	27
findConditional . . . . .	28
forward_sampling . . . . .	29
generateNormalPriorData . . . . .	30
getChildParentsFromGraph . . . . .	31
getCoefficients . . . . .	32
getNonNormalisedRandomMoTBF . . . . .	34
goodnessDiscreteVariables . . . . .	35
goodnessMoTBFBN . . . . .	36
integralJointMoTBF . . . . .	37
integralMOP . . . . .	38
integralMoTBF . . . . .	39
integralMTE . . . . .	41
is.discrete . . . . .	42
is.observed . . . . .	43
is.root . . . . .	43
jointCDF . . . . .	44
jointmotbf.learning . . . . .	45
LearningHC . . . . .	46
learnMoTBFpriorInformation . . . . .	48
marginalJointMoTBF . . . . .	49
mop.learning . . . . .	51
MoTBF-Distribution . . . . .	52
MoTBFs_Learning . . . . .	54

motbf_type . . . . .	56
mte.learning . . . . .	56
newRangePriorData . . . . .	58
nVariables . . . . .	59
parentValues . . . . .	60
plot.jointmotbf . . . . .	61
plot.motbf . . . . .	63
plotConditional . . . . .	64
preprocessedData . . . . .	65
printBN . . . . .	66
printConditional . . . . .	67
printDiscreteBN . . . . .	68
probDiscreteVariable . . . . .	68
r.data.frame . . . . .	69
rescaledFunctions . . . . .	70
rnormMultiv . . . . .	71
sample_MoTBFs . . . . .	72
Subclass-MoTBF . . . . .	73
subsetData . . . . .	74
summary.jointmotbf . . . . .	75
summary.motbf . . . . .	76
thyroid . . . . .	78
univMoTBF . . . . .	79
UpperBoundLogLikelihood . . . . .	80

**Index****82**


---

as.function.jointmotbf

*Coerce a "jointmotbf" Object to a Function*


---

**Description**

Takes a "jointmotbf" object and constructs an R function to evaluate it at multidimensional points.

**Usage**

```
## S3 method for class 'jointmotbf'
as.function(x, ...)
```

**Arguments**

x                   An object of class "jointmotbf".

...                  Further arguments to be passed to or from the method. Not necessary for this method.

**Details**

This is an S3 method for the generic function [as.function](#).

**Value**

It returns a function to evaluate an object of class "jointmotbf".

**See Also**

[parametersJointMoTBF](#) and [jointMoTBF](#)

**Examples**

```
## 1.EXAMPLE
## Dataset
data <- data.frame(X = rnorm(100), Y = rexp(100))

## Joint function
dim <- c(3,2)
param <- parametersJointMoTBF(data, dimensions = dim)
P <- jointMoTBF(param)
density <- as.function(P)(data[,1], data[,2])
density

## Log-likelihood
sum(log(density))

#####
## MORE EXAMPLES #####
#####

## Dataset
data <- data.frame(X = rnorm(100), Y = rexp(100), Z = rnorm(100))

## Joint function
dim <- c(2,3,4)
param <- parametersJointMoTBF(data, dimensions = dim)
P <- jointMoTBF(param)
density <- as.function(P)(data[,1], data[,2], data[,3])
density

## Log-likelihood
sum(log(density))
```

**Description**

Takes an "motbf" object and constructs an R function to evaluate it at points.

**Usage**

```
## S3 method for class 'motbf'
as.function(x, ...)
```

**Arguments**

x	An object of class "motbf".
...	Further arguments to be passed to or from the method. Not necessary for this method.

**Details**

This is an S3 method for the generic function [as.function](#).

**Value**

It returns a function to evaluate an object of class "motbf".

**Examples**

```
## Data
X <- rchisq(5000, df = 3)

## Learning
P <- univMotBF(X, POTENTIAL_TYPE = "MOP"); P

## Evaluation
as.function(P)(min(X))
as.function(P)(max(X))
as.function(P)(10)
density <- as.function(P)(X)

## Plot
hist(X, prob = TRUE, main = "")
points(X, density, col=4, pch=16)
```

---

asMOPString

*Parameters to MOP String*


---

**Description**

This function builds a string with the structure of a 'mop' function.

**Usage**

```
asMOPString(parameters)
```

**Arguments**

parameters     A "numeric" vector containing the coefficients.

**Value**

A "character" string with a 'mop' structure.

**Examples**

```
param <- c(1,2,3,4)
asMOPString(param)
```

```
param <- 3.4
asMOPString(param)
```

---

asMTEString

*Converting MTEs to strings*

---

**Description**

This function builds a string with the structure of an 'mte' function.

**Usage**

```
asMTEString(parameters, num = 5)
```

**Arguments**

parameters     A "numeric" vector containing the coefficients.

num            A "numeric" value which contains the denominator of the coefficient in the exponential.

**Value**

A "character" string with an 'mte' structure.

**Examples**

```
param <- -5.8
asMTEString(param)
```

```
param <- c(5.2,0.3,-3,4)
asMTEString(param)
```

---

**BICMoTBF***Computing the BIC score of an MoTBF function*

---

**Description**

Computes the Bayesian information criterion value (BIC) of a mixture of truncated basis functions. The BIC score is the log likelihood penalized by the number of parameters of the function and the number of records of the evaluated data.

**Usage**

```
BICMoTBF(Px, X)
```

**Arguments**

**Px** A function of class "motbf".  
**X** A "numeric" vector with the data to evaluate.

**Value**

A "numeric" value corresponding to the BIC score.

**See Also**

[univMoTBF](#)

**Examples**

```
## Data
X <- rexp(10000)

## Data test
Xtest <- rexp(1000)
Xtest <- Xtest[Xtest>=min(X) & Xtest<=max(X)]

## Learning
f1 <- univMoTBF(X, POTENTIAL_TYPE = "MOP", nparam = 10); f1
f2 <- univMoTBF(X, POTENTIAL_TYPE = "MTE", maxParam = 11); f2

## BIC values
BICMoTBF(Px = f1, X = Xtest)
BICMoTBF(Px = f2, X = Xtest)
```

BICMultiFunctions      *BIC score for multiple functions*

---

### Description

Compute the BIC score using more than one probability functions.

### Usage

```
BICMultiFunctions(Px, X)
```

### Arguments

Px                    A list of objects of class "motbf".  
X                     A list with as many "numeric" vectors as densities in Px, used to compute the BIC score for each density.

### Value

The "numeric" BIC value.

### See Also

[univMoTBF](#)

### Examples

```
## Data
X <- rnorm(500)
Y <- rnorm(500, mean=1)
data <- data.frame(X=X, Y=Y)
## Data as a "list"
Xlist <- sapply(data, list)

## Learning as a "list"
Plist <- lapply(data, univMoTBF, POTENTIAL_TYPE="MOP")
Plist

## BIC value
BICMultiFunctions(Px=Plist, X=Xlist)
```



---

Class-JointMoTBF	Class "jointmotbf"
------------------	--------------------

---

### Description

Defines an object of class "jointmotbf" and other basic functions for manipulating "jointmotbf" objects.

### Usage

```
jointmotbf(x = 0)

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

## S3 method for class 'jointmotbf'
as.character(x, ...)

## S3 method for class 'jointmotbf'
as.list(x, ...)

is.jointmotbf(x, class = "jointmotbf")
```

### Arguments

x	Preferably, a list containing an expression and other posibles elements like a "numeric" matrix with the domain of the variables, the dimension of the variables, the number of iterations needed to solve the optimization problem, among others. Any R object can be entered, but the utility of this function is not to transform objects of other classes into objects of class "jointmotbf".
...	Additional arguments, not needed by these methods.
class	By default is "jointmotbf".

### See Also

[jointMoTBF](#)

### Examples

```
## n.parameters is the product of the dimensions
dim <- c(3,3)
param <- seq(1,prod(dim), by=1)
## Joint Function
f <- list(Parameters=param, Dimensions=dim)
jointF <- jointMoTBF(f)

print(jointF) ## jointF
```

```

as.character(jointF)
as.list(jointF)
is(jointF)
is.jointmotbf(jointF)

```

---

Class-MoTBF

*Class "motbf"*


---

### Description

Defines an object of class "motbf" and other basic functions for manipulating "motbf" objects.

### Usage

```

motbf(x = 0)

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

## S3 method for class 'motbf'
as.character(x, ...)

## S3 method for class 'motbf'
as.list(x, ...)

is.motbf(x, class = "motbf")

```

### Arguments

<code>x</code>	Preferably, a list containing an 'mte' or 'mop' univariate expression and other possible elements like a "numeric" vector with the domain of the variable, the number of iterations needed to solve the optimization problem, among others. Any R object can be entered, but the utility of this function is not to transform objects of other classes into objects of class "motbf".
<code>...</code>	Additional arguments, not needed for these methods.
<code>class</code>	By default is "motbf".

### See Also

[asMOPString](#) and [asMTEString](#)

### Examples

```

## Subclass 'MOP'
param <- c(1,2,3,4,5)
MOPString <- asMOPString(param)
fMOP <- motbf(MOPString)
print(fMOP) ## fMOP

```

```

as.character(fMOP)
as.list(fMOP)
is(fMOP)
is.motbf(fMOP)

## Subclass 'MTE'
param <- c(6,7,8,9,10)
MTEString <- asMTEString(param)
fMTE <- motbf(MTEString)
print(fMTE) ## MTE
as.character(fMTE)
as.list(fMTE)
is(fMTE)
is.motbf(fMTE)

```

---

clean	<i>Remove Objects from Memory</i>
-------	-----------------------------------

---

### Description

Clean the memory. Delete all the objects in memory and a garbage collection takes place.

### Usage

```
clean(envir = globalenv(), n = 2)
```

### Arguments

envir	The currently active environment; by default It is the gloval environment.
n	Number of garbage collection repetitions; by default n = 2.

### Examples

```

## Run to clean the environment
clean()
clean(n=2)

```

---

coef.jointmotbf	<i>Coefficients of a "jointmotbf" object</i>
-----------------	--

---

### Description

Extracts the parameters of a joint MoTBF density.

### Usage

```

## S3 method for class 'jointmotbf'
coef(object, ...)

```

**Arguments**

object            An MoTBF function.  
 ...              Other arguments, unnecessary for this function.

**Value**

A "numeric" vector with the parameters of the function.

**See Also**

[parametersJointMoTBF](#) and [jointMoTBF](#)

**Examples**

```
## Generate a dataset
data <- data.frame(X1 = rnorm(100), X2 = rnorm(100))

## Joint function
dim <- c(2,4)
param <- parametersJointMoTBF(data, dimensions = dim)
P <- jointMoTBF(param)
P$Time

## Coefficients
coef(P)

#####
## MORE EXAMPLES #####
#####

## Generate a dataset
data <- data.frame(X1 = rnorm(100), X2 = rnorm(100), X3 = rnorm(100))

## Joint function
dim <- c(2,4,3)
param <- parametersJointMoTBF(data, dimensions = dim)
P <- jointMoTBF(param)
P$Time

## Coefficients
coef(P)
```

---

coef.mop

*Extract coefficients from MOPs*

---

**Description**

It extracts the parameters of the learned mixtures of polynomial models.

**Usage**

```
coeffMOP(fx)
```

```
coeffPol(fx)
```

**Arguments**

fx                    An "motbf" function of subclass 'mop'.

**Details**

coeffMOP() return the coefficients of the terms in the function.

coeffPol() returns the coefficients of the potential of the polynomial basis in the function.

**Value**

An array with the parameters of the function.

**See Also**

[coef.motbf](#) and [univMoTBF](#)

**Examples**

```
## 1. EXAMPLE
data <- rchisq(1000, df=5)
fx1 <- univMoTBF(data, POTENTIAL_TYPE = "MOP")
hist(data, prob=TRUE, main="")
plot(fx1, xlim=range(data), col="red", add=TRUE)
coeffMOP(fx1) ## coef(fx1)
coeffPol(fx1)

## 2. EXAMPLE
data <- rexp(1000, rate=1/2)
fx2 <- univMoTBF(data, POTENTIAL_TYPE = "MOP")
hist(data, prob=TRUE, main="")
plot(fx2, xlim=range(data), col="red", add=TRUE)
coeffMOP(fx2) ## coef(fx2)
coeffPol(fx2)
```

---

coef.motbf

*Extract the coefficients of an MoTBF*

---

**Description**

Extracts the parameters of the learned mixtures of truncated basis functions.

**Usage**

```
## S3 method for class 'motbf'  
coef(object, ...)
```

**Arguments**

object	An object of class motbf.
...	other arguments.

**Value**

A numeric vector with the parameters of the function.

**See Also**

[univMoTBF](#), [coeffMOP](#) and [coeffMTE](#)

**Examples**

```
## Data  
X <- rchisq(2000, df = 5)  
  
## Learning  
f1 <- univMoTBF(X, POTENTIAL_TYPE = "MOP"); f1  
## Coefficients  
coef(f1)  
  
## Learning  
f2 <- univMoTBF(X, POTENTIAL_TYPE = "MTE", maxParam = 10); f2  
## Coefficients  
coef(f2)  
  
## Learning  
f3 <- univMoTBF(X, POTENTIAL_TYPE = "MOP", nparam=10); f3  
## Coefficients  
coef(f3)  
  
## Plots  
plot(NULL, xlim = range(X), ylim = c(0,0.2), xlab="X", ylab="density")  
plot(f1, xlim = range(X), col = 1, add = TRUE)  
plot(f2, xlim = range(X), col = 2, add = TRUE)  
plot(f3, xlim = range(X), col = 3, add = TRUE)  
hist(X, prob = TRUE, add= TRUE)
```

---

`coef.mte`*Extracting the coefficients of an MTE*

---

**Description**

It extracts the parameters of the learned mixtures of truncated exponential models.

**Usage**

```
coeffMTE(fx)
```

```
coeffExp(fx)
```

**Arguments**

`fx` An "motbf" function of subclass 'mte'.

**Details**

`coeffMOP()` return the coefficients of the terms in the function.

`coeffPol()` returns the coefficients of the potential of the exponential basis in the function.

**Value**

An array with the parameters of the function.

**See Also**

[coef.motbf](#) and [univMoTBF](#)

**Examples**

```
## 1. EXAMPLE
data <- rnorm(1000, mean=5)
fx1 <- univMoTBF(data, POTENTIAL_TYPE = "MTE")
hist(data, prob=TRUE, main="")
plot(fx1, xlim=range(data), col="red", add=TRUE)
coeffMTE(fx1) ## coef(fx1)
coeffExp(fx1)

## 2. EXAMPLE
data <- rexp(1000, rate=1/2)
fx2 <- univMoTBF(data, POTENTIAL_TYPE = "MTE")
hist(data, prob=TRUE, main="")
plot(fx2, xlim=range(data), col="red", add=TRUE)
coeffMTE(fx2) ## coef(fx2)
coeffExp(fx2)
```

---

 coefExpJointCDF

*Degree Function*


---

**Description**

Compute the degree for each term of a joint CDF.

**Usage**

```
coefExpJointCDF(dimensions)
```

**Arguments**

`dimensions` A "numeric" vector including the number of parameters of each variable.

**Value**

A list with `n` element. Each element contains a numeric vector with the degree for each variable and each term of the joint CDF.

**Examples**

```
## Dimension of the joint PDF of 2 variables
dim <- c(4,5)
## Potentials of each term of the CDF
c <- coefExpJointCDF(dim)
length(c) + 1 ## plus 1 because of the constant coefficient

## Dimension of the joint density function of 2 variables
dim <- c(5,5,3)
## Potentials of the cumulative function
coefExpJointCDF(dim)
```

---

 conditionalmotbf.learning

*Learning conditional MoTBF densities*


---

**Description**

Collection of functions used for learning conditional MoTBFs, computing the internal BIC, selecting the parents that get the best BIC value, and other internal functions required to learn the conditional densities.



**Usage**

```
conditionalMethod(  
  data,  
  nameParents,  
  nameChild,  
  numIntervals,  
  POTENTIAL_TYPE,  
  maxParam = NULL,  
  s = NULL,  
  priorData = NULL  
)
```

```
conditional(  
  data,  
  nameParents,  
  nameChild,  
  domainChild,  
  domainParents,  
  numIntervals,  
  mm,  
  POTENTIAL_TYPE,  
  maxParam = NULL,  
  s = NULL,  
  priorData = NULL  
)
```

```
select(  
  data,  
  nameParents,  
  nameChild,  
  domainChild,  
  domainParents,  
  numIntervals,  
  POTENTIAL_TYPE,  
  maxParam = NULL,  
  s = NULL,  
  priorData = NULL  
)
```

```
learn.tree.Intervals(  
  data,  
  nameParents,  
  nameChild,  
  domainParents,  
  domainChild,  
  numIntervals,  
  POTENTIAL_TYPE,  
  maxParam = NULL,
```

```

    s = NULL,
    priorData = NULL
)

```

```
BICscoreMoTBF(conditionalfunction, data, nameParents, nameChild)
```

### Arguments

<code>data</code>	An object of class "data.frame".
<code>nameParents</code>	A "character" vector containing the names of the parent variables.
<code>nameChild</code>	A "character" string containing the name of the child variable.
<code>numIntervals</code>	A positive integer indicating the maximum number of intervals for splitting the domain of the parent variables.
<code>POTENTIAL_TYPE</code>	A "character" string, either <i>MOP</i> or <i>MTE</i> , corresponding to the type of basis function.
<code>maxParam</code>	A positive integer which indicates the maximum number of coefficients in the function. If specified, the output is the function which gets the best BIC with, at most, this number of parameters. By default, it is set to NULL.
<code>s</code>	A "numeric" value indicating the expert's confidence in the prior knowledge. This argument takes values on the interval $[0, N]$ , where $N$ is the sample size, and is used to synchronize the support of the prior knowledge and the sample. By default, it is NULL, and must be modified only if prior information is to be incorporated in the learning process.
<code>priorData</code>	An object of class "data.frame", corresponding to the prior information.
<code>domainChild</code>	A "numeric" vector with the range of the child variable.
<code>domainParents</code>	An object of class "matrix" with the range of the parent variables, or a "numeric" vector if there is only one parent.
<code>mm</code>	One of the inputs and the output of the recursive internal function "conditional".
<code>conditionalfunction</code>	The output of the internal function <code>learn.tree.Intervals</code> .

### Details

The main function, `conditionalMethod()`, fits truncated basis functions for the conditioned variable for each configuration of splits of the parent variables. The domain of the parent variables is splitted in different intervals and univariate functions are fitted in these ranges. The remaining above described functions are internal to the main function.

### Value

The main function `conditionalMethod` returns a list with the name of the parents, the different intervals and the fitted densities

### See Also

[printConditional](#)

**Examples**

```

## Dataset
X <- rnorm(1000)
Y <- rbeta(1000, shape1 = abs(X)/2, shape2 = abs(X)/2)
Z <- rnorm(1000, mean = Y)
data <- data.frame(X = X, Y = Y, Z = Z)

## Conditional Method
parents <- c("X","Y")
child <- "Z"
intervals <- 2

potential <- "MTE"
fMTE <- conditionalMethod(data, nameParents = parents, nameChild = child,
numIntervals = intervals, POTENTIAL_TYPE = potential)
printConditional(fMTE)

#####

potential <- "MOP"
fMOP <- conditionalMethod(data, nameParents = parents, nameChild = child,
numIntervals = intervals, POTENTIAL_TYPE = potential, maxParam = 15)
printConditional(fMOP)

#####

#####
## Internal functions: Not needed to run #####
#####

domainP <- range(data[,parents])
domainC <- range(data[, child])
t <- conditional(data, nameParents = parents, nameChild = child,
domainParents = domainP, domainChild = domainC, numIntervals = intervals,
mm = NULL, POTENTIAL_TYPE = potential)
printConditional(t)
selection <- select(data, nameParents = parents, nameChild = child,
domainParents = domainP, domainChild = domainC, numIntervals = intervals,
POTENTIAL_TYPE = potential)
parent1 <- selection$parent; parent1
domainParent1 <- range(data[,parent1])
treeParent1 <- learn.tree.Intervals(data, nameParents = parent1,
nameChild = child, domainParents = domainParent1, domainChild = domainC,
numIntervals = intervals, POTENTIAL_TYPE = potential)
BICscoreMoTBF(treeParent1, data, nameParents = parent1, nameChild = child)

#####
#####

```

**Description**

Collection of functions for discretizing, standardizing, converting factors to characters and other useful methods for pre-processing datasets.

**Usage**

```
whichDiscrete(dataset, discreteVariables)

discreteVariables_as.character(dataset, discreteVariables)

standardizeDataset(dataset)

discretizeVariablesEWDIS(dataset, numIntervals, factor = FALSE, binary = FALSE)

discreteVariablesStates(namevariables, discreteData)

nstates(DiscreteVariablesStates)

quantileIntervals(X, numIntervals)

scaleData(dataset, scale)
```

**Arguments**

dataset	A dataset of class "data.frame". The variables of the dataset can be discrete and continuous.
discreteVariables	A "character" array with the names of the discrete variables.
numIntervals	Number of bins used to discretize the continuous variables.
factor	A boolean value indicating if the variables should be considered as "factor" or as "character". By default it is set to FALSE.
binary	By default it is set to FALSE, indicating that only binary entries are used for continuous variables; a TRUE value means that binary entries are used to discretize the full dataset taking into account the states the discrete variables.
namevariables	an array with the names of the variables.
discreteData	A discretized dataset of class "data.frame".
DiscreteVariablesStates	The output of the function discreteVariablesStates.
X	A "numeric" vector with the data values of a continuous variable.
scale	A "numeric" vector (when it refers to a single variable) or a "list" containing the name(s) of the variable(s) and the scale value.

**Details**

whichDiscrete() selects the position of the discrete variables.

discreteVariables\_as.character() transforms the values of the discrete variables into character values.

standardizeDataset() standardizes all the variables in a data set.

discretizeVariablesEWdis() discretizes the continuous variables in a dataset using equal width binning.

discreteVariablesStates() extracts the states of the qualitative variables.

nstates() computes the number of different values of the discrete variables.

quantileIntervals() gets the quantiles of a variable taking into account the number of intervals into which its domain is splitted.

## Examples

```
## dataset: 2 continuous variables, 1 discrete variable.
data <- data.frame(X = rnorm(100), Y = rexp(100, 1/2), Z = as.factor(rep(c("s", "a"), 50)))
disVar <- "Z" ## Discrete variable
class(data[,disVar]) ## factor

data <- discreteVariables_as.character(dataset = data, discreteVariables = disVar)
class(data[,disVar]) ## character

whichDiscrete(dataset = data, discreteVariables = "Z")

standData <- standardizeDataset(dataset = data)

disData <- discretizeVariablesEWdis(dataset = data, numIntervals = 3)

l <- discreteVariablesStates(namevariables = names(data), discreteData = disData)

nstates(DiscreteVariablesStates = l)

## Continuous variables
quantileIntervals(X = data[,1], numIntervals = 4)
quantileIntervals(X = data[,2], numIntervals = 10)
```

---

derivMOP

*Derivative of a MOP*

---

## Description

Compute the derivative of an "motbf" object with 'mop' subclass.

## Usage

```
derivMOP(fx)
```

## Arguments

fx                    An "motbf" object of the 'mop' subclass.

**Value**

The derivative which is also an "motbf" function.

**See Also**

[univMoTBF](#) for learning and [derivMoTBF](#) for general "motbf" models.

**Examples**

```
## 1. EXAMPLE
X <- rexp(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MOP")
derivMOP(Px)

## 2. EXAMPLE
X <- rnorm(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MOP")
derivMOP(Px)

## Not run:
## 3. EXAMPLE
X <- rnorm(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MTE")
derivMOP(Px)
## Error in derivMOP(Px): fx is an 'motbf' function but not 'mop' subclass.
class(Px)
subclass(Px)

## End(Not run)
```

---

derivMoTBF

*Derivating MoTBFs*

---

**Description**

Compute the derivative of a one-dimensional mixture of truncated basis function.

**Usage**

```
derivMoTBF(fx)
```

**Arguments**

fx                    An object of class "motbf".

**Value**

The derivative of the MoTBF function, which is also an object of class "motbf".

**See Also**

[univMoTBF](#), [derivMOP](#) and [derivMTE](#)

**Examples**

```
## 1. EXAMPLE
X <- rexp(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MOP")
derivMoTBF(Px)

## 2. EXAMPLE
X <- rnorm(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MOP")
derivMoTBF(Px)

## 3. EXAMPLE
X <- rchisq(1000, df = 3)
Px <- univMoTBF(X, POTENTIAL_TYPE="MTE")
derivMoTBF(Px)

## Not run:
## 4. EXAMPLE
Px <- "x+2"
class(Px)
derivMoTBF(Px)
## Error in derivMoTBF(Px): "fx is not an 'motbf' function."

## End(Not run)
```

---

derivMTE

*Derivating MTEs*

---

**Description**

Compute the derivative of an "motbf" object with 'mte' subclass.

**Usage**

```
derivMTE(fx)
```

**Arguments**

fx                    An "motbf" object of the 'mte' subclass.

**Value**

The derivative which is also an "motbf" function.

**See Also**

[univMoTBF](#) for learning and [derivMoTBF](#) for general "motbf" models.

**Examples**

```
## 1. EXAMPLE
X <- rexp(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MTE")
derivMTE(Px)

## 2. EXAMPLE
X <- rnorm(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MTE")
derivMTE(Px)

## Not run:
## 3. EXAMPLE
X <- rnorm(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MOP")
derivMTE(Px)
## Error in derivMTE(Px): fx is an 'motbf' function but not 'mte' subclass.
class(Px)
subclass(Px)

## End(Not run)
```

---

dimensionFunction      *Dimension of MoTBFs*

---

**Description**

Get the dimension of "motbf" and "jointmotbf" densities.

**Usage**

```
dimensionFunction(P)
```

**Arguments**

P                    An object of class "motbf" and subclass 'mop' or "jointmotbf".

**Value**

Dimension of the function.

**See Also**

[univMoTBF](#) and [jointMoTBF](#)



## Examples

```
## 1. EXAMPLE
## Data
X <- rnorm(2000)

## Univariate function
subclass <- "MOP"
f <- univMoTBF(X, POTENTIAL_TYPE = subclass)
dimensionFunction(f)

## 2. EXAMPLE
## Dataset with 2 variables
X <- data.frame(rnorm(100), rnorm(100))

## Joint function
dim <- c(2,3)
param <- parametersJointMoTBF(X, dimensions = dim)
P <- jointMoTBF(param)

## Dimension of the joint function
dimensionFunction(P)
```

---

discreteStatesFromBN *Get the states of all discrete nodes from a MoTFB-BN*

---

## Description

This function returns the states of all discrete nodes from a list obtained from [MoTBFs\\_Learning](#).

## Usage

```
discreteStatesFromBN(bn, dag)
```

## Arguments

**bn** A list of lists obtained from [MoTBFs\\_Learning](#).  
**dag** A network of class "bn".

## Value

discreteStatesFromBN returns a list of length equal to the number of discrete nodes in the network. Each element of the list corresponds to a node and contains a character vector indicating the states of the node.

**Examples**

```
## Create a dataset
# Continuous variables
x <- rnorm(100)
y <- rnorm(100)

# Discrete variable
z <- sample(letters[1:2],size = 100, replace = TRUE)

data <- data.frame(C1 = x, C2 = y, D1 = z, stringsAsFactors = FALSE)

## Get DAG
dag <- LearningHC(data)

## Learn a BN
bn <- MoTBFs_Learning(dag, data, POTENTIAL_TYPE = "MTE")

## Get the states of the discrete nodes

discreteStatesFromBN(bn, dag)
```

---

 ecoli

---

*Data set Ecoli: Protein Localization Sites*


---

**Description**

This data set contains information of Escherichia coli. It is a bacterium of the genus Escherichia that is commonly found in the lower intestine of warm-blooded organism.

**Format**

A data frame with 336 rows, 8 variables and the class.

**Details**

**Sequence Name** Accession number for the SWISS-PROT database.

**mcbg** McGeoch's method for signal sequence recognition.

**gvh** Von Heijne's method for signal sequence recognition.

**lip** Von Heijne's Signal Peptidase II consensus sequence score. Binary attribute.

**chg** Presence of charge on N-terminus of predicted lipoproteins. Binary attribute.

**aac** Score of discriminant analysis of the amino acid content of outer membrane and periplasmic proteins.

**alm1** Score of the ALOM membrane spanning region prediction program.

**alm2** Score of ALOM program after excluding putative cleavable signal regions from the sequence.

**Class** Class variable. 8 possible states.

**Source**

<http://archive.ics.uci.edu/ml/datasets/Ecoli>

---

evalJointFunction      *Evaluation of joint MoTBFs*

---

**Description**

Evaluates a "jointmotbf" object at a specific point.

**Usage**

```
evalJointFunction(P, values)
```

**Arguments**

P                    A "jointmotbf" object.  
 values              A list with the name of the variables equal to the values to be evaluated.

**Value**

If all the variables in the equation are evaluated then a "numeric" value is returned. Otherwise, an "motbf" object or a "jointmotbf" object is returned.

**Examples**

```
#' ## 1. EXAMPLE
## Dataset with 2 variables
X <- data.frame(rnorm(100), rexp(100))

## Joint function
dim <- c(3,3) # dim <- c(5,4)
param <- parametersJointMoTBF(X, dimensions = dim)
P <- jointMoTBF(param)
P

## Evaluation
nVariables(P)
val <- list(x = -1.5, y = 3)
evalJointFunction(P, values = val)
val <- list(x = -1.5)
evalJointFunction(P, values = val)
val <- list(y = 3)
evalJointFunction(P, values = val)

#####
## MORE EXAMPLES #####
#####
```

```
## Dataset with 3 variables
X <- data.frame(rnorm(100), rexp(100), rnorm(100, 1))

## Joint function
dim <- c(2,1,3)
param <- parametersJointMoTBF(X, dimensions = dim)
P <- jointMoTBF(param)
P

## Evaluation
nVariables(P)
val <- list(x = 0.8, y = -2.1, z = 1.2)
evalJointFunction(P, values = val)
val <- list(x = 0.8, z = 1.2)
evalJointFunction(P, values = val)
val <- list(y = -2.1)
evalJointFunction(P, values = val)
val <- list(y = -2.1)
evalJointFunction(P, values = val)
```

---

findConditional

*Find fitted conditional MoTBFs*


---

## Description

This function returns the conditional probability function of a node given an MoTBF-bayesian network and the value of its parents.

## Usage

```
findConditional(node, bn, evi = NULL)
```

## Arguments

node	A character string, representing the target variable.
bn	A list of lists obtained from <a href="#">MoTBFs_Learning</a> , containing the conditional functions.
evi	A data.frame of dimension '1xn' that contains the values of the 'n' parents of the target node. This argument can be NULL if "node" is a root node.

## Value

A list containing the conditional distribution of the target variable.

**Examples**

```
## Dataset
data("ecoli", package = "MoTBFs")
data <- ecoli[,-c(1,9)]

## Get directed acyclic graph
dag <- LearningHC(data)

## Learn bayesian network
bn <- MoTBFs_Learning(dag, data = data, numIntervals = 4, POTENTIAL_TYPE = "MTE")

## Specify the evidence set and node of interest
evi <- data.frame(lip = "0.48", alm1 = 0.55, gvh = 1, stringsAsFactors=FALSE)
node = "alm2"

## Get the conditional distribution
findConditional(node, bn, evi)
```

---

forward_sampling	<i>Forward Sampling</i>
------------------	-------------------------

---

**Description**

`forward_sampling()` returns the conditional distribution of a target variable given a set of observed variables. The forward sampling algorithm approximates the conditional distribution from a random sample.

**Usage**

```
forward_sampling(bn, dag, target, evi, size, ...)
```

**Arguments**

<code>bn</code>	A list of lists obtained from the function <a href="#">MoTBFs_Learning</a> .
<code>dag</code>	An object of class "bn", representing the directed acyclic graph.
<code>target</code>	A character string equal to the name of the variable of interest.
<code>evi</code>	A <code>data.frame</code> containing the observed variables.
<code>size</code>	A positive integer giving the number of instances to be generated.
<code>...</code>	Optional arguments passed on to the <a href="#">univMoTBF</a> function. <code>evalRange</code> , <code>nparam</code> and <code>maxParam</code> can be specified. <code>POTENTIAL_TYPE</code> is taken from the 'bn' object.

**Value**

A list containing the conditional distribution of the target variable and a `data.frame` with the generated sample.

## References

Henrion, M. (1988). Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In Machine Intelligence and Pattern Recognition (Vol. 5, pp. 149-163). North-Holland.

## Examples

```
## Dataset
data("ecoli", package = "MoTBFS")
data <- ecoli[,-c(1,9)]

## Get directed acyclic graph
dag <- LearningHC(data)

## Learn bayesian network
bn <- MoTBFS_Learning(dag, data = data, numIntervals = 4, POTENTIAL_TYPE = "MTE")

## Specify the evidence set and target variable
obs <- data.frame(lip = "0.48", alm1 = 0.55, gvh = 1, stringsAsFactors=FALSE)
node <- "alm2"

## Get the conditional distribution of 'node' and the generated sample
forward_sampling(bn, dag, target = node, evi = obs, size = 10, maxParam = 15)
```

---

```
generateNormalPriorData
```

*Prior data generation*

---

## Description

Generate a prior dataset taking in to account the relationships between the variables in a given network.

## Usage

```
generateNormalPriorData(graph, data, size, means, deviations = NULL)
```

## Arguments

graph	A network of the class "bn", "graphNEL" or "network".
data	An object of class "data.frame" containing the continuous variables in the dataset.
size	A positive integer indicating the number of records to generate for each variable in the dataset.
means	A "numeric" vector with the average of the variables whose prior information is available. The names in the vector must be the same as the names of the variables in the data.frame.

deviations      A "numeric" vector with the standard deviations of the variables whose prior information is available. The names of the vector must be the same as the names of the variables in the data.frame. If not specified, the standard deviation of each variable is computed from 'data'.

### Value

A normal prior data set of class "data.frame".

### See Also

[rnormMultiv](#)

### Examples

```
## Data
data(ecoli)
data <- ecolli[,-c(1,9)] ## remove sequece.name and class
X <- TrainingandTestData(data, percentage_test = 0.95)
Xtraining <- X$Training
Xtest <- X$Test

## DAG
dag <- LearningHC(data)
plot(dag)

## Means and desviations
colnames(data)

m <- sapply(data, function(x){ifelse(is.numeric(x), mean(x),NA)})
d <- sapply(data, function(x){ifelse(is.numeric(x), sd(x),NA)})

## Prior Dataset
n <- 5600
priorData <- generateNormalPriorData(dag, data = Xtraining, size = n, means = m)
summary(priorData)
ncol(priorData)
nrow(priorData)
class(priorData)
```

---

getChildParentsFromGraph

*Get the list of relations in a graph*

---

### Description

Compute the parents of each variable in the graph.

**Usage**

```
getChildParentsFromGraph(graph, nameVars = NULL)
```

**Arguments**

`graph` A directed acyclic graph of the class "graphNEL", "network" or "bn".

`nameVars` A character array containing the names of the variables in the graph. This parameter is only used when graph is of class "network".

**Value**

A list where each element is a vector containing the name of a variable and its parents in the graph.

**Examples**

```
## Data
data(ecoli)
ecoli <- ecol[,,-1] ## Sequence Name

## DAG1
dag1 <- LearningHC(ecoli)
dag1
plot(dag1)
getChildParentsFromGraph(dag1)

## DAG2
dag2 <- LearningHC(ecoli, numIntervals = 10)
dag2
plot(dag2)
getChildParentsFromGraph(dag2)
```

---

getCoefficients      *Get the coefficients*

---

**Description**

Compute the coefficients for the linear opinion pool

**Usage**

```
getCoefficients(fPI, rangeNewPriorData, fD, data, domain, coeffversion)
```



**Arguments**

fPI	The function fitted to the prior data, of class "motbf".
rangeNewPriorData	An array of length 2 with the new domain of the prior function.
fD	The function fitted to the original data, of class "motbf".
data	A "numeric" array which contains the sample.
domain	A "numeric" array with the domain of the data density function.
coeffversion	A "numeric" value between 1--4 which contains the used version for computing the coefficients in the linear opinion pool to combine the prior function and the data function. By default coeffversion = "4" is used, so the combination depends on the goodness of the model versus another random positive MoTBF model.

**Details**

coeffversion can be: "1" coef1 and coef2 are the sum of the probabilities of one of the function over the sum of all probabilities, respectively; "2" coef1 and coef2 are the solution of a linear optimization problem which tries to maximize the sum 1 for each row of probabilities; "3" coef1 and coef2 are the difference of the log-likelihood of the evaluated model and a random uniform model over the sum of both differences, respectively; "4" coef1 and coef2 are the difference of the log-likelihood of the evaluated model and a random positive MoTBF model over the sum of both differences, respectively.

**Value**

A "numeric" value of length 2 giving the coefficients which are the weight of the two function to combine.

**See Also**

[learnMoTBFpriorInformation](#)

**Examples**

```
## Data
X <- rnorm(15)

## Prior Data
priordata <- rnorm(5000)

## Learning
confident <- 5
type <- "MOP"
f <- learnMoTBFpriorInformation(priorData = priordata, data = X, s = confident,
POTENTIAL_TYPE = type)
attributes(f)

## Coefficients: linear opinion pool
getCoefficients(fPI = f$priorFunction, rangeNewPriorData = f$domain, fD = f$dataFunction,
```

```

data = X, domain = range(X), coeffversion = 4)

getCoefficients(fPI = f$priorFunction, rangeNewPriorData = f$domain, fD = f$dataFunction,
data = X, domain = range(X), coeffversion = 1)

getCoefficients(fPI = f$priorFunction, rangeNewPriorData = f$domain, fD = f$dataFunction,
data = X, domain = range(X), coeffversion = 3)

getCoefficients(fPI = f$priorFunction, rangeNewPriorData = f$domain, fD = f$dataFunction,
data = X, domain = range(X), coeffversion = 2)

```

---

```

getNonNormalisedRandomMoTBF
      Random MoTBF

```

---

**Description**

Generates a non normalized (i.e. not integrating to 1) positive MoTBF function.

**Usage**

```
getNonNormalisedRandomMoTBF(degree, POTENTIAL_TYPE = "MOP")
```

**Arguments**

**degree** A "numeric" value containing the degree of the random function.

**POTENTIAL\_TYPE** A "character" string specifying the possibles potential types, must be one of "MOP" or "MTE".

**Value**

A "numeric" vector of length 2 giving the coefficients.

**Examples**

```

getNonNormalisedRandomMoTBF(8, POTENTIAL_TYPE = "MOP")
getNonNormalisedRandomMoTBF(11, POTENTIAL_TYPE = "MTE")

```

---

goodnessDiscreteVariables  
*BIC score and log-likelihood*

---

### Description

Compute the loglikelihood and the BIC score for discrete models, i.e multinomial Bayesian Networks.

### Usage

```
getlogLikelihoodDiscreteBN(discreteBN)

getBICDiscreteBN(discreteBN, sameData = FALSE)
```

### Arguments

`discreteBN` A list of multiples lists. Each list contains two entries, the probabilities and the size of the data which is in each leaf of the discrete tree.

`sameData` A logical argument; FALSE means that different datasets were used for learning.

### Value

The loglikelihood and the BIC score of the discrete network.

### Examples

```
## 1. EXAMPLE
## Discrete data
X <- rep(c("yes", "no", "maybe"), 500)
Y <- rep(c("M", "F"), 750)
data <- data.frame(X=X, Y=Y)
disVar <- c("X", "Y")
data <- discreteVariables_as.character(data, discreteVariables=disVar)
n <- nrow(data)

## Probabilities
s <- discreteVariablesStates(namevariables=disVar, discreteData=data)
p <- lapply(1:length(s), function(i) probDiscreteVariable(stateNames=
s[[i]]$states, Variable=data[,i]))

## Log-likelihood
getlogLikelihoodDiscreteBN(p)

## BIC
getBICDiscreteBN(p, sameData = TRUE)

## 2. EXAMPLE
## Discrete variables
```

```

X <- rep(c("1", "2", "3"), 500)
data <- data.frame(X=as.character(X))
s <- discreteVariablesStates(namevariables="X", discreteData=data)
p1 <- probDiscreteVariable(stateNames = s[[1]]$states, Variable = data[,1])

Y <- rep(c("YES", "NO"), 100)
data <- data.frame(Y = as.character(Y))
s <- discreteVariablesStates(namevariables = "Y", discreteData = data)
p2 <- probDiscreteVariable(stateNames = s[[1]]$states, Variable = data[,1])
## Probabilities
P <- list(p1,p2)

## Log-likelihood
getlogLikelihoodDiscreteBN(P)

## BIC
getBICDiscreteBN(P, sameData = TRUE)

```

---

goodnessMoTBFBN

*BIC of a hybrid BN*


---

### Description

Compute the BIC score and the loglikelihood from the fitted MoTBFs functions in a hybrid Bayesian network.

### Usage

```
logLikelihood.MoTBFBN(MoTBF.BN, data)
```

```
BiC.MoTBFBN(MoTBF.BN, data)
```

### Arguments

MoTBF.BN      The output of the 'MoTBF\_Learning' method.

data            The dataset of class data.frame.

### Value

A numeric value giving the log-likelihood of the BN.

### See Also

[MoTBFs\\_Learning](#)

**Examples**

```

## Dataset Ecoli
require(MoTBFs)
data(ecoli)
data <- ecoli[,-c(1)] ## remove variable sequence

## Directed acyclic graph
dag <- LearningHC(data)

## Learning BN
intervals <- 3
potential <- "MOP"
P1 <- MoTBFs_Learning(graph = dag, data = data, POTENTIAL_TYPE=potential,
numIntervals = intervals, maxParam = 5)
logLikelihood.MoTBFBN(P1, data) ##BIC$LogLikelihood
BIC <- BiC.MoTBFBN(P1, data)
BIC$BIC

## Learning BN
intervals <- 2
potential <- "MTE"
P2 <- MoTBFs_Learning(graph = dag, data = data, POTENTIAL_TYPE=potential,
numIntervals = intervals, maxParam = 10)
logLikelihood.MoTBFBN(P2, data) ##BIC$LogLikelihood
BIC <- BiC.MoTBFBN(P2, data)
BIC$BIC

```

---

integralJointMoTBF      *Integration with MoTBFs*

---

**Description**

Integrate a "jointmotbf" object over an non defined domain. It is able to get the integral of a joint function over a set of variables or over all the variables in the function.

**Usage**

```
integralJointMoTBF(P, var = NULL)
```

**Arguments**

P	A "jointmotbf" object.
var	A "character" vector containing the name of the variables that will be integrated out. Instead of the names, the position of the variables can be given. By default it's NULL then all the variables are integrated out.

**Value**

A multiintegral of a joint function of class "jointmotbf".

**Examples**

```
## 1. EXAMPLE
## Dataset with 2 variables
X <- data.frame(rnorm(100), rnorm(100))

## Joint function
dim <- c(2,3)
param <- parametersJointMoTBF(X, dimensions = dim)
P <- jointMoTBF(param)

## Integral
integralJointMoTBF(P)
integralJointMoTBF(P, var="x")
integralJointMoTBF(P, var="y")

#####
## MORE EXAMPLES #####
#####

## Dataset with 3 variables
X <- data.frame(rnorm(50), rnorm(50), rnorm(50))

## Joint function
dim <- c(2,1,3)
param <- parametersJointMoTBF(X, dimensions = dim)
P <- jointMoTBF(param)

## Integral
integralJointMoTBF(P)
integralJointMoTBF(P, var="x")
integralJointMoTBF(P, var="y")
integralJointMoTBF(P, var="z")
integralJointMoTBF(P, var=c("x","z"))
```

---

integralMOP

*Integration of MOPs*


---

**Description**

Method to calculate the non-defined integral of an "motbf" object of 'mte' subclass.

**Usage**

```
integralMOP(fx)
```

**Arguments**

fx                    An "motbf" object of subclass 'mop'.

**Value**

The non-defined integral of the function.

**See Also**

[univMoTBF](#) for learning and [integralMoTBF](#) for a more complete function to get defined and non-defined integrals of class "motbf".

**Examples**

```
## 1. EXAMPLE
X <- rexp(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MOP")
integralMOP(Px)

## 2. EXAMPLE
X <- rnorm(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MOP")
integralMOP(Px)

## Not run:
## 3. EXAMPLE
X <- rnorm(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MTE")
integralMOP(Px)
## Error in integralMOP(Px): fx is an 'motbf' function but not 'mop' subclass.
class(Px)
subclass(Px)

## End(Not run)
```

---

integralMoTBF

*Integrating MoTBFs*


---

**Description**

Compute the integral of a one-dimensional mixture of truncated basis function over a bounded or unbounded interval.

**Usage**

```
integralMoTBF(fx, min = NULL, max = NULL)
```

**Arguments**

fx	An object of class "motbf".
min	The lower integration limit. By default it is NULL.
max	The upper integration limit. By default it is NULL.

## Details

If the limits of the interval, min and max are NULL, then the output is the expression of the indefinite integral. If only 'min' contains a numeric value, then the expression of the integral is evaluated at this point.

## Value

integralMoTBF() returns either the indefinite integral of the MoTBF function, which is also an object of class "motbf", or the definite integral, which is a "numeric" value.

## See Also

[univMoTBF](#), [integralMOP](#) and [integralMTE](#)

## Examples

```
## 1. EXAMPLE
X <- rexp(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MOP")
integralMoTBF(Px)
integralMoTBF(Px, 1.2)
integralMoTBF(Px, min(X), max(X))

## 2. EXAMPLE
X <- rnorm(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MOP")
iP <- integralMoTBF(Px); iP
plot(iP, xlim=range(X))
integralMoTBF(Px, 0.2)
integralMoTBF(Px, min(X), max(X))

## 3. EXAMPLE
X <- rchisq(1000, df = 3)
Px <- univMoTBF(X, POTENTIAL_TYPE="MTE")
integralMoTBF(Px)
integralMoTBF(Px, 1)
integralMoTBF(Px, min(X), max(X))

## Not run:
## 4. EXAMPLE
Px <- "1+x+5"
class(Px)
integralMoTBF(Px)
## Error in integralMoTBF(Px): "fx is not an 'motbf' function."

## End(Not run)
```



---

`integralMTE`*Integrating MTEs*

---

**Description**

Method to calculate the non-defined integral of an "motbf" object of 'mte' subclass.

**Usage**

```
integralMTE(fx)
```

**Arguments**

`fx` An "motbf" object of subclass 'mte'.

**Value**

The non-defined integral of the function.

**See Also**

[univMoTBF](#) for learning and [integralMoTBF](#) for a more complete function to get defined and non-defined integrals of class "motbf".

**Examples**

```
## 1. EXAMPLE
X <- rexp(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MTE")
integralMTE(Px)

## 2. EXAMPLE
X <- rnorm(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MTE")
integralMTE(Px)

## Not run:
## 3. EXAMPLE
X <- rnorm(1000)
Px <- univMoTBF(X, POTENTIAL_TYPE="MOP")
integralMTE(Px)
## Error in integralMTE(Px): fx is an 'motbf' function but not 'mte' subclass.
class(Px)
subclass(Px)

## End(Not run)
```

---

`is.discrete`*Check discreteness of a node*

---

**Description**

This function allows to check whether a node is discrete or not

**Usage**

```
is.discrete(node, bn)
```

**Arguments**

`node` A character (name of node) or numeric (index of node in the bn list) input.  
`bn` A list of lists obtained from [MoTBFs\\_Learning](#).

**Value**

`is.discrete` returns TRUE or FALSE depending on whether the node is discrete or not.

**Examples**

```
## Create a dataset
# Continuous variables
x <- rnorm(100)
y <- rnorm(100)

# Discrete variable
z <- sample(letters[1:2],size = 100, replace = TRUE)

data <- data.frame(C1 = x, C2 = y, D1 = z, stringsAsFactors = FALSE)

## Get DAG
dag <- LearningHC(data)

## Learn BN
bn <- MoTBFs_Learning(dag, data, POTENTIAL_TYPE = "MTE")

## Check wheter a node is discrete or not

# Using its name
is.discrete("D1", bn)

# Using its index position
is.discrete(3, bn)
```

---

is.observed	<i>Observed Node</i>
-------------	----------------------

---

**Description**

is.observed() checks whether a node belongs to the evidence set or not.

**Usage**

```
is.observed(node, evi)
```

**Arguments**

node	A character string, matching the node's name.
evi	A data.frame of the evidence set.

**Value**

This function returns TRUE if "node" is included in "evi", or, otherwise, FALSE.

**Examples**

```
## Data frame of the evidence set
obs <- data.frame(lip = "1", alm2 = 0.5, stringsAsFactors=FALSE)

## Check if x is in obs
is.observed("x", obs)
```

---

is.root	<i>Root nodes</i>
---------	-------------------

---

**Description**

is.root checks whether a node has parents or not.

**Usage**

```
is.root(node, dag)
```

**Arguments**

node	A character string indicating the name of the node.
dag	An object of class "bn".

**Value**

is.root returns TRUE or FALSE depending on whether the node is root or not.

**Examples**

```
## Create a dataset
# Continuous variables
x <- rnorm(100)
y <- rnorm(100)

# Discrete variable
z <- sample(letters[1:2],size = 100, replace = TRUE)

data <- data.frame(C1 = x, C2 = y, D1 = z, stringsAsFactors = FALSE)

## Get DAG
dag <- LearningHC(data)

## Check if a node is root
is.root("C1", dag)
```

---

jointCDF

*Joint MoTBFs CDFs*

---

**Description**

Function to compute multivariate CDFs.

**Usage**

```
jointCDF(df, grid)
```

**Arguments**

df	The dataset as an object of class <code>data.frame</code> .
grid	a <code>data.frame</code> with the selected data points where the objective function will be evaluated when optimizing the parameters.

**Value**

`jointCDF()` returns a vector.

**Examples**

```
## Create dataset with 2 variables
n = 2
size = 50
df <- as.data.frame(matrix(round(rnorm(size*n),2), ncol = n))

## Create grid dataset
npointsgrid <- 10
ranges <- sapply(df, range)
eg <- list()
```

```

for(i in 1: ncol(df)){
  eg[[i]] <- seq(ranges[1,i], ranges[2,i], length.out = npointsgrid)
}

x <- expand.grid(eg)

## Joint cumulative values
jointCDF(df = df, grid = x)

```

---

jointmotbf.learning     *Joint MoTBF density learning*

---

## Description

Two functions for learning joint MoTBFs. The first one, `parametersJointMoTBF()`, gets the parameters by solving a quadratic optimization problem, minimizing the mean squared error between the empirical joint CDF and the estimated CDF. The density is obtained as the derivative of the estimated CDF. The second one, `jointMoTBF()`, fixes the equation of the joint function using the previously learned parameters and converting this "character" string into an object of class "jointmotbf".

## Usage

```

parametersJointMoTBF(X, ranges = NULL, dimensions = NULL)

jointMoTBF(object)

```

## Arguments

<code>X</code>	A dataset of class "data.frame".
<code>ranges</code>	A "numeric" matrix containing the range of the variables used to fit the function, where each column corresponds to a variable. If not specified, the range of each variable is computed from the data.
<code>dimensions</code>	A "numeric" vector containing the number of parameters of each variable.
<code>object</code>	A list with the output of the function <code>parametersJointMoTBF()</code> .

## Value

`parametersJointMoTBF()` returns a list with the following elements: **Parameters**, which contains the computed coefficients of the resulting function; **Dimension**, which is a "numeric" vector containing the number of coefficients used for each variable; **Range** contains a "numeric" matrix with the domain of each variable, by columns; **Iterations** contains the number of iterations needed to solve the problem; **Time** contains the execution time.

`jointMoTBF()` returns an object of class "jointmotbf", which is a list whose only visible element is the analytical expression of the learned density. It also contains the other aforementioned elements, which can be retrieved using `attributes()`

**Examples**

```

## 1. EXAMPLE
## Generate a multinormal dataset
data <- data.frame(X1 = rnorm(100), X2 = rnorm(100))

## Joint learnings
dim <- c(2,3)
param <- parametersJointMoTBF(X = data, dimensions = dim)

param$Parameters
length(param$Parameters)
param$Dimension
param$Range

P <- jointMoTBF(param)
P
attributes(P)
class(P)

#####
## MORE EXAMPLES #####
#####

## Generate a dataset
data <- data.frame(X1 = rnorm(100), X2 = rnorm(100), X3 = rnorm(100))

## Joint learnings
dim <- c(3,2,3)
param <- parametersJointMoTBF(X = data, dimensions = dim)

param$Parameters
length(param$Parameters) ## prod(dim)
param$Dimension
param$Range
param$Time

P <- jointMoTBF(param)
P
attributes(P)
class(P)

```

**Description**

Learn the structure of a hybrid Bayesian network using the **hill climbing** local search method.

**Usage**

```
LearningHC(dataset, numIntervals = NULL)
```

**Arguments**

dataset	A dataset with discrete and continuous variables. If the discrete variables are not of class "factor", they are automatically converted.
numIntervals	A "numeric" value indicating the number of categories used when discretizing a continuous variable, corresponding to intervals of equal width. By default it is NULL, meaning that the continuous variables are not discretized.

**Details**

LearningHC() automatically converts non-numeric variables into factors before calling function hc() from the bnlearn package. LearningHC() can also be used to discretize the dataset, using the equal width method, before calling hc().

**Value**

The output is a "bn" object containing the learned graph.

**See Also**

[hc](#)

**Examples**

```
## Data
data(ecoli)
ecoli <- ecol[, -1] ## Sequence Name

## DAG1
dag1 <- LearningHC(ecoli)
dag1
plot(dag1)

## DAG2
dag2 <- LearningHC(ecoli, numIntervals = 10)
dag2
plot(dag2)
```

---

 learnMoTBFpriorInformation

*Incorporating prior knowledge in the estimation process*


---

## Description

Learns a univariate MoTBF function using prior information.

## Usage

```
learnMoTBFpriorInformation(
  priorData,
  data,
  s,
  POTENTIAL_TYPE,
  domain = range(data),
  coeffversion = 4,
  restrictDomain = TRUE,
  maxParam = NULL
)
```

## Arguments

priorData	A "numeric" vector which contains the prior information.
data	A "numeric" vector containing the observed data.
s	A "numeric" value which specifies the expert confidence in the prior knowledge. This argument takes values on the interval $[0, N]$ , where $N$ is the sample size, and is used to synchronize the support of the prior knowledge and the sample.
POTENTIAL_TYPE	A "character" string, either <i>MOP</i> or <i>MTE</i> , corresponding to the type of basis function.
domain	A "numeric" vector which contains the bounding values to fit the function. By default, it is the range of the data.
coeffversion	A "numeric" value between 1--4 which contains the used version for computing the coefficients of the linear opinion pool to combine the prior function and the data function. By default, <code>coeffversion = "4"</code> is used, so the combination depends on the goodness of the model versus another random model.
restrictDomain	A logical value. This argument allows to choose if the domain is used joining both domains, the prior one and the data domain or trimming them. By default, TRUE is used, so the domain will be trimmed.
maxParam	A positive integer which indicates the maximum number of coefficients in the function. If specified, the output is the function which gets the best BIC with, at most, this number of parameters. By default, it is set to NULL.



**Value**

A list with the elements

coeffs            An "numeric" vector with the two coefficients of the linear opinion pool

posteriorFunction        The final function after combining.

priorFunction    The fit of the prior data.

dataFunction     The fit of the original data.

rangeNewPriorData      A "numeric" vector which contains the final domain where the functions are defined.

**See Also**

[getCoefficients](#)

**Examples**

```
## Data
X <- rnorm(15)

## Prior Data
priordata <- rnorm(5000)

## Test data
test <- rnorm(1000)
testData <- test[test>=min(X)&test<=max(X)]

## Learning
type <- "MOP"
confident <- 3 ## confident <- 1,2,...,length(X)
f <- learnMoTBFpriorInformation(priorData = priordata, data = X, s = confident,
POTENTIAL_TYPE = type)
attributes(f)

## Log-likelihood
sum(log(as.function(f$dataFunction)(testData)))
sum(log(as.function(f$posteriorFunction)(testData))) ## best loglikelihood
```

---

marginalJointMoTBF      *Marginalization of MoTBFs*

---

**Description**

Computes the marginal densities from a "jointmotbf" object.

**Usage**

```
marginalJointMoTBF(P, var)
```

**Arguments**

P                    An object of class "jointmotbf", i.e., the joint density function.  
var                   The "numeric" position or the "character" name of the marginal variable.

**Value**

The marginal of a "jointmotbf" function. The result is an object of class "motbf".

**See Also**

[jointMoTBF](#) and [evalJointFunction](#)

**Examples**

```
## 1. EXAMPLE
## Dataset with 2 variables
X <- data.frame(rnorm(100), rnorm(100))

## Joint function
dim <- c(4,3)
param <- parametersJointMoTBF(X, dimensions = dim)
P <- jointMoTBF(param)
P

## Marginal
marginalJointMoTBF(P, var = "x")
marginalJointMoTBF(P, var = 2)

#####
## MORE EXAMPLES #####
#####

## Generate a dataset with 3 variables
data <- data.frame(rnorm(100), rnorm(100), rnorm(100))

## Joint function
dim <- c(2,1,3)
param <- parametersJointMoTBF(data, dimensions = dim)
P <- jointMoTBF(param)
nVariables(P)

## Marginal
marginalJointMoTBF(P, var="x")
marginalJointMoTBF(P, var="y")
marginalJointMoTBF(P, var="z")
```

**Description**

These functions fit mixtures of polynomials (MOPs). Least square optimization is used to minimize the quadratic error between the empirical cumulative distribution and the estimated one.

**Usage**

```
mop.learning(X, nparam, domain)
```

```
bestMOP(X, domain, maxParam = NULL)
```

**Arguments**

X	A "numeric" data vector.
nparam	Number of parameters of the function.
domain	A "numeric" containing the interval over which the function is defined.
maxParam	A "numeric" value indicating the maximum number of coefficients in the function. By default it is NULL, which means that the number of parameter is not limited. The output is the function which gets the best BIC (with at most maxParam parameters if not NULL).

**Details**

`mop.learning()`: The returned value `$Function` is the only visible element which contains the algebraic expression. Using [attributes](#) the name of the others elements are shown and also they can be extracted with `$`. The [summary](#) of the function also shows all these elements.

`bestMOP()`: The first returned value `$bestPx` contains the output of the `mop.learning()` function with the number of parameters which gets the best BIC values, taking into account the BIC score to penalize the functions. It evaluates the two next functions, if the BIC score does not improve then the function with the last best BIC is returned.

**Value**

`mop.lerning()` returns a list of n elements:

Function	An "motbf" object of the 'mop' subclass.
Subclass	'mop'.
Domain	The range where the function is defined to be a legal density function.
Iterations	The number of iterations that the optimization problem takes to minimize the errors.
Time	The CPU time employed.

`bestMOP()` returns a list including the polynomial function with the best BIC score, the number of parameters and an array with the BIC values of the evaluated functions.

**See Also**

[univMoTBF](#) A complete function for learning MOPs which includes extra options.

**Examples**

```
## 1. EXAMPLE
data <- rnorm(1000)

## MOP with fix number of degrees
fx <- mop.learning(data, nparam=7, domain=range(data))
fx
hist(data, prob=TRUE, main="")
plot(fx, col=2, xlim=range(data), add=TRUE)

## Best MOP in terms of BIC
fMOP <- bestMOP(data, domain=range(data))
attributes(fMOP)
fMOP$bestPx
hist(data, prob=TRUE, main="")
plot(fMOP$bestPx, col=2, xlim=range(data), add=TRUE)

## 2. EXAMPLE
data <- rbeta(4000, shape1=1/2, shape2=1/2)

## MOP with fix number of degrees
fx <- mop.learning(data, nparam=6, domain=range(data))
fx
hist(data, prob=TRUE, main="")
plot(fx, col=2, xlim=range(data), add=TRUE)

## Best MOP in terms of BIC
fMOP <- bestMOP(data, domain=range(data), maxParam=6)
attributes(fMOP)
fMOP$bestPx
attributes(fMOP$bestPx)
hist(data, prob=TRUE, main="")
plot(fMOP$bestPx, col=2, xlim=range(data), add=TRUE)
```

---

MoTBF-Distribution      *Random generation for MoTBF distributions*

---

**Description**

Random generation for mixtures of truncated basis functions defined in a specific domain. The inverse transform method is used.

**Usage**

```
rMoTBF(size, fx, domain = NULL)
```

```
inversionMethod(size, fx, domain = NULL, data = NULL)
```

**Arguments**

size	A non-negative integer indicating the number of records to generate.
fx	An object of class "motbf".
domain	A "numeric" vector indicating the lower and upper limits to sample from. If not specified, the range is taken from the object fx.
data	A "numeric" vector to be compared with the simulated sample. By default, it is NULL; otherwise, the empirical cumulative distributions of both the data and the simulated sample are plotted and the Kolmogorov Smirnov test is used to test whether or not both samples can be considered to be drawn from the same distribution.

**Value**

rMoTBF() returns a "numeric" vector containing the simulated values. inversionMethod() returns a list with the simulated values and the results of the two-sample Kolmogorov-Smirnov test, as well as the plot of the CDFs of the original and simulated data.

**See Also**

[integralMoTBF](#)

**Examples**

```
## 1. EXAMPLE
## Data
X <- rnorm(1000, mean = 5, sd = 3)

## Learning
f <- univMoTBF(X, POTENTIAL_TYPE="MOP", nparam=10)
plot(f, xlim = f$Domain)

## Random sample
Y <- rMoTBF(size = 500, fx = f)
ks.test(X,Y)

## Plots
hist(Y, prob = TRUE, add = TRUE)

## 2. EXAMPLE
## Data
X <- rweibull(5000, shape=2)

## Learning
f <- univMoTBF(X, POTENTIAL_TYPE="MOP", nparam=10)
plot(f, xlim = f$Domain)

## Random sample
inv <- inversionMethod(size = 500, fx = f, data = X)
attributes(inv)
inv$test
```

```

Y <- inv$sample

## Plots
plot(f, xlim = f$Domain)
hist(Y, prob = TRUE, add = TRUE)

```

---

MoTBFs\_Learning

*Learning hybrid BNs with MoTBFs*


---

## Description

Learn mixtures of truncated basis functions in a full hybrid network.

## Usage

```

MoTBFs_Learning(
  graph,
  data,
  numIntervals,
  POTENTIAL_TYPE,
  maxParam = NULL,
  s = NULL,
  priorData = NULL
)

```

## Arguments

graph	A network of the class "bn", "graphNEL" or "network".
data	An object of class "data.frame"; it can contain continuous and discrete variables.
numIntervals	A positive integer indicating the maximum number of intervals for splitting the domain of the continuous parent variables.
POTENTIAL_TYPE	A "character" string, either <i>MOP</i> or <i>MTE</i> , corresponding to the type of basis function.
maxParam	A positive integer which indicates the maximum number of coefficients in the function. If specified, the output is the function which gets the best BIC with, at most, this number of parameters. By default, it is set to NULL.
s	A "numeric" value which specifies the expert confidence in the prior knowledge. This argument takes values on the interval $[0, N]$ , where $N$ is the sample size, and is used to synchronize the support of the prior knowledge and the sample. By default, it is NULL, and must be modified only if prior information is to be incorporated to the fits.
priorData	An object of class "data.frame", corresponding to the prior information.

## Details

If the variable is discrete then it computes the probabilities and the size of each leaf. Children that have discrete parents have as many functions as configurations of the parents. Children that have continuous parents have as many functions as the number indicated in the argument "numIntervals" for each parent. Children that have mixed parents, combine both methods. The BIC criterion is used to decide the number of splitting points of the parent domains and to choose the number of basis functions used.

## Value

A list of lists. Each list contains two elements

Child	A "charater" string which contains the name of the child variable.
functions	A list of three elements: the name of the parents; a "numeric" vector indicating the interval of the parent; and the fitted function in this interval.

## See Also

[printBN](#) and [ecoli](#)

## Examples

```
## Dataset Ecoli
require(MoTBFs)
data(ecoli)
data <- ecolli[,-c(1)] ## remove variable sequence

## Directed acyclic graph
dag <- LearningHC(data)

## Learning BN
intervals <- 3
potential <- "MOP"
P1 <- MoTBFs_Learning(graph = dag, data = data, numIntervals = intervals, POTENTIAL_TYPE=potential,
maxParam = 5)
printBN(P1)

## Learning BN
intervals <- 4
potential <- "MTE"
P2 <- MoTBFs_Learning(graph = dag, data = data, numIntervals = intervals, POTENTIAL_TYPE=potential,
maxParam = 15)
printBN(P2)
```

---

motbf_type	<i>Type of MoTBF</i>
------------	----------------------

---

**Description**

This function checks whether the density functions of a MoTBF-BN are of type MTE or MOP.

**Usage**

```
motbf_type(bn)
```

**Arguments**

bn                    A list of lists obtained from the function [MoTBFs\\_Learning](#).

**Value**

A character string, specifying the subclass of MoTBF, i.e., either MTE or MOP.

**Examples**

```
## Dataset
data("ecoli", package = "MoTBFs")
data <- ecoli[,-c(1,9)]

## Get directed acyclic graph
dag <- LearningHC(data)

## Learn bayesian network
bn <- MoTBFs_Learning(dag, data = data, numIntervals = 4, POTENTIAL_TYPE = "MTE")

## Get MoTBF sub-class
motbf_type(bn)
```

---

mte.learning	<i>Fitting mixtures of truncated exponentials.</i>
--------------	--

---

**Description**

These functions fit mixtures of truncated exponentials (MTEs). Least square optimization is used to minimize the quadratic error between the empirical cumulative distribution function and the estimated one.

**Usage**

```
mte.learning(X, nparam, domain)
```

```
bestMTE(X, domain, maxParam = NULL)
```



**Arguments**

X	A "numeric" data vector.
nparam	Number of parameters of the resulting density function.
domain	A "numeric" containing the domain if the function to estimate.
maxParam	A "numeric" value indicating the maximum number of coefficients in the function. By default it is NULL; otherwise, the output is the function which gets the best BIC with at most this number of parameters.

**Details**

mte.learning(): The returned value \$Function is the only visible element which contains the algebraic expression. Using [attributes](#) the name of the others elements are shown and also they can be abstract with \$. The [summary](#) of the function also shows all this elements.

bestMTE(): The first returned value \$bestPx contains the output of the mte.learning() function with the number of parameters which gets the best BIC value, taking into account the Bayesian information criterion (BIC) to penalize the functions. It evaluates the two next functions, if the BIC doesn't improve then the function with the last best BIC is returned.

**Value**

mte.lerning() returns a list of n elements:

Function	An "motbf" object of the 'mte' subclass.
Subclass	'mte'.
Domain	The range where the function is defined to be a legal density function.
Iterations	The number of iterations that the optimization problem employed to minimize the errors.
Time	The CPU time consumed.

bestMTE() returns a list including the MTE function with the best BIC score, the number of parameters, the best BIC value and an array contained the BIC values of the evaluated functions.

**See Also**

[univMoTBF](#) A complete function for learning MoTBFs which includes extra options.

**Examples**

```
## 1. EXAMPLE
data <- rchisq(1000, df=3)

## MTE with fix number of parameters
fx <- mte.learning(data, nparam=7, domain=range(data))
hist(data, prob=TRUE, main="")
plot(fx, col=2, xlim=range(data), add=TRUE)

## Best MTE in terms of BIC
fMTE <- bestMTE(data, domain=range(data))
```

```

attributes(fMTE)
fMTE$bestPx
hist(data, prob=TRUE, main="")
plot(fMTE$bestPx, col=2, xlim=range(data), add=TRUE)

## 2. EXAMPLE
data <- rexp(1000, rate=1/3)

## MTE with fix number of parameters
fx <- mte.learning(data, nparam=8, domain=range(data))
## Message: The nearest function with odd number of coefficients
hist(data, prob=TRUE, main="")
plot(fx, col=2, xlim=range(data), add=TRUE)

## Best MTE in terms of BIC
fMTE <- bestMTE(data, domain=range(data), maxParam=10)
attributes(fMTE)
fMTE$bestPx
attributes(fMTE$bestPx)
hist(data, prob=TRUE, main="")
plot(fMTE$bestPx, col=2, xlim=range(data), add=TRUE)

```

---

newRangePriorData      *Redefining the Domain*

---

## Description

Computes the new domain of two datasets.

## Usage

```
newRangePriorData(fPI, priorData, N, domain, s, POTENTIAL_TYPE)
```

## Arguments

fPI	The function fitted to the prior data, of class "motbf".
priorData	A "numeric" array with the values to be included as prior information.
N	A "numeric" value equal to the data size.
domain	A "numeric" array with the domain of the data density.
s	A "numeric" value which is the expert's confidence on the prior information. It is a number between 0 and the data size.
POTENTIAL_TYPE	A "character" string giving the potential of the model, i.e. "MOP" if the basis functions are polynomials, or "MTE" if they are exponentials.

## Value

A "numeric" array which contains the new domain of the prior function.

**Examples**

```
## Data
X <- rnorm(15)

## Prior Data
priorData <- rnorm(5000)

## Learning
type = "MTE"
fPrior <- univMoTBF(priorData, POTENTIAL_TYPE = type)

## New range
confident <- 5 ## confident <- 1,2,...,length(X)
domain <- range(X)
N <- length(X)
newRange <- newRangePriorData(fPrior, priorData = priorData, N = N,
domain = domain, s = confident, POTENTIAL_TYPE = type)
newRange
```

---

nVariables

*Number of Variables in a Joint Function*


---

**Description**

Compute the number of variables which are in a jointmotbf object.

**Usage**

```
nVariables(P)
```

**Arguments**

P                    An "motbf" object or a "jointmotbf" object.

**Value**

A "character" vector with the names of the variables in the function.

**Examples**

```
# 1. EXAMPLE
## Generate a dataset
data <- data.frame(X1 = rnorm(100), X2 = rnorm(100))

## Joint function
dim <- c(3,2)
param <- parametersJointMoTBF(data, dimensions = dim)
P <- jointMoTBF(param)
```

```

P

## Variables
nVariables(P)

#####
## MORE EXAMPLES #####
#####

## Generate a dataset
data <- data.frame(X1 = rnorm(100), X2 = rnorm(100), X3 = rnorm(100))

## Joint function
dim <- c(2,1,3)
param <- parametersJointMoTBF(data, dimensions = dim)
P <- jointMoTBF(param)

## Variables
nVariables(P)

```

---

parentValues

*Value of parent nodes*


---

## Description

This function returns a `data.frame` of dimension '1xn' containing the values of the 'n' parents of a 'node' of interest. Use this function if you have a random sample and an observed sample with information about the parents. The values of the parents are obtained from the evidence set unless they are not observed. In this case, the values are taken from the random sample.

## Usage

```
parentValues(node, bn, obs, rdf)
```

## Arguments

node	A character string that represents the node's name.
bn	A list of lists obtained from the function <a href="#">MoTBFs_Learning</a> . It contains the conditional density functions of the bayesian network.
obs	A <code>data.frame</code> of dimension '1xm' containing an instance of the 'm' variables that belong to the evidence set.
rdf	A <code>data.frame</code> of dimension '1xk' containing an instance of the 'k' variables sampled from the bayesian network.

## Value

A `data.frame` containing the values of the parents of 'node'.

**Examples**

```

## Dataset
data("ecoli", package = "MoTBFS")
data <- ecoli[,-c(1,9)]

## Get directed acyclic graph
dag <- LearningHC(data)

## Learn bayesian network
bn <- MoTBFS_Learning(dag, data = data, numIntervals = 4, POTENTIAL_TYPE = "MTE")

## Specify the evidence set
obs <- data.frame(lip = "1", alm1 = 0.5, stringsAsFactors=FALSE)

## Create a random sample
contData <- data[,which(lapply(data, is.numeric) == TRUE)]
fx <- lapply(contData, univMoTBF, POTENTIAL_TYPE = "MTE")
disData <- data[,which(lapply(data, is.numeric) == FALSE)]
conSample <- lapply(fx, rMoTBF, size = 1)
disSample <- lapply(unique(disData), sample, size = 1)

rdf <- as.data.frame(list(conSample,disSample), stringsAsFactors = FALSE)

## Get the values of the parents of node "alm2"
parentValues("alm2", bn, obs, rdf)

```

---

plot.jointmotbf

*Bidimensional plots for 'jointmotbf' objects*


---

**Description**

PLot the perspective and the contour plots for joint MoTBF functions.

**Usage**

```

## S3 method for class 'jointmotbf'
plot(
  x,
  type = "contour",
  ranges = NULL,
  orientation = c(5, -30),
  data = NULL,
  filled = TRUE,
  ticktype = "simple",
  ...
)

```

**Arguments**

x	An object of class 'jointmotbf'.
type	A "character" string, either <i>contour</i> or <i>perspective</i> . It is set to "contour" by default.
ranges	A "numeric" matrix containing the domain of the variables, by columns, which is used to specify the plotting range.
orientation	A "numeric" vector indicating the perspective of the plot in degrees. By default, it is set to (5, -30).
data	An object of class "data.frame" containing two columns only. This argument is used to draw the points over the main plot. By default, it is set to NULL.
filled	A logical argument; it is only used if type = "contour". is active. By default, it is TRUE, so filled contours are plotted.
ticktype	A "character" string, either <i>simple</i> or <i>detailed</i> . By default, it is set to "simple", which draws just an arrow parallel to the axis to indicate direction of increase. In contrast, "detailed" draws normal ticks. This argument is only used in the "perspective" plot.
...	Further arguments to be passed to <a href="#">plot</a> .

**Value**

A plot of the joint MoTBF.

**See Also**

[jointMoTBF](#)

**Examples**

```
## 1 .EXAMPLE
## Dataset
X <- data.frame(rnorm(500), rnorm(500))

## Joint function
dim <- c(3,3)
param1 <- parametersJointMoTBF(X, dimensions = dim)
P <- jointMoTBF(param1)
P

## Plots
plot(P)
plot(P, type = "perspective", orientation = c(90,0))

#####
## MORE EXAMPLES #####
#####

## Dataset
X <- data.frame(rnorm(200,2), rexp(200, 1))
```

```
## Joint function
dim <- c(4,5)
param2 <- parametersJointMoTBF(X, dimensions = dim)
P <- jointMoTBF(param2)
P

## Plots
plot(P)
plot(P, filled = FALSE, data = X)
plot(P, type = "perspective", orientation = c(10,180))
```

---

plot.motbf

*Plots for 'motbf' objects*


---

### Description

Draws an 'motbf' function.

### Usage

```
## S3 method for class 'motbf'
plot(x, xlim = 0:1, ylim = NULL, type = "l", ...)
```

### Arguments

x	An object of class 'motbf'.
xlim	The range to be encompassed by the x axis; by default 0:1.
ylim	The range of the y axis.
type	As for <a href="#">plot</a> .
...	Further arguments to be passed as for <a href="#">plot</a> .

### Value

A plot of the specified function.

### Examples

```
## 1. EXAMPLE
## Data
X <- rexp(2000)

## Learning
f1 <- univMoTBF(X, POTENTIAL_TYPE = "MOP"); f1
f2 <- univMoTBF(X, POTENTIAL_TYPE = "MTE", maxParam = 10); f2
f3 <- univMoTBF(X, POTENTIAL_TYPE = "MOP", nparam=10); f3
## Plots
```

```

plot(NULL, xlim = range(X), ylim = c(0,0.8), xlab="X", ylab="density")
plot(f1, xlim = range(X), col = 1, add = TRUE)
plot(f2, xlim = range(X), col = 2, add = TRUE)
plot(f3, xlim = range(X), col = 3, add = TRUE)
hist(X, prob = TRUE, add= TRUE)

## 2. EXAMPLE
## Data
X <- c(rnorm(2000, mean = -3),rnorm(2000, mean = 3))

## Learning
f1 <- univMoTBF(X, POTENTIAL_TYPE = "MOP"); f1
f2 <- univMoTBF(X, POTENTIAL_TYPE = "MTE"); f2
## Plots
plot(NULL, xlim = range(X), ylim = c(0,0.20), xlab="X", ylab="density")
plot(f1, xlim = range(X), col = 2, add = TRUE)
plot(f2, xlim = range(X), col = 4, add = TRUE)
hist(X, prob = TRUE, add= TRUE)

```

---

plotConditional

*Plot Conditional Functions*


---

## Description

Plot conditional MoTBF densities.

## Usage

```

plotConditional(
  conditionalFunction,
  data,
  nameChild = NULL,
  points = FALSE,
  color = NULL,
  ...
)

```

## Arguments

conditionalFunction	the output of function <a href="#">conditionalMethod</a> . A list containing the the interval of the parent and the final conditional density (MTE or MOP).
data	An object of class <code>data.frame</code> , corresponding to the dataset used to fit the conditional density.
nameChild	A character string, corresponding to the name of the child variable in the conditional density. By default, it is <code>NULL</code> .
points	A logical value. If <code>TRUE</code> , the sample points are overlaid.



color            If not specified, a default palette is used.  
...              Additional graphical parameters passed to filled.contour().

### Details

If the number of parents is greater than one, then the error message "It is not possible to plot the conditional function." is reported.

### Value

A plot of the conditional density function.

### See Also

[conditionalMethod](#)

### Examples

```
## Data
X <- rnorm(1000)
Y <- rnorm(1000, mean=X)
data <- data.frame(X=X,Y=Y)
cov(data)

## Conditional Learning
parent <- "X"
child <- "Y"
intervals <- 5
potential <- "MTE"
P <- conditionalMethod(data, nameParents=parent, nameChild=child,
numIntervals=intervals, POTENTIAL_TYPE=potential)
plotConditional(conditionalFunction=P, data=data)
plotConditional(conditionalFunction=P, data=data, points=TRUE)
```

---

preprocessedData      *Data cleaning*

---

### Description

Delete rows of a dataset wich contains anomalous values.

### Usage

```
preprocessedData(data, strangeElements)
```

**Arguments**

data                    A dataset of class "matrix" or "data.frame",  
 strangeElements        A "character" string which contains the elements to remove.

---

printBN	<i>BN printing</i>
---------	--------------------

---

**Description**

Prints the content of a hybrid Bayesian network

**Usage**

```
printBN(MoTBF.BN)
```

**Arguments**

MoTBF.BN                The output of the method MoTBFs\_Learning().

**Value**

The results of the fitted functions in the full network.

**See Also**

[MoTBFs\\_Learning](#)

**Examples**

```
## Dataset Ecoli
require(MoTBFs)
data(ecoli)
data <- ecol[, -c(1)] ## remove variable sequence

## Directed acyclic graph
dag <- LearningHC(data)

## Learning BN
intervals <- 3
potential <- "MOP"
P <- MoTBFs_Learning(graph = dag, data = data, numIntervals = intervals, POTENTIAL_TYPE=potential,
maxParam = 15)
printBN(P)
```

---

printConditional      *Summary of conditional MoTBF densities*

---

### Description

Print the description of an MoTBF demnsity for one variable conditional on another variable.

### Usage

```
printConditional(conditionalFunction)
```

### Arguments

conditionalFunction

the output of the function conditionalMethod. A list with the interval of the parent and the final "motbf" density function fitted in each interval.

### Value

The results of the conditional function are shown.

### See Also

[conditionalMethod](#)

### Examples

```
## Data
X <- rexp(500)
Y <- rnorm(500, mean=X)
data <- data.frame(X=X, Y=Y)
cov(data)

## Conditional Learning
parent <- "X"
child <- "Y"
intervals <- 5
potential <- "MOP"
P <- conditionalMethod(data, nameParents=parent, nameChild=child,
numIntervals=intervals, POTENTIAL_TYPE=potential)
printConditional(P)
```

---

printDiscreteBN      *Printing discrete Bayesian networks*

---

**Description**

Prints the univariate and conditional distributions of a discrete BN.

**Usage**

```
printDiscreteBN(BN)
```

**Arguments**

BN                      A discrete learning.

**Value**

The results are shown on the screen.

---

probDiscreteVariable      *Probability distribution of discrete variables*

---

**Description**

Compute the probabilities of a discrete variable from a dataset.

**Usage**

```
probDiscreteVariable(stateNames, Variable)
```

**Arguments**

stateNames              A "character" array indicating the states of the variable.

Variable                A "numeric" array containing the records of the variable.

**Value**

A list of "numeric" arrays:

coeff                    Contains the probabilities.

sizeDataLeaf          Number of records in each leaf of the discrete tree.

**See Also**

[discreteVariablesStates](#)

**Examples**

```
## Discrete Variable
data <- data.frame(X=rep(c("yes", "no", "maybe"), 500))
data <- discreteVariables_as.character(data, "X")
n <- nrow(data)

## Probabilities
s <- discreteVariablesStates(namevariables="X", discreteData=data)
states <- s[[1]]$states
p <- probDiscreteVariable(stateNames=states, Variable=data$X)
p
```

---

r.data.frame

*Data frame initialization for forward sampling*


---

**Description**

The function `r.data.frame()` initializes a data frame with as many columns as nodes in the MoTBF-network. It also assigns each column its data type, i.e., numeric or character. In the case of character columns, the states of the variable are extracted from the "bn" argument and included as levels.

**Usage**

```
r.data.frame(bn, dag)
```

**Arguments**

bn                    A list of lists obtained from the function [MoTBFs\\_Learning](#).  
dag                    An object of class "bn", representing the graph of the bayesian network.

**Value**

An object of class "data.frame", which contains the data type of each column and has no rows.

**Examples**

```
## Create a dataset
# Continuous variables
x <- rnorm(100)
y <- rnorm(100)

# Discrete variable
z <- sample(letters[1:2],size = 100, replace = TRUE)

data <- data.frame(C1 = x, C2 = y, D1 = z, stringsAsFactors = FALSE)

## Get DAG
```

```

dag <- LearningHC(data)

## Learn a BN
bn <- MoTBFs_Learning(dag, data, POTENTIAL_TYPE = "MTE")

## Initialize a data.frame containing 3 columns (x, y and z) with their attributes.
r.data.frame(bn, dag)

```

---

rescaledFunctions      *Rescaling MoTBF functions*

---

### Description

A collation of function to rescale an MoTBF function to the original offset and scale. This is useful when data was standardized previously to learning.

### Usage

```

rescaledMoTBFs(fx, data)

rescaledMOP(fx, data)

ToStringRe_MOP(parameters, data)

rescaledMTE(fx, data)

ToStringRe_MTE(parameters, data, num = 5)

meanMOP(fx)

```

### Arguments

fx	A function of class "motbf" learned from a scaled data.
data	A "numeric" vector containing the original data (non standardized).
parameters	A "numeric" vector with the coefficients to create the rescaled MoTBF.
num	A "numeric" value which contains the denominator of the coefficient in the exponential. By default it is 5.

### Value

An "motbf" function of the original data.

### See Also

[univMoTBF](#)

**Examples**

```
## 1. EXAMPLE
X <- rchisq(1000, df = 8) ## data
modX <- scale(X) ## scale data

## Learning
f <- univMotBF(modX, POTENTIAL_TYPE = "MOP", nparam=10)
plot(f, xlim = range(modX), col=2)
hist(modX, prob = TRUE, add = TRUE)

## Rescale
origF <- rescaledMotBFs(f, X)
plot(origF, xlim = range(X), col=2)
hist(X, prob = TRUE, add = TRUE)
meanMOP(origF)
mean(X)

## 2. EXAMPLE
X <- rweibull(1000, shape = 20, scale= 10) ## data
modX <- as.numeric(scale(X)) ## scale data

## Learning
f <- univMotBF(modX, POTENTIAL_TYPE = "MTE", nparam = 9)
plot(f, xlim = range(modX), col=2, main="")
hist(modX, prob = TRUE, add = TRUE)

## Rescale
origF <- rescaledMotBFs(f, X)
plot(origF, xlim = range(X), col=2)
hist(X, prob = TRUE, add = TRUE)
```

---

rnormMultiv

*Multivariate Normal sampling*


---

**Description**

Generate a multivariate normal data vector taking into account the real data and the relationships with other variables in the dataset.

**Usage**

```
rnormMultiv(n, dataParents, dataChild)
```

**Arguments**

n	A "numeric" value which is the size of the prior data to generate.
dataParents	A data set of class "data.frame" giving the data of the set of conditional parent variables.
dataChild	A "numeric" vector containing the original data of the child variable.

**Value**

A "numeric" vector giving the prior data values.

**See Also**

[generateNormalPriorData](#)

**Examples**

```
## Data
data(ecoli)
data <- ecolli[,-c(1,9)] ## remove sequece.name and class

## DAG
dag <- LearningHC(data)
plot(dag)
getChildParentsFromGraph(dag)

## 1. Random sample
parents <- "mcg"
child <- "alm1"
n <- 1000
rnormMultiv(n, dataParents = data.frame(data[,parents]), dataChild = data[,child])

## 2. Random sample
parents <- "alm1"
child <- "aac"
n <- 256
rnormMultiv(n, dataParents = data.frame(data[,parents]), dataChild = data[,child])
```

---

sample\_MoTBFs

*Sample generation from conditional MoTBFs*

---

**Description**

This function generates a sample from conditional MoTBFs.

**Usage**

```
sample_MoTBFs(bn, dag, obs = NULL, size, force_size = T)
```

**Arguments**

bn	A list of lists obtained from the function <a href="#">MoTBFs_Learning</a> .
dag	An object of class "bn", representing the directed acyclic graph.
obs	A data.frame containing the observed variables. This argument can be omitted if no variable is observed.



`size` A non-negative integer giving the number of instances to be generated.

`force_size` logical indicating if the sample must be of the size indicated. As a default, it is set to TRUE.

**Value**

A `data.frame` containing the generated sample.

**Examples**

```
## Dataset
data("ecoli", package = "MoTBFS")
data <- ecoli[,-c(1,9)]

## Get directed acyclic graph
dag <- LearningHC(data)

## Learn bayesian network
bn <- MoTBFS_Learning(dag, data = data, numIntervals = 4, POTENTIAL_TYPE = "MTE")

## Specify the evidence set
obs <- data.frame(lip = "0.48", alm1 = 0.55, gvh = 1, stringsAsFactors=FALSE)

## Get the conditional sample
sample_MoTBFS(bn, dag, obs, size = 10)
```

---

Subclass-MoTBF

*Subclass "motbf" Functions*


---

**Description**

Collection of functions for detecting the subclass of an "motbf" object. It can be "mop" or "mte".

**Usage**

```
is.mte(fx)

is.mop(fx)

subclass(fx)
```

**Arguments**

`fx` A function of the class "motbf".

**Value**

`is.mte` and `is.mop` return a logical value, TRUE if it is an "motbf" object of the subclass "mte" or "mop", respectively; or FALSE otherwise. `subclass` returns a "character" string, "mte" or "mop".

**See Also**[univMoTBF](#)**Examples**

```
## MOP Function
X <- rnorm(1000)
P <- univMoTBF(X, POTENTIAL_TYPE="MOP")
is.mop(P)
subclass(P)

## MTE Function
X <- rchisq(1000, df=4)
P <- univMoTBF(X, POTENTIAL_TYPE="MTE")
is.mte(P)
subclass(P)
```

---

`subsetData`*Dataset subsetting*

---

**Description**

Collection of functions for subsetting a "data.frame" by rows or columns, and to create training and test partitions.

**Usage**

```
TrainingandTestData(data, percentage_test, discreteVariables = NULL)

newData(data, nameX, nameY)

splitdata(data, nameVariable, min, max)
```

**Arguments**

<code>data</code>	A dataset of class data.frame.
<code>percentage_test</code>	The proportion of data that goes to the test set (between 0 and 1).
<code>discreteVariables</code>	A character vector with the name of the discrete variables.
<code>nameX</code>	A character vector with the name of the child variable in the conditional method.
<code>nameY</code>	A character vector with the name of the parent variables in the conditional method.
<code>nameVariable</code>	A character vector with the name of the variable to be filtered.
<code>min, max</code>	Boundary values to filter out.

**Value**

TrainingandTestData() returns a list of 2 elements containing the train and test datasets. newData() and splitdata() return a subset of variables or observations, respectively.

**Examples**

```
## Dataset
X <- rnorm(1000)
Y <- rchisq(1000, df = 8)
Z <- rep(letters[1:10], times = 1000/10)
data <- data.frame(X = X, Y = Y, Z = Z)
data <- discreteVariables_as.character(dataset = data, discreteVariables = "Z")

## Training and Test Datasets
TT <- TrainingandTestData(data, percentage_test = 0.2)
TT$Training
TT$Test

## Subset Dataset
newData(data, nameX = "X", nameY = "Z")
splitdata(data, nameVariable = "X", min = 2, max = 3)
```

---

summary.jointmotbf      *Summary of a "jointmotbf" object*

---

**Description**

Summarize a "jointmotbf" object by describing its main features.

**Usage**

```
## S3 method for class 'jointmotbf'
summary(object, ...)

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

**Arguments**

object	An object of class "jointmotbf".
...	further arguments passed to or from other methods.
x	An object of class "summary.jointmotbf".

**Value**

The summary of a "jointmotbf" object. It contains a list of elements with the most important information about the object.

**See Also**

[parametersJointMoTBF](#) and [jointMoTBF](#)

**Examples**

```
## 1. EXAMPLE
X <- rnorm(100)
Y <- rexp(100)
data <- data.frame(X, Y)
dim <- c(3,4)
param <- parametersJointMoTBF(data, dimensions=dim)
P <- jointMoTBF(param)
summary(P)
attributes(sP <- summary(P))
attributes(sP)
sP$Function
sP$Domain
sP$Iterations

#####
## MORE EXAMPLES #####
#####

X <- rnorm(100)
Y <- rexp(100)
Z <- rnorm(100, mean=1)
data <- data.frame(X, Y, Z)
dim <- c(3,2,4)
param <- parametersJointMoTBF(data, dimensions=dim)
P <- jointMoTBF(param)
summary(P)
attributes(sP <- summary(P))
sP$Function
sP$Domain
sP$Iterations

#####
#####
```

---

summary.motbf

*Summary of an "motbf" object*


---

**Description**

Summarize an "motbf" object by describing its main features.

## Usage

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

## Arguments

object	An object of class "motbf".
...	further arguments passed to or from other methods.
x	An object of class "summary.motbf".

## Value

The summary of an "motbf" object. It contains a list of elements with the most important information of the object.

## See Also

[univMoTBF](#)

## Examples

```
## Subclass 'MOP'  
X <- rnorm(1000)  
P <- univMoTBF(X, POTENTIAL_TYPE="MOP") ## or POTENTIAL_TYPE="MTE"  
summary(P)  
attributes(sP <- summary(P))  
attributes(sP)  
sP$Function  
sP$Subclass  
sP$Iterations  
  
## Subclass 'MTE'  
X <- rnorm(1000)  
P <- univMoTBF(X, POTENTIAL_TYPE="MTE")  
summary(P)  
attributes(sP <- summary(P))  
attributes(sP)  
sP$Function  
sP$Subclass  
sP$Iterations
```

thyroid

*Data set Thyroid Disease (thyroid0387)***Description**

This data set is one of the several databases about Thyroid available at the UCI repository. The task is to detect if a given patient is normal (1) or suffers from hyperthyroidism (2) or hypothyroidism (3).

**Format**

A data frame with 7200 rows, 21 variables and the class.

**Details**

**Age** Age of the patient (0.01–0.97). Continuous variable.  
**Sex** Sex of the patient, 0 (Male) 1 (Female). Binary variable.  
**On\_thyroxine** 0 (FALSE) 1 (TRUE). Binary variable.  
**Query\_on\_thyroxine** 0 (FALSE) 1 (TRUE). Binary variable.  
**On\_antithyroid\_medication** 0 (FALSE) 1 (TRUE). Binary variable.  
**Sick** 0 (FALSE) 1 (TRUE). Binary variable.  
**Pregnant** 0 (FALSE) 1 (TRUE). Binary variable.  
**Thyroid\_surgery** 0 (FALSE) 1 (TRUE). Binary variable.  
**I131\_treatment** 0 (FALSE) 1 (TRUE). Binary variable.  
**Query\_hypothyroid** 0 (FALSE) 1 (TRUE). Binary variable.  
**Query\_hyperthyroid** 0 (FALSE) 1 (TRUE). Binary variable.  
**Lithium** 0 (FALSE) 1 (TRUE). Binary variable.  
**Goitre** 0 (FALSE) 1 (TRUE). Binary variable.  
**Tumor** 0 (FALSE) 1 (TRUE). Binary variable.  
**Hypopituitary** 0 (FALSE) 1 (TRUE). Binary variable.  
**Psych** 0 (FALSE) 1 (TRUE). Binary variable.  
**TSH** amount of TSH (0.0–0.53). Continuous variable.  
**T3** amount of T3 (0.0005–0.18). Continuous variable.  
**TT4** amount of TT4 (0.002–0.6). Continuous variable.  
**T4U** amount of T4U (0.017–0.233). Continuous variable.  
**FTI** amount of FTI (0.002–0.642). Continuous variable.  
**Class** 1 (normal) 2 (hyperthyroidism) 3 (hypothyroidism). Class variable.

**Source**

<http://archive.ics.uci.edu/ml/datasets/Thyroid+Disease>

univMoTBF

*Fitting MoTBFs***Description**

Function for fitting univariate mixture of truncated basis functions. Least square optimization is used to minimize the quadratic error between the empirical cumulative distribution and the estimated one.

**Usage**

```
univMoTBF(
  data,
  POTENTIAL_TYPE,
  evalRange = NULL,
  nparam = NULL,
  maxParam = NULL
)
```

**Arguments**

data	A "numeric" vector.
POTENTIAL_TYPE	A "character" string specifying the potential type, must be either "MOP" or "MTE".
evalRange	A "numeric" vector that specifies the domain over which the model will be fitted. By default, it is NULL and the function is defined over the complete data range.
nparam	The exact number of basis functions to be used. By default, it is NULL and the best MoTBF is fitted taking into account the Bayesian information criterion (BIC) to score and select the functions. It evaluates the next two functions and, if the BIC value does not improve, the function with the best BIC score so far is returned.
maxParam	A "numeric" value which indicates the maximum number of coefficients in the function. By default, it is NULL; otherwise, the function which gets the best BIC score with at most this number of parameters is returned.

**Value**

univMoTBF() returns an object of class "motbf". This object is a list containing several elements, including its mathematical expression and other hidden elements related to the learning task. The processing time is one of the values returned by this function and it can be extracted by \$Time. Although the learning process is always the same for a particular data sample, the processing can vary inasmuch as it depends on the CPU.

**Examples**

```
## 1. EXAMPLE
## Data
X <- rnorm(5000)

## Learning
f1 <- univMoTBF(X, POTENTIAL_TYPE = "MTE"); f1
f2 <- univMoTBF(X, POTENTIAL_TYPE = "MOP"); f2

## Plots
hist(X, prob = TRUE, main = "")
plot(f1, xlim = range(X), col = 1, add = TRUE)
plot(f2, xlim = range(X), col = 2, add = TRUE)

## Data test
Xtest <- rnorm(1000)
## Filtered data test
Xtest <- Xtest[Xtest>=min(X) & Xtest<=max(X)]

## Log-likelihood
sum(log(as.function(f1)(Xtest)))
sum(log(as.function(f2)(Xtest)))

## 2. EXAMPLE
## Data
X <- rchisq(5000, df = 5)

## Learning
f1 <- univMoTBF(X, POTENTIAL_TYPE = "MTE", nparam = 11); f1
f2 <- univMoTBF(X, POTENTIAL_TYPE = "MOP", maxParam = 10); f2

## Plots
hist(X, prob = TRUE, main = "")
plot(f1, xlim = range(X), col = 3, add = TRUE)
plot(f2, xlim = range(X), col = 4, add = TRUE)

## Data test
Xtest <- rchisq(1000, df = 5)
## Filtered data test
Xtest <- Xtest[Xtest>=min(X) & Xtest<=max(X)]

## Log-likelihood
sum(log(as.function(f1)(Xtest)))
sum(log(as.function(f2)(Xtest)))
```



**Description**

Computes an upper bound of the expected loglikelihood of a dataset given a randomly generated MoTBF density.

**Usage**

```
UpperBoundLogLikelihood(f, data, min, max)
```

**Arguments**

f	A function to evaluate of class "character", "motbf" or others.
data	A "numeric" array which contains the values to evaluate.
min	A "numeric" value giving the lower limit of the domain.
max	A "numeric" value giving the upper limit of the domain.

**Value**

A "numeric" value which is the log-likelihood of the evaluated random function.

**See Also**

[getNonNormalisedRandomMoTBF](#)

**Examples**

```
data <- rnorm(20)
f <- getNonNormalisedRandomMoTBF(degree = 8, POTENTIAL_TYPE = "MOP")
UpperBoundLogLikelihood(f, data, min = -2.5, max = 3.2)

data <- rexp(20)
f <- getNonNormalisedRandomMoTBF(degree = 8, POTENTIAL_TYPE = "MTE")
UpperBoundLogLikelihood(f, data, min = 0, max = 5)
```

# Index

- as.character.jointmotbf  
(Class-JointMoTBF), 9
- as.character.motbf (Class-MoTBF), 10
- as.function, 4, 5
- as.function.jointmotbf, 3
- as.function.motbf, 4
- as.list.jointmotbf (Class-JointMoTBF), 9
- as.list.motbf (Class-MoTBF), 10
- asMOPString, 5, 10
- asMTEString, 6, 10
- attributes, 51, 57
  
- bestMOP (mop.learning), 51
- bestMTE (mte.learning), 56
- BiC.MoTBFBN (goodnessMoTBFBN), 36
- BICMoTBF, 7
- BICMultiFunctions, 8
- BICscoreMoTBF  
(conditionalmotbf.learning), 16
  
- Class-JointMoTBF, 9
- Class-MoTBF, 10
- clean, 11
- coef.jointmotbf, 11
- coef.mop, 12
- coef.motbf, 13, 13, 15
- coef.mte, 15
- coefExpJointCDF, 16
- coeffExp (coef.mte), 15
- coeffMOP, 14
- coeffMOP (coef.mop), 12
- coeffMTE, 14
- coeffMTE (coef.mte), 15
- coeffPol (coef.mop), 12
- conditional  
(conditionalmotbf.learning), 16
- conditionalMethod, 64, 65, 67
- conditionalMethod  
(conditionalmotbf.learning), 16
- conditionalmotbf.learning, 16
  
- dataMining, 19
- derivMOP, 21, 23
- derivMoTBF, 22, 22, 24
- derivMTE, 23, 23
- dimensionFunction, 24
- discreteStatesFromBN, 25
- discreteVariables\_as.character  
(dataMining), 19
- discreteVariablesStates, 68
- discreteVariablesStates (dataMining), 19
- discretizeVariablesEWdis (dataMining),  
19
  
- ecoli, 26, 55
- evalJointFunction, 27, 50
  
- findConditional, 28
- forward\_sampling, 29
  
- generateNormalPriorData, 30, 72
- getBICDiscreteBN  
(goodnessDiscreteVariables), 35
- getChildParentsFromGraph, 31
- getCoefficients, 32, 49
- getlogLikelihoodDiscreteBN  
(goodnessDiscreteVariables), 35
- getNonNormalisedRandomMoTBF, 34, 81
- goodnessDiscreteVariables, 35
- goodnessMoTBFBN, 36
  
- hc, 47
  
- integralJointMoTBF, 37
- integralMOP, 38, 40
- integralMoTBF, 39, 39, 41, 53
- integralMTE, 40, 41
- inversionMethod (MoTBF-Distribution), 52
- is.discrete, 42
- is.jointmotbf (Class-JointMoTBF), 9
- is.mop (Subclass-MoTBF), 73
- is.motbf (Class-MoTBF), 10

- is.mte (Subclass-MoTBF), 73
- is.observed, 43
- is.root, 43
  
- jointCDF, 44
- jointMoTBF, 4, 9, 12, 24, 50, 62, 76
- jointMoTBF (jointmotbf.learning), 45
- jointmotbf (Class-JointMoTBF), 9
- jointmotbf.learning, 45
  
- learn.tree.Intervals
  - (conditionalmotbf.learning), 16
- LearningHC, 46
- learnMoTBFpriorInformation, 33, 48
- logLikelihood.MoTBFBN
  - (goodnessMoTBFBN), 36
  
- marginalJointMoTBF, 49
- meanMOP (rescaledFunctions), 70
- mop.learning, 51
- motbf (Class-MoTBF), 10
- MoTBF-Distribution, 52
- motbf\_type, 56
- MoTBFs\_Learning, 25, 28, 29, 36, 42, 54, 56, 60, 66, 69, 72
- mte.learning, 56
  
- newData (subsetData), 74
- newRangePriorData, 58
- nstates (dataMining), 19
- nVariables, 59
  
- parametersJointMoTBF, 4, 12, 76
- parametersJointMoTBF
  - (jointmotbf.learning), 45
- parentValues, 60
- plot, 62, 63
- plot.jointmotbf, 61
- plot.motbf, 63
- plotConditional, 64
- preprocessedData, 65
- print.jointmotbf (Class-JointMoTBF), 9
- print.motbf (Class-MoTBF), 10
- print.summary.jointmotbf
  - (summary.jointmotbf), 75
- print.summary.motbf (summary.motbf), 76
- printBN, 55, 66
- printConditional, 18, 67
- printDiscreteBN, 68
  
- probDiscreteVariable, 68
  
- quantileIntervals (dataMining), 19
  
- r.data.frame, 69
- rescaledFunctions, 70
- rescaledMOP (rescaledFunctions), 70
- rescaledMoTBFs (rescaledFunctions), 70
- rescaledMTE (rescaledFunctions), 70
- rMoTBF (MoTBF-Distribution), 52
- rnormMultiv, 31, 71
  
- sample\_MoTBFs, 72
- scaleData (dataMining), 19
- select (conditionalmotbf.learning), 16
- splitdata (subsetData), 74
- standardizeDataset (dataMining), 19
- subclass (Subclass-MoTBF), 73
- Subclass-MoTBF, 73
- subsetData, 74
- summary, 51, 57
- summary.jointmotbf, 75
- summary.motbf, 76
  
- thyroid, 78
- ToStringRe\_MOP (rescaledFunctions), 70
- ToStringRe\_MTE (rescaledFunctions), 70
- TrainingandTestData (subsetData), 74
  
- univMoTBF, 7, 8, 13–15, 22–24, 29, 39–41, 52, 57, 70, 74, 77, 79
- UpperBoundLogLikelihood, 80
  
- whichDiscrete (dataMining), 19