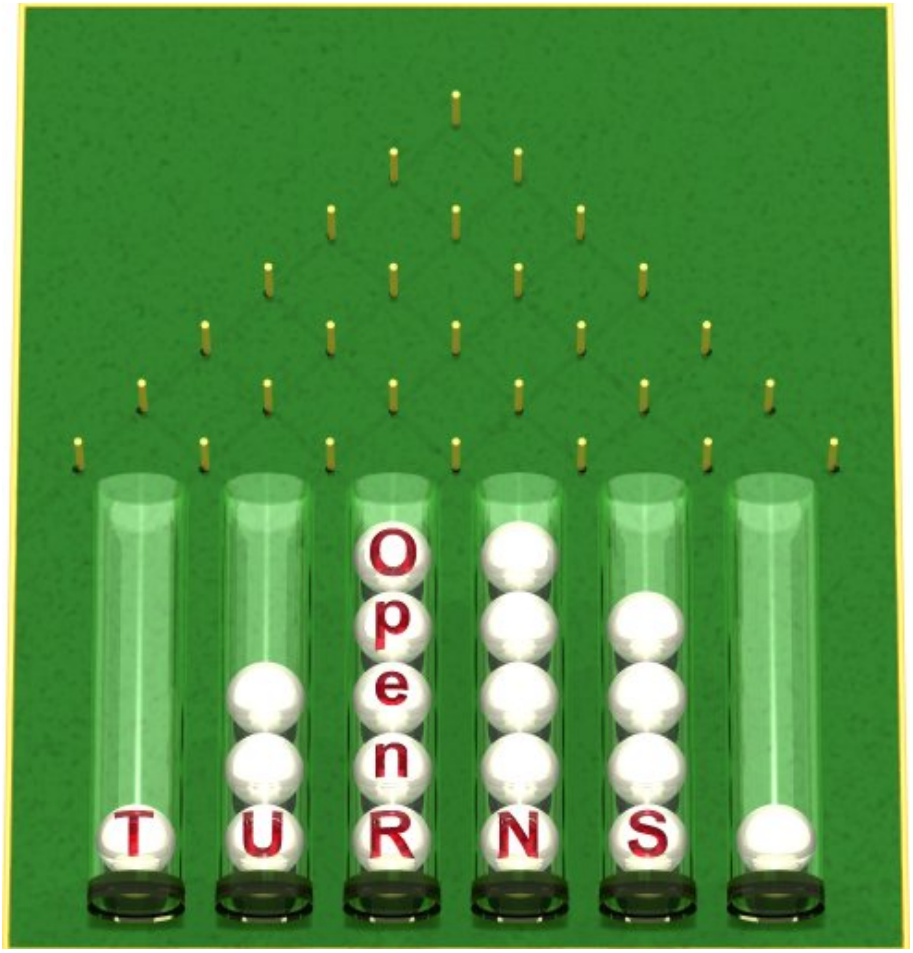
User Manual for the Textual User Interface

Open TURNS version 0.12.1

November 8, 2008



Contents

©2007 EDF - EADS - PhiMeca

©2007 EDF - EADS - PhiMeca

©2007 EDF - EADS - PhiMeca

1 Base Objects

In this section a description of general objects is given. These objects are used in the different following sections.

1.1 CorrelationMatrix

Usage :

$CorrelationMatrix(dim)$

$CorrelationMatrix(dim, values)$

Arguments :

dim : an integer, the dimension of the CorrelationMatrix (square matrix with *dim* rows and *dim* colons)

values : a NumericalScalarCollection of dimension dim^2 which contains values to put in the CorrelationMatrix, filled by rows. When these values are not specified, the CorrelationMatrix is initialised to the identity matrix.

Value : a CorrelationMatrix

while using the first parameters set, the correlation matrix is the identity matrix

while using second parameters set, the correlation matrix contains the specified values, filled by row

Some methods :

str

Usage : *str*()
Arguments : no argument
Value : a string giving the description of the (class, name, dimension, values)

transpose

Usage : *transpose*()
Arguments : no argument
Value : a CorrelationMatrix, the transposed CorrelationMatrix

computeDeterminant

Usage : *computeDeterminant*()
Arguments : no argument
Value : a real value giving the determinant of the CorrelationMatrix

computeEigenValues

Usage : *computeEigenValues*()
Arguments : no argument
Value : a NumericalPoint giving the eigen values of the CorrelationMatrix

Links : see docref_B121_ChoixLoi

©2007 EDF - EADS - PhiMeca

1.2 Description

Usage :

$Description(dim)$

$Description(dim, name)$

Arguments :

dim : an integer, the dimension of the Description

$name$: a string to name the Description

Value : a Description

Some methods :

[]

Usage : $Description[i]$

Arguments : i : an integer, constraint : $0 \leq i \leq dim - 1$

Value : a string, the description of the $(i + 1)$-th element of the Description

add

Usage : $add(str)$

Arguments : str : a string

Value : an element is added to the Description which name is str

getSize

Usage : $getSize()$

Arguments : none

Value : an integer, the size of the Description (dim)

getName

Usage : $getName()$

Arguments : none

Value : a string, the name of the Description

setName

Usage : $setName(name)$

Arguments : $name$: a string to name the Description

Value : the Description is then named $name$

©2007 EDF - EADS - PhiMeca

1.3 DistributionCollection

Usage : *DistributionCollection*(*dim*)

Arguments : *dim* : an integer, the dimension of the collection of distributions

Value : a DistributionCollection, to be filled after

Some methods :

[]

Usage : *DistributionCollection*[*i*]
Arguments : i : an integer, constraint : $0 \leq i \leq dim - 1$
Value : a Distribution, the $(i+1)$-th Distribution of the DistributionCollection

add

Usage : *add*(*distribution*)
Arguments : *distribution*, a Distribution
Value : a DistributionCollection of size $dim + 1$, which $(dim + 1)$-th element is *distribution*

at

Usage : *at*(*i*)
Arguments : i, an integer
Value : a Distribution, the distribution of the $(i+1)$-th component of a random vector (type CollectionOfDistribution.at(i).str() to have a brief description)

Links : see docref_B_JoinedCDF_en

©2007 EDF - EADS - PhiMeca

1.4 HistogramPair

Usage : $HistogramPair(h, l)$

Arguments :

h : a real value, the height of each element of the Histogram

l : a real value, the width of each element of the Histogram

Value : a HistogramPair

Details :

This object is used to build a HistogramPairCollection (hence, also used to create an Histogram)

©2007 EDF - EADS - PhiMeca

1.5 HistogramPairCollection

Usage : $HistogramPairCollection(dim)$

Arguments : dim : an integer, the number of elements of the HistogramPairCollection

Value : a HistogramPairCollection, to be filled after

Some methods :

[]

Usage : $HistogramPairCollection[i]$

Arguments : i : an integer, must be $< dim$

Value : a HistogramPair, the $(i+1)$-th of the HistogramPairCollection

add

Usage : $add(HistP)$

Arguments : $HistP$: a HistogramPair

Value : an HistogramPairCollection of $Size = dim + 1$ with instance of the $dim + 1$-th element of the HistogramPairCollection

getSize

Usage : $getSize()$

Arguments : no argument

Value : an integer, the size of HistogramPairCollection (returns dim)

str

Usage : $str()$

Arguments : no argument

Value : a string with elements of HistogramPairCollection

©2007 EDF - EADS - PhiMeca

1.6 Matrix

Usage :

$Matrix(n_r, n_c)$
$Matrix(n_r, n_c, values)$

Arguments :

n_r : an integer, the number of rows of the Matrix

n_c : an integer, the number of columns of the Matrix

$values$: a NumericalScalarCollection with $n_r \times n_c$ elements

Value : a Matrix

while using the first parameters set, the matrix is filled with 0.

while using second parameters set, the Matrix contains values of the NumericalScalarCollection. The matrix is filled by row

Some methods :

[,]

Usage : $Matrix[i, j]$

Arguments :

i : an integer, constraint : $0 \leq i \leq n_r - 1$

j : an integer, constraint : $0 \leq j \leq n_c - 1$

Value : a real value, the (i, j) element of the Matrix

getNbColumns

Usage : $getNbColumns()$

Arguments : none

Value : an integer : the number of column n_c

getNbRows

Usage : $getNbRows()$

Arguments : none

Value : an integer : the number of row n_r

transpose

Usage : $transpose()$

Arguments : none

Value : the transposed Matrix

getName

Usage : $getname()$

Arguments : none

©2007 EDF - EADS - PhiMeca

Value : a string, the name of the Matrix

setName

Usage : *setname*(*name*)
Arguments : name : a string
Value : the Matrix is named *name*

solveLinearSystem

Usage : *solveLinearSystem*(y)
Arguments : y a NumericalPoint of dimension n_r (the number of rows of the Matrix)
Value : NumericalPoint, x, such that

```
Matrix * x = y
```

©2007 EDF - EADS - PhiMeca

1.7 NumericalMathFunction

Usage :

$NumericalMathFunction(file)$

$NumericalMathFunction(input, output, formula)$

$NumericalMathFunction(f, g)$

Arguments :

input : a Description which describes the input of the NumericalMathFuction

output : a Description which describes the output of the NumericalMathFuction

$formula$: a Description, the analytical formula of the NumericalMathFuction

$file$: a string to name the XML file (without the extension ".xml") which contains the implementation of the considered function

f, g : two NumericalMathFunction in norder to create the composition function $f \circ g$

Value : NumericalMathFunction

Some methods :

()

Usage :

$NumericalMathFunction(x)$

$NumericalMathFunction(sample)$

Arguments :

x : a NumericalPoint

sample : a NumericalSample

Value :

while using the first usage, a NumericalPoint, the NumericalMathFunction value at point x

while using the second usage, a NumericalSample, the NumericalMathFunction value on the sample *sample*

getDescription

Usage : *getDescription*()

Arguments : none

Value : a Description which describes the inputs and the ouputs of the NumericalMathFunction (use `print NumericalMathFuction.getDescription()` command to visualize it)

getEvaluationCallsNumber

Usage : *getEvaluationCallsNumber*()

Arguments : none

Value : an integer that counts the number of times the NumericalMathFunction has been called since the beginning of the python session

getGradientCallsNumber

©2007 EDF - EADS - PhiMeca

Usage : *getGradientCallsNumber*()

Arguments : none

Value : an integer that counts the number of times the gradient of the NumericalMathFunction has been called since the beginning of the python session. Note that if the gradient is implemented by a finite difference method, the gradient calls numbers is equal to 0 and the different calls are comptabilised in the evaluation calls number

getHessianCallsNumber

Usage : *getHessianCallsNumber*()

Arguments : none

Value : an integer that counts the number of times the gradient of the NumericalMathFunction has been called since the beginning of the python session. Note that if the hessian is implemented by a finite difference method, the hessian calls numbers is equal to 0 and the different calls are comptabilised in the evaluation calls number

getInputDescription

Usage : *getInputDescription*()

Arguments : none

Value : a Description which describes the inputs of the NumericalMathFunction

getInputNumericalPointDimension or *getInputDimension*

Usage : *getInputNumericalPointDimension*(), *getInputDimension*()

Arguments : none

Value : an integer, the dimension of the input space

getMarginal

Usage : *getMarginal*(i)

Arguments : i : an integer corresponding to the marginal (Care : the numerotation begins at 0)

Value : a NumericalMathFunction, noted f_i if $f : \mathcal{R}^n \longrightarrow \mathcal{R}^p$, with $f = (f_0, \cdots, f_p)$.

getOutputDescription

Usage : *getOutputDescription*()

Arguments : none

Value : a Description which describes the outputs of the NumericalMathFunction object

*getOutputNumericalPointDimension*or *getOutputDimension*

Usage : *getOutputNumericalPointDimension*(), *getOutputDimension*()

Arguments : none

Value : an integer, the dimension of the output space

getParameters

Usage : *getParameters*()

©2007 EDF - EADS - PhiMeca

Arguments : none
Value : a NumericalPoint, the NumericalPoint corresponding to parameters of the NumericalMathFunction

GetValidOperators

Usage : *GetValidOperators*()
Arguments : none
Value : a Description, containing the list of the operators we can use within Open TURNS

GetValidFunctions

Usage : *GetValidFunctions*()
Arguments : none
Value : a Description, containing the list of the functions we can use within Open TURNS

GetValidConstants

Usage : *GetValidConstants*()
Arguments : none
Value : a Description, containing the list of the constants we can use within Open TURNS

gradient

Usage : *gradient*(x)
Arguments : x : a NumericalPoint (which has the same dimension as the inputs)
Value : a Matrix, the gradient (with respect to the inputs) of the NumericalMathFunction

hessian

Usage : *hessian*(x)
Arguments : x : a NumericalPoint (which has the same dimension as the inputs)
Value : a SymmetricTensor, the hessian (with respect to the inputs) of the NumericalMathFunction

setName

Usage : *setName*(*name*)
Arguments : *name* : a string (between quotations marks)
Value : it gives a name to the NumericalMathFunction

Here is the list of constants proposed by Open TURNS :

- _e : Euler's constant (2.71828...),
- _pi : Pi constant (3.14159...)

Here is the list of functions proposed by Open TURNS :

- $sin(arg)$: sine function,
- $cos(arg)$: cosine function,

©2007 EDF - EADS - PhiMeca

- $tan(arg)$: tangent function,
- $asin(arg)$: inverse sine function,
- $acos(arg)$: inverse cosine function,
- $atan(arg)$: inverse tangent function,
- $sinh(arg)$: hyperbolic sine function,
- $cosh(arg)$: hyperbolic cosine function,
- $tanh(arg)$: hyperbolic tangens function,
- $asinh(arg)$: inverse hyperbolic sine function,
- $acosh(arg)$: inverse hyperbolic cosine function,
- $atanh(arg)$: inverse hyperbolic tangent function,
- $log2(arg)$: logarithm in base 2,log10(arg) : logarithm in base 10,log(arg) : logarithm in base e (2.71828...),ln(arg) : alias for log function,
- $lngamma(arg)$: log of the gamma function,
- $gamma(arg)$: gamma function,
- $exp(arg)$: exponential function,
- $erf(arg)$: error function,
- $erfc(arg)$: complementary error function,
- $sqrt(arg)$: square root function,
- $cbrt(arg)$: cubic root function,
- $besselJ0(arg)$: 1rst kind Bessel function with parameter 0,
- $besselJ1(arg)$: 1rst kind Bessel function with parameter 1,
- $besselY0(arg)$: 2nd kind Bessel function with parameter 0,
- $besselY1(arg)$: 2nd kind Bessel function with parameter 1,
- $sign(arg)$: sign function -1 if x<0; 1 if x>0,
- $rint(arg)$: round to nearest integer function,
- $abs(arg)$: absolute value function,
- $if(arg1, arg2, arg3)$: if arg1 then arg2 else arg3,
- $min(arg1, ..., argn)$: min of all arguments,
- $max(arg1, ..., argn)$: max of all arguments,
- $sum(arg1, ..., argn)$: sum of all arguments,

©2007 EDF - EADS - PhiMeca

- $avg(arg1, ..., argn)$: mean value of all arguments .

Here is the list of operators proposed by Open TURNS :

- $=$: assignement, can only be applied to variable names (priority -1),
- *and* : logical and (priority 1),
- *or* : logical or (priority 1),
- *xor* : logical xor (priority 1),
- $<=$: less or equal (priority 2),
- $>=$: greater or equal (priority 2),
- $!=$: not equal (priority 2),
- $==$: equal (priority 2),
- $>$: greater than (priority 2),
- $<$: less than (priority 2),
- $+$: addition (priority 3),
- $-$: subtraction (priority 3),
- $*$: multiplication (priority 4),
- $/$: division (priority 4),
- : logical negation (priority 4),
- *not* : alias for (priority 4),
- $-$: sign change (priority 4),
- : raise x to the power of y (priority 5).

1.8 NumericalPoint

Usage : $NumericalPoint(dim, value)$

Arguments :

dim : an integer, the dimension of the NumericalPoint
$value$: a real value, the value of each component of the NumericalPoint

Some methods :

[]

Usage : $NumericalPoint[i]$
Arguments :
i : an integer, constraint : $0 \leq i \leq dim - 1$

©2007 EDF - EADS - PhiMeca

Value : a real value, the value of the $(i+1)$-th element of the NumericalPoint

getDimension

Usage : *getDimension*()
Arguments : none
Value : an integer, the value of the dimension of the NumericalPoint (it returns dim)

norm

Usage : *norm*()
Arguments : none
Value : a real value, the euclidian norm of the NumericalPoint

*norm*2

Usage : *norm*2()
Arguments : none
Value : a real value, the square of the euclidian norm of the NumericalPoint

str

Usage : *str*()
Arguments : none
Value : a string describing the NumericalPoint

dot

Usage : $dot(x, y)$
Arguments : x, y : NumericalPoint
Value : a real value, the dot product (also known as the scalar product) of x and y

getName

Usage : *getName*()
Arguments : none
Value : a string giving the name of the NumericalPoint

setName

Usage : *setName*(*name*)
Arguments : *name* : a string
Value : no value, it gives a name for the considered NumericalPoint

©2007 EDF - EADS - PhiMeca

1.9 NumericalPointCollection

Usage : $NumericalPointCollection(dim)$

Arguments : dim : an integer, the number of elements of the NumericalPointCollection

Value : a NumericalPointCollection, filled by default with 0.0

Some methods :

[]

Usage : $NumericalPointCollection[i]$
Arguments : i : an integer, constraint : $0 \leq i \leq dim - 1$
Value : a NumericalPoint, the $(i+1)$-th element of the NumericalPointCollection

add

Usage : $add(NumericalPoint2)$
$NumericalPoint$: a NumericalPoint
Value : The NumericalPointCollection of size $dim + 1$. The $dim + 1$ element of this object is then equal to $NumericalPoint2$

getSize

Usage : $getSize()$
Arguments : none
Value : an integer : the size of the NumericalPointCollection

©2007 EDF - EADS - PhiMeca

1.10 NumericalScalarCollection

Usage : $NumericalScalarCollection(dim)$

Arguments : dim : an integer, the number of elements of the NumericalScalarCollection

Value : a NumericalScalarCollection, filled by default with 0

Some methods :

[]

Usage : $NumericalScalarCollection[i]$

Arguments :

i : an integer, constraint : $0 \leq i \leq dim - 1$

Value : a real value, the $(i+1)$-th element of the NumericalScalarCollection

add

Usage : $add(val)$

val : a real value

Value : The NumericalScalarCollection of size $dim + 1$. The $dim + 1$ element of this object is then equal to val

getSize

Usage : $getSize()$

Arguments : none

Value : an integer : the size of the NumericalScalarCollection

©2007 EDF - EADS - PhiMeca

1.11 SquareMatrix

Usage :

SquareMatrix(*dim*)
SquareMatrix(*dim*, *values*)

Arguments :

dim : an integer, the dimension of the SquareMatrix (square matrix with *dim* rows and *dim* colons)
values : a NumericalScalarCollection of dimension dim^2

Value : SquareMatrix

while using the first parameters set, the SquareMatrix is filled with 0.
while using the second parameters set, the SquareMatrix contains values of the NumericalScalarCollection. SquareMatrix is filled by rows.

Some methods :

computeDeterminant
Usage : *computeDeterminant*()
Arguments : none
Value : a real value giving the determinant of the SquareMatrix

computeEigenValues
Usage : *computeEigenValues*()
Arguments : none
Value : a NumericalPoint giving the eigen values of the SquareMatrix

getDimension
Usage : *getDimension*()
Arguments : none
Value : an integer, the dimension of the SquareMatrix (it returns *dim*)

solveLinearSystem
Usage : *solveLinearSystem*(*y*)
Arguments : y : a NumericalPoint of dimension n_r (the number of row of the SquareMatrix)
Value : NumericalPoint, this NumericalPoint, x, is such that

```
SquareMatrix * x = y
```

transpose
Usage : *transpose*()
Arguments : none
Value : the transposed SquareMatrix

©2007 EDF - EADS - PhiMeca

1.12 Tensor

Usage :

$Tensor(n_r, n_c, n_s)$
$Matrix(n_r, n_c, n_s, values)$

Arguments :

n_r : an integer, the number of row of the Tensor

n_c : an integer, the number of row of the Tensor

n_s : an integer, the number of sheet of the Tensor

$values$: NumericalScalarCollection with $n_r \times n_c \times n_s$ elements

Value : Tensor

while using the first parameters set, the matrix is filled with 0.

while using the second parameters set, the Matrix contains values of the NumericalScalarCollection. The tensor is filled by row.

Some methods :

[, ,]

Usage : $Tensor[i, j, k]$
Arguments :
i : an integer, constraint : $0 \leq i \leq n_r - 1$
j : an integer, constraint : $0 \leq j \leq n_c - 1$
k: an integer, constraint : $0 \leq j \leq n_s - 1$
Value : a real value, the (i, j, k) element of the Tensor

getNbColumns

Usage : $getNbColumns()$
Arguments : none
Value : an integer : the number of column n_c

getNbRows

Usage : $getNbRows()$
Arguments : none
Value : an integer : the number of row n_r

getNbSheets

Usage : $getNbSheets()$
Arguments : none
Value : an integer : the number of sheet n_s

getName

©2007 EDF - EADS - PhiMeca

Usage : $getName()$
Arguments : none
Value : a string, the name of the Tensor

setName

Usage : $setName(name)$
Arguments : name : a string
Value : the Tensor is named *name*

©2007 EDF - EADS - PhiMeca

1.13 UserDefinedPair

Usage : $UserDefinedPair(x, p)$

Arguments :

x : a NumericalPoint,

p : a real value, constraint $0 \leq p \leq 1$ (the probabiliy associated to the point x)

Value : a UserDefinedPair

Some methods :

$getX()$

Usage : $getX()$

Arguments : no argument

Value : a NumericalPoint, the point of the UserDefinedPair

$getP()$

Usage : $getP()$

Arguments : no argument

Value : a NumericalScalar, the scalar of the UserDefinedPair

Each get method is associated to a set method.

©2007 EDF - EADS - PhiMeca

1.14 UserDefinedPairCollection

Usage : $UserDefinedPairCollection(dim)$

Arguments : dim : an integer, the number of elements of the UserDefinedPairCollection

Value : an UserDefinedPairCollection, to be filled after

Some methods :

[]

Usage : $UserDefinedPairCollection[i]$
Arguments : i : an integer, the ith element of UserDefinedPairCollection
Value : a UserDefinedPair, the $(i+1)$-th element of UserDefinedPairCollection

add

Usage : $add(UseDefP)$
Arguments : $UseDefP$: an UserDefinedPair
Value : a UserDefinedPairCollection of size $dim+1$ with instance of the $(dim+1)$ element of the UserDefinedPairCollection

getSize

Usage : $getSize()$
Arguments : no argument
Value : an integer, the size of UserDefinedPairCollection (returns dim)

str

Usage : $str()$
Arguments : no argument
Value : a string with elements of UserDefinedPairCollection

©2007 EDF - EADS - PhiMeca

2 Distributions

In this section, a description about the use of Distribution object is given.

Be aware of the fact that for some uses in the TUI, it is necessary to explicitely cast a given distribution into the general Distribution class.

2.1 Distribution

Usage : $Distribution(dist, name)$

Arguments :

dist : a DistributionImplementation which is Beta, Exponential, Gamma, Geometric, Gumbel, Histogram, Logistic, LogNormal, MultiNomial, Normal, Non Central Student, Poisson, Student, Triangular, TruncatedNormal, Weibull, UserDefined,

name : a string to name the distribution

Value : a Distribution

Some methods :

computeCDF

Usage :

$computeCDF(value)$

$computeCDF(x)$

$computeCDF(sample)$

Arguments :

x : a NumericalScalar

x : a NumericalPoint

sample : a NumericalSample

Value :

while using the first usage, a NumericalScalar, the CDF (Cumulative Distribution Function) of dimension 1 value of the considered distribution at *value*

while using the second usage, a NumericalPoint, the CDF (Cumulative Distribution Function) value of the considered distribution at the vector x

while using the third usage, a NumericalSample, the CDF (Cumulative Distribution Function) values of the considered distribution at *sample*

computeCDFGradient

Usage : $computeCDFGradient(x)$

Arguments : x : a NumericalPoint

Value : a NumericalPoint object, the gradient of the distribution CDF, with respect to the parameters of the distribution, evaluated at point x

computeDDF

Usage :

©2007 EDF - EADS - PhiMeca

computeDDF(x)

computeDDF(*sample*)

Arguments :

x : a NumericalPoint

sample : a NumericalSample

Value :

while using the first usage, a NumericalPoint value, the gradient of the PDF (Probability Distribution Function) of the considered distribution at x (DDF = Derivative Density Function)

while using the second usage, a NumericalSample, the gradient of the PDF (Probability Distribution Function) of the considered distribution at x (DDF = Derivative Density Function)

computePDF

Usage :

computePDF(*value*)

computePDF(x)

computePDF(*sample*)

Arguments :

x : a NumericalPoint

sample : a NumericalSample

Value :

while using the first usage, a NumericalScalar, the PDF (Cumulative Distribution Function) of dimension 1 value of the considered distribution at *value*

while using the second usage, a NumericalPoint, the PDF (Cumulative Distribution Function) value of the considered distribution at the vector x

while using the third usage, a NumericalSample, the PDF (Cumulative Distribution Function) values of the considered distribution at *sample*

computePDFGradient

Usage : *computePDFGradient*(x)

Arguments : x : a NumericalPoint

Value : a NumericalPoint object, the gradient of the distribution PDF, with respect to the parameters of the distribution, evaluated at point x

computeQuantile

Usage : *computeQuantile*(x)

Arguments : x : a real scalar $0 \leq x \leq 1$

Value : a NumericalPoint, the value of the $x-$ quantile

drawCDF

Usage :

drawCDF()

drawCDF(min, max)

drawCDF($min, max, pointNumber$)

©2007 EDF - EADS - PhiMeca

$drawCDF(vectMin, vectMax)$

$drawCDF(vectMin, vectMax, vectPointNumber)$

Arguments :

min and max : real values with $min < max$, the range for the CDF curve of a distribution of dimension 1

$pointNumber$: an integer, the number of points to draw the CDF iso-curves of a distribution of dimension 1

$vectMin$ and $vectMax$: two NumericalPoint of dimension 2, respectively the left-bottom and ritgh-up corners of the square for the CDF iso-curves of a distribution of dimension 2

$vectPointNumber$: a NumericalPoint of dimension 2, the the number of points to draw the iso-curves of a distribution of dimension 2 on each direction

Value : a Graph, containing the elements of the curve or iso-curves of the CDF, depending on the dimension of the distribution (1 or 2)

$drawPDF$

Usage :

$drawPDF()$

$drawPDF(min, max)$

$drawPDF(min, max, pointNumber)$

$drawPDF(vectMin, vectMax)$

$drawPDF(vectMin, vectMax, vectPointNumber)$

Arguments :

min and max : real values with $min < max$, the range for the PDF curve of a distribution of dimension 1

$pointNumber$: an integer, the number of points to draw the PDF iso-curves of a distribution of dimension 1

$vectMin$ and $vectMax$: two NumericalPoint of dimension 2, respectively the left-bottom and ritgh-up corners of the square for the PDF iso-curves of a distribution of dimension 2

$vectPointNumber$: a NumericalPoint of dimension 2, the number of points to draw the iso-curves of a distribution of dimension 2 on each direction

Value : a Graph, containing the elements of the curve or iso-curves of the PDF, depending on the dimension of the distribution (1 or 2)

$drawMarginal1DCDF$

Usage :

$drawMarginal1DCDF(i, min, max, pointNumber)$

Arguments :

i : an integer, the marginal we want to draw (Care : numerotation begins at 0)

min and max : real values with $min < max$, the range for the CDF curve of a distribution of dimension >1

$pointNumber$: an integer, the number of points to draw the CDF iso-curves of a distribution of dimension >1

Value : a Graph, containing the elements of the curve of the CDF of the marginal i of the distribution of dimension >1

©2007 EDF - EADS - PhiMeca

drawMarginal1DPDF

Usage :

$drawMarginal1DPDF(i, min, max, pointNumber)$

Arguments :

i : an integer, the marginal we want to draw (Care : numerotation begins at 0)

min and max : real values with $min < max$, the range for the PDF curve of a distribution of dimension >1

$pointNumber$: an integer, the number of points to draw the PDF iso-curves of a distribution of dimension >1

Value : a Graph, containing the elements of the curve of the PDF of the marginal i of the distribution of dimension >1

drawMarginal2DCDF

Usage :

$drawMarginal2DCDF(i, j, vectMin, vectMax, vectPointNumber)$

Arguments :

i and j : two integer, the marginal we want to draw (Care : numerotation begins at 0)

$vectMin$ and $vectMax$: two NumericalPoint of dimension n>2, respectively the left-bottom and ritgh-up corners of the square for the PDF iso-curves of a distribution of dimension n

$vectPointNumber$: a NumericalPoint of dimension n>2, the number of points to draw the iso-curves of a distribution of dimension n on each direction

Value : a Graph, containing the elements of the iso-curve of the CDF of the marginals (i,j) of distribution of dimension n>2

drawMarginal2DPDF

Usage :

$drawMarginal2DPDF(i, j, vectMin, vectMax, vectPointNumber)$

Arguments :

i and j : two integer, the marginal we want to draw (Care : numerotation begins at 0)

$vectMin$ and $vectMax$: two NumericalPoint of dimension n>2, respectively the left-bottom and ritgh-up corners of the square for the PDF iso-curves of a distribution of dimension n

$vectPointNumber$: a NumericalPoint of dimension n>2, the number of points to draw the iso-curves of a distribution of dimension n on each direction

Value : a Graph, containing the elements of the iso-curve of the PDF of the marginals (i,j) of distribution of dimension n>2

getCopula

Usage : $getCopula()$

Arguments : no argument

Value : a Copula, the copula of the considered distribution which must be of type ComposedDistribution

getCovariance

©2007 EDF - EADS - PhiMeca

Usage : *getCovariance*()

Arguments : no argument

Value : a CovarianceMatrix of the considered distribution (if the distribution is unidimensional, it is the variance)

getMarginal

Usage :

getMarginal(i)

getMarginal($indices$)

Arguments :

i : an integer (i is lower or equal to the dimension of the considered distribution), with $0 \leq i$

$indices$: a Indices, which regroup all the indices considered

Value : a Distribution, the distribution of an extracted vector of the initial distribution

getKurtosis

Usage : *getKurtosis*()

Arguments : no argument

Value : a NumericalPoint, the value the kurtosis of each 1D marginal of the distribution

getMean

Usage : *getMean*()

Arguments : no argument

Value : a NumericalPoint, the value of the considered distribution mean

getNumericalSample

Usage : *getNumericalSample*(n)

Arguments : n : integer, the size of the sample

Value : a NumericalSample representing n realizations of the random variable with the considered distribution

getParametersCollection

Usage : *getParametersCollection*()

Arguments : one

Value : a NumericalPointCollection, the list of the parameters of the distribution

getRealization

Usage : *getRealization*()

Arguments : no argument

Value : a NumericalPoint, one realization of random variable with the considered distribution

getRoughness

Usage : *getRoughness*()

©2007 EDF - EADS - PhiMeca

Arguments : no argument

Value : a NumericalScalar, the value $roughness(\underline{X}) = ||p||_{\mathcal{L}^2} = \sqrt{\int_{\underline{x}} p^2(\underline{x}) d\underline{x}}$

getSkewness

Usage : *getSkewness*()

Arguments : no argument

Value : a NumericalPoint, the value the standard deviation of each 1D marginal of the distribution

getStandardDeviation

Usage : *getStandardDeviation*()

Arguments : no argument

Value : a NumericalPoint, the value the standard deviation of each 1D marginal of the distribution

getWeight

Usage : *getWeight*()

Arguments : no argument

Value : a NumericalScalar between 0 and 1, the weight of the considered distribution if used in a Mixture

hasEllipticalCopula

Usage : *hasEllipticalCopula*()

Arguments : no argument

Value : a boolean, it says if the considered distribution is elliptical

hasIndependentCopula

Usage : *hasIndependentCopula*()

Arguments : no argument

Value : a boolean which indicates wether the considered distribution is independent

isElliptical

Usage : *isElliptical*()

Arguments : no argument

Value : a boolean which indicates wether the considered distribution has an elliptical distribution

str

Usage : *str*()

Arguments : no argument

Value : a string describing the object

©2007 EDF - EADS - PhiMeca

2.2 Usual Distributions

2.2.1 Beta

This class inherits from the Distribution class.

Usage :

Main parameters set : $Beta(r, t, a, b)$

Second parameters set : $Beta(\mu, \sigma, a, b, Beta.MUSIGMA)$

Default construction : $Beta()$

Arguments :

r : real value, first shape parameter, constraint : $r > 0$

t : real value, second shape parameter, constraint : $t > r$

a : real value, lower bound

b : real value, upper bound, constraint : $b > a$

μ : real value, mean value

σ : real value, standard deviation, constraint : $\sigma > 0$

Value : a Beta. In the default construction, we use the $Beta(r, t, a, b) = Beta(2, 4, -1, 1)$ definition.

Some methods :

getA

Usage : $getA()$
Arguments : none
Value : a real value, the a parameter of the Beta distribution

getB

Usage : $getB()$
Arguments : none
Value : a real value, the b parameter of the Beta distribution

getMu

Usage : $getMu()$
Arguments : none
Value : a real value, the μ parameter of the distribution

getSigma

Usage : $getSigma()$
Arguments : none
Value : a real value, the σ parameter of the distribution

©2007 EDF - EADS - PhiMeca

getR

Usage : *getR*()
Arguments : none
Value : a real value, the r parameter of the distribution

getT

Usage : *getT*()
Arguments : none
Value : a real value, the t parameter of the distribution

Details :

density probability function :

$$f(x) = \frac{(x-a)^{(r-1)}(b-x)^{(t-r-1)}}{(b-a)^{(t-1)}B(r,t-r)}\mathbf{1}_{[a,b]}(x)$$

relation between parameters set :

$$\begin{aligned}\mu &= a + \frac{(b-a)r}{t} \\ \sigma &= \frac{(b-a)}{t}\sqrt{\frac{r(t-r)}{(t+1)}}\end{aligned}$$

Links : see docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.2.2 Exponential

This class inherits from the Distribution class.

Usage : Main parameters set : $Exponential(\lambda, \gamma)$

Default construction : $Exponential()$

Arguments :

λ : real value, constraint : $\lambda > 0$

γ : real value

Value : an Exponential. In the default construction, we use the $Exponential(lambda, gamma) = Exponential(1.0, 0.0)$ definition.

Some methods :

getGamma

Usage : $getGamma()$
Arguments : none
Value : a real value, the γ parameter of the distribution

getLambda

Usage : $getLambda()$
Arguments : none
Value : a real value, the λ parameter of the distribution

Details :

density probability function :

$$f(x) = \lambda e^{-\lambda(x-\gamma)} \mathbf{1}_{[\gamma,+\infty[}(x)$$

relation between parameters set :

$$\mu = \frac{1}{\lambda} \quad \text{where} \quad \mu = \mathrm{E}[X]$$
$$\sigma = \frac{1}{\lambda} \quad \text{where} \quad \sigma = \sqrt{\mathrm{Var}[X]}$$

Links : voir docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.2.3 Gamma

This class inherits from the Distribution class.

Usage :

Main parameters set : $Gamma(k, \lambda, \gamma)$

Second parameters set : $Gamma(\mu, \sigma, \gamma, Gamma.MUSIGMA)$

Default construction : $Gamma()$

Arguments :

k : integer value constraint : $k > 0$

λ : real value, constraint : $\lambda > 0$

γ : real value,

μ : real value, mean value

σ : real value, standard deviation, constraint : $\sigma > 0$

Value : a Gamma. In the default construction, we use the $Gamma(k, lambda, gamma) = Gamma(1.0, 1.0, 0.0)$ definition.

Some methods :

getGamma

Usage : $getGamma()$
Arguments : none
Value : a real value, the γ parameter of the distribution

getK

Usage : $getK()$
Arguments : none
Value : a real value, the k parameter of the distribution

getLambda

Usage : $getLambda()$
Arguments : none
Value : a real value, the λ parameter of the distribution

Details :

density probability function :

$$f(x) = \frac{\lambda}{\Gamma(k)}(\lambda(x-\gamma))^{(k-1)}e^{-\lambda(x-\gamma)}\mathbf{1}_{[\gamma,+\infty[}(x)$$

©2007 EDF - EADS - PhiMeca

relation between parameters set :

$$\mu = \frac{k}{\lambda} + \gamma \quad \text{where} \quad \mu = \mathrm{E}[X]$$
$$\sigma = \frac{\sqrt{k}}{\lambda} \quad \text{where} \quad \sigma = \sqrt{\mathrm{Var}[X]}$$

Links : voir docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.2.4 Geometric

This class inherits from the Distribution class.

Usage : Main parameters set : $Geometric(p)$

Arguments : p : a real value, constraint : $0 < p < 1$

Value : Geometric

Some methods :

getP

Usage : $getP()$
Arguments : none
Value : a real positive value <1, the p parameter of the distribution

Details :

probability distribution:

$$\mathbb{P}(k) = (1-p)^{k-1}p,\ k \in \mathbb{N}^*$$

relation between parameters set :

$$\mu = \frac{1}{p} \quad \text{where} \quad \mu = \mathrm{E}[X]$$
$$\sigma = \sqrt{\frac{1-p}{p^2}} \quad \text{where} \quad \sigma = \sqrt{\mathrm{Var}[X]}$$

Links : see docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.2.5 Gumbel

This class inherits from the Distribution class.

Usage :

Main parameters set : $Gumbel(\alpha, \beta)$

Second parameters set : $Gumbel(\mu, \sigma, 1)$

Default construction : $Gumbel()$

Arguments :

α : a real value, the scale parameter (the inverse), constraint : $\alpha > 0$

β : a real value, location parameter

μ : a real value, the mean value

σ : a real value, standard deviation, constraint : $\sigma > 0$

Value : a Gumbel. In the default construction, we use the $Gumbel(alpha, beta) = Gumbel(1.0, 1.0)$ definition.

Some methods :

getAlpha

Usage : $getAlpha()$
Arguments : none
Value : a real value, the α of the considered distribution

getBeta

Usage : $getBeta()$
Arguments : none
Value : a real value, the β of the considered distribution

getMu

Usage : $getMu()$
Arguments : none
Value : a real value, the μ parameter of the distribution

getSigma

Usage : $getSigma()$
Arguments : none
Value : a real value, the σ parameter of the distribution

Details :

density probability function :

$$f(x) = \alpha e^{-\alpha(x-\beta)-e^{-\alpha(x-\beta)}}$$

©2007 EDF - EADS - PhiMeca

relation between parameters set :

$$\mu = \beta + \frac{c}{\alpha} \qquad \text{where } c \text{ is the Euler-Mascheroni constant} \qquad (c \approx 0.5772156649)$$
$$\sigma = \frac{1}{\sqrt{6}} \frac{\pi}{\alpha}$$

$$\text{where} \qquad \mu = \mathrm{E}[X] \qquad \sigma = \sqrt{\mathrm{Var}[X]}$$

Links : see docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.2.6 Histogram

This class inherits from the Distribution class.

Usage :

Main parameters set : $Histogram(first, Coll)$

Arguments :

$first$: a real value, the upper bound of the distribution

$Coll$: an HistogramPairCollection, the collection of (h_i, l_i) where h_i is the height and l_i the width of each barplot of the Histogram

Value : an Histogram with normalized heights

Some methods :

getFirst

Usage : $getFirst()$

Arguments : none

Value : a real value, the $first$ parameter of the considered distribution

getPairCollection

Usage : $getPairCollection()$

Arguments : none

Value : a HistogramPairCollection, the $Coll$ parameter of the considered distribution

Details :

density probability function :

$$f(x) = \sum_{i=1}^{n} H_i \, \mathbf{1}_{[x_i, x_{i+1}]}(x)$$

where

$H_i = h_i / S$ is the normalized heights, with $S = \sum_{i=1}^{n} h_i \, l_i$ being the initial surface of the histogram.

$l_i = x_{i+1} - x_i$, $1 \leq i \leq n$

n is the size of the HistogramPairCollection

Each $getMethod$ is associated to a $setMethod$.

©2007 EDF - EADS - PhiMeca

2.2.7 Logistic

This class inherits from the Distribution class.

Usage :

Main parameters set : $Logistic(\alpha, \beta)$

Default construction : $Logistic()$

Arguments :

α : a real value, mean value

β :a real value, scale parameter, constraint : $\beta \geq 0$

Value : Logistic. In the default construction, we use the $Logistic(alpha, beta) = Logistic(0.0, 1.0)$ definition.

Some methods :

getAlpha

Usage : $getAlpha()$

Arguments : none

Value : a real value, the α parameter of the considered distribution

getBeta

Usage : $getBeta()$

Arguments : none

Value : a real value, the β parameter of the considered distribution

Details :

density function :

$$f(x) = \frac{\exp\left(\frac{x-\alpha}{\beta}\right)}{\beta\left[1+\exp\left(\frac{x-\alpha}{\beta}\right)\right]^2}\mathbf{1}_{[\alpha,+\infty[}(x)$$

relation between parameters set :

$$\begin{aligned} \mu &= \alpha \\ \sigma &= \sqrt{\frac{1}{3}\pi^2\beta^2} \end{aligned}$$

where $\mu = \mathrm{E}[X]$ $\sigma = \sqrt{\mathrm{Var}[X]}$

Links : see docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.2.8 LogNormal

This class inherits from the Distribution class.

Usage :

Main parameters set : $LogNormal(\mu_\ell, \sigma_\ell, \gamma)$

Second parameters set : $LogNormal(\mu, \sigma, \gamma, LogNormal.MUSIGMA)$

Third parameters set : $LogNormal(\mu, \sigma/\mu, \gamma, LogNormal.MU_SIGMAOVERMU)$

Default construction : $LogNormal()$

Arguments :

μ_ℓ : a real value, mean value of $\log(X)$,

σ_ℓ : a real value, standard deviation of $\log(X)$, constraint : $\sigma_\ell > 0$

γ : a real value

μ : a real value, mean value, constraint : $\mu > \gamma$

σ : a real value, standard deviation, constraint : $\sigma > 0$

Value : a LogNormal . In the default construction, we use the $LogNormal(mu_l, sigma_l, gamma) = LogNormal(0.0, 1.0,$ definition.

Some methods :

getGamma

Usage : $getGamma()$
Arguments : none
Value : a real value, the γ parameter of the LogNormal distribution

getMu

Usage : $getMu()$
Arguments : none
Value : a real value, the μ parameter of the LogNormaldistribution

getMuLog

Usage : $getMuLog()$
Arguments : none
Value : a real value, the μ_ℓ parameter of the LogNormal distribution

getSigma

Usage : $getSigma()$
Arguments : none
Value : a real value, the σ parameter of the LogNormal distribution

getSigmaLog

©2007 EDF - EADS - PhiMeca

Usage : *getSigmaLog*()
Arguments : none
Value : a real value, the σ_ℓ parameter of the LogNormal distribution

getSigmaOverMu

Usage : *getSigmaOverMu*()
Arguments : none
Value : a real value, the σ/μ parameter of the considered distribution

Details :

density probability function :

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma_\ell(x-\gamma)} \, e^{-\frac{1}{2}(\frac{\log(x-\gamma)-\mu_\ell)}{\sigma_l})^2} \mathbf{1}_{[\gamma,+\infty[}(x)$$

relation between parameters set :

$$\begin{aligned} \mu &= e^{\mu_\ell+\sigma_\ell^2/2} + \gamma \\ \sigma &= e^{\mu_\ell+\sigma_\ell^2/2}\sqrt{\left(e^{\sigma_\ell^2}-1\right)} \end{aligned}$$

where $\mu = \mathrm{E}[X]$ $\sigma = \sqrt{\mathrm{Var}[X]}$

Links : see docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.2.9 MultiNomial

This class inherits from the Distribution class.

Usage : Main parameters set : $MultiNomial(p, N)$

Arguments :

p : NumericalPoint of dimension n, constraint : $0 \leq p_i \leq 1$, $q = \sum_{i=1}^{n} p_i \leq 1$

N : an integer,

Value : a Multinomial

Some methods :

getN

Usage : $getN()$
Arguments : none
Value : a integer, the N parameter of the considered distribution

getP

Usage : $getP()$
Arguments : none
Value : a NumericalPoint, the p parameter of the considered distribution

Details :

probability function :

$$P(\underline{X} = \underline{x}) = \frac{N!}{x_1! \dots x_n!(N-s)!} p_1^{x_1} \dots p_n^{x_n} (1-q)^{N-s}$$

with $0 \leq p_i \leq 1$, $x_i \in \mathbb{N}$, $q = \sum_{i=1}^{n} p_i \leq 1$, $s = \sum_{i=1}^{n} x_i \leq N$

relation between parameters set :

$$\begin{aligned} \mu_i &= n\, p_i \\ \sigma_i &= \sqrt{n\, p_i\, (1-p_i)} \\ \sigma_{i,j} &= -n\, p_i\, p_j \end{aligned}$$

where $\mu_i = \mathrm{E}\left[X\right]_i$ $\sigma_i = \sqrt{\mathrm{Var}\left[X\right]_i}$ $\sigma_{i,j} = \mathrm{Cov}\left[(\right] X_i, X_j)$

Links : see docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.2.10 NonCentralStudent

This class inherits from the Distribution class.

Usage :

Main parameters set : $NonCentralStudent(\nu, \delta, \gamma)$
Default construction : $NonCentralStudent()$

Arguments :

ν : a real positive value, generalised number degree of freedom, constraint : $\nu > 0$
δ : a real value
γ : a real value

Value : Student. In the default construction, we use the $NonCentralStudent(nu, delta, gamma) = NonCentralStudent$ definition.

Some methods :

getMu
Usage : $getMu()$
Arguments : none
Value : a real value, the μ parameter of the NonCentralStudent distribution

getDelta
Usage : $getDelta()$
Arguments : none
Value : a real value, the δ parameter of the NonCentralStudent distribution

getGamma
Usage : $getGamma()$
Arguments : none
Value : a real value, the γ parameter of the NonCentralStudent distribution

Details :

density function :

$$f(x) = \frac{1}{\sqrt{\nu} B(\frac{1}{2}, \frac{\nu}{2})} \left(1 + \frac{(x-\mu)^2}{\nu}\right)^{-\frac{1}{2}(\nu+1)}$$

where B is the β–function

relation between parameters set :

$$\sigma = \sqrt{\frac{\nu}{\nu - 2}}$$

where $\mu = \mathrm{E}[X]$ $\sigma = \sqrt{\mathrm{Var}[X]}$

Links : see docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.2.11 Normal

Usage :

$Normal(mean, standardDeviation)$

$Normal(dim)$

$Normal(\mu, \sigma, R)$

$Normal(\mu, \Sigma)$

Default construction : $Normal()$

Arguments :

mean : a scalar, the mean value of the 1D normal distribution

standardDeviation : a scalar, the standard deviation value of the 1D normal distribution

dim, an integer : the dimension of the Normal distribution

μ : a NumericalPoint, the mean of the Distribution

σ : a NumericalPoint, the standard deviation of each component, constraint : $\sigma[i] > 0, \forall i$

R : a CorrelationMatrix, the linear correlation matrix of the Normal distribution

Σ : a CovarianceMatrix, the covariance matrix of the Normal distribution

value :

while using the first usage, a 1D normal distribution with *mean* as mean value, *standardDeviation* as standard deviation

while using the second usage, a normal distribution of dimension *dim*, with $\underline{0}$ mean value vector, $\underline{1}$-standard deviation vector and identity correlation matrix

while using the third usage, a nD normal distribution with μ as mean vector, σ as standard deviation vector and R as linear correlation matrix

while using the fourth usage, a nD normal distribution with μ as mean vector, Σ as covariance matrix

while using the defalt usage, a 1D normal distribution with O mean and unit variance.

Some methods :

getMu

Usage : $getMu()$
Arguments : none
Value : a NumericalPoint, the μ parameter of the distribution

getSigma

Usage : $getSigma()$
Arguments : none
Value : a NumericalPoint, the σ parameter of the distribution

getCorrelationMatrix

Usage : $getCorrelationMatrix()$

©2007 EDF - EADS - PhiMeca

Arguments : none
Value : a CorrelationMatrix, the R parameter of the distribution

Details : probability density function :

$$\frac{1}{(2\pi)^{\frac{n}{2}}(\det\mathbf{\Sigma})^{\frac{1}{2}}}e^{-\frac{1}{2}(x-\mu)^t\mathbf{\Sigma}^{-1}(x-\mu)}$$

with $\Sigma = \Lambda(\sigma)R\Lambda(\sigma)$, $\Lambda(\sigma) = diag(\sigma)$, R symetric, definite and positive, $\sigma_i > 0$.

Links : see docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.2.12 Poisson

This class inherits from the Distribution class.

Usage : Main parameters set : $Poisson(\lambda)$

Arguments : λ : real value, mean and variance value, constraint : $\lambda > 0$

Value : Poisson

Some methods :

getLambda

Usage : $getLambda()$
Arguments : none
Value : a real positive value, the λ parameter of the considered distribution

Details :

probability function :

$$\mathbb{P}(k) = \frac{\lambda^k}{k!} e^{-\lambda},\ k \in \mathbb{N}$$

relation between parameters set :

$$\begin{aligned} \mu &= \lambda \\ \sigma &= \sqrt{\lambda} \end{aligned}$$

where $\mu = \mathrm{E}[X]$ $\sigma = \sqrt{\mathrm{Var}[X]}$

Links : see docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.2.13 Student

This class inherits from the Distribution class.

Usage :

Main parameters set : $Student(\nu, \mu)$

Default construction : $Student()$

Arguments :

ν : a real positive value, generalised number degree of freedom, constraint : $\nu > 2$

μ : a real value, the mean value

Value : Student . In the default construction, we use the $Student(nu, mu) = Student(3.0, 0.0)$ definition.

Some methods :

getMu

Usage : $getMu()$
Arguments : none
Value : a real value, the μ parameter of the Beta distribution

getNu

Usage : $N()$
Arguments : none
Value : a real value, the ν parameter of the Beta distribution

Details :

density function :

$$f(x) = \frac{1}{\sqrt{\nu} B(\frac{1}{2}, \frac{\nu}{2})} \left(1 + \frac{(x-\mu)^2}{\nu}\right)^{-\frac{1}{2}(\nu+1)}$$

where B is the $\beta-$function and $\nu \geq 2$

relation between parameters set :

$$\sigma = \sqrt{\frac{\nu}{\nu - 2}}$$

where $\mu = \mathrm{E}[X]$ $\sigma = \sqrt{\mathrm{Var}[X]}$

Links : see docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.2.14 Triangular

This class inherits from the Distribution class.

Usage :

Main parameters set : $Triangular(a, m, b)$

Default construction : $Triangular()$

Arguments :

a : a real value, the lower bound

b : a real value, the upper bound, constraint : $b \geq a$

m : a real value, the mode, constant, $b \geq m \geq a$

Value : Triangular. In the default construction, we use the $Triangular(a, m, b) = Triangular(-1.0, 0.0, 1.0)$ definition.

Some methods :

getA

Usage : $getA()$
Arguments : none
Value : a real value, the a parameter of the Triangular distribution

getB

Usage : $getB()$
Arguments : none
Value : a real value, the b parameter of the Triangular distribution

getM

Usage : $getM()$
Arguments : none
Value : a real value, the m parameter of the Triangular distribution

Details :

density function :

$$f(x) = \begin{cases} \dfrac{2(x-a)}{(m-a)(b-a)} & a \leq x \leq m \\ \dfrac{2(b-x)}{(b-m)(b-a)} & m \leq x \leq b \\ 0 & \text{elsewhere} \end{cases}$$

©2007 EDF - EADS - PhiMeca

relation between parameters set :

$$\begin{aligned} \mu &= \frac{1}{3}(a+m+b) \\ \sigma &= \sqrt{\frac{1}{18}(a^2+b^2+m^2-ab-am-bm)} \end{aligned}$$

where $\mu = \mathrm{E}[X]$ $\sigma = \sqrt{\mathrm{Var}[X]}$

Links : see docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.2.15 TruncatedNormal

This class inherits from the Distribution class.

Usage :

Main parameters set : $TruncatedNormal(\mu_n, \sigma_n, a, b)$
Default construction : $TruncatedNormal()$

Arguments :

μ_n : a real value which corresponds to the mean of the associated non truncated normal
σ_n : a real value which corresponds to the standard deviation of the associated non truncated normal
a : a real value, the lower bound
b : a real value, the upper bound, constraint : $b \geq a$

Value : TruncatedNormal . In the default construction, we use the $TruncatedNormal(mu_n, sigma_n, a, b) = TruncatedNormal(0.0, 1.0, -1.0, 1.0)$ definition.

Some methods :

getA
Usage : $getA()$
Arguments : none
Value : a real value, the a parameter of the TruncatedNormal distribution

getB
Usage : $getB()$
Arguments : none
Value : a real value, the b parameter of the TruncatedNormal distribution

getMu
Usage : $getMu()$
Arguments : none
Value : a real value, the μ_n parameter of the TruncatedNormal distribution

getSigma
Usage : $getSigma()$
Arguments : none
Value : a real value, the σ_n parameter of the TruncatedNormal distribution

Details :

probability density function :

$$f(x) = \frac{\frac{1}{\sigma_n}\,\phi(\frac{x-\mu_n}{\sigma_n})}{\Phi(\frac{b-\mu_n}{\sigma_n}) - \Phi(\frac{a-\mu_n}{\sigma_n})}\mathbf{1}_{[a,b]}(x)$$

(where ϕ and Φ are, respectively, the probability density distribution function and the cumulative distribution function of a standard normal distribution)

©2007 EDF - EADS - PhiMeca

relation between parameters set :

$$\begin{aligned}
\mu &= \mu_n - \sigma_n \frac{\phi(b_{red}) - \phi(a_{red})}{\Phi(b_{red}) - \Phi(a_{red})} \\
\sigma &= \sigma_n \left\{ 1 - \frac{b_{red}\,\phi(b_{red}) - a_{red}\,\phi(a_{red})}{\Phi(b_{red}) - \Phi(a_{red})} - \left[\frac{\phi(b_{red}) - \phi(a_{red})}{\Phi(b_{red}) - \Phi(a_{red})} \right]^2 \right\}^{1/2}
\end{aligned}$$

where

$$a_{red} = \frac{a - \mu_n}{\sigma_n} \qquad b_{red} = \frac{b - \mu_n}{\sigma_n}$$

and

$$\mu = \mathrm{E}\left[X\right] \qquad \sigma = \sqrt{\mathrm{Var}\left[X\right]}$$

Links : see docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.2.16 Uniform

This class inherits from the Distribution class.

Usage :

Main parameters set : $Uniform(a, b)$

Default construction : $Uniform()$

Arguments :

a : a real value, the lower bound

b : a real value, the upper bound, constraint : $b \geq a$

Value : Uniform. In the default construction, we use the $Uniform(a, b) = Uniform(-1.0, 1.0)$ definition.

Some methods :

getA

Usage : $getA()$
Arguments : none
Value : a real value, the a parameter of the Uniform distribution

getB

Usage : $getB()$
Arguments : none
Value : a real value, the b parameter of the Uniform distribution

Details :

density function :

$$f(x) = \begin{cases} \dfrac{1}{(b-a)} & a \leq x \leq b \\ 0 & \text{elsewhere} \end{cases}$$

relation between parameters set :

$$\begin{aligned} \mu &= \frac{a+b}{2} \\ \sigma &= \frac{b-a}{2\sqrt{3}} \end{aligned}$$

where $\mu = \mathrm{E}[X]$ $\sigma = \sqrt{\mathrm{Var}[X]}$

Links : see docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.2.17 Uniform

This class inherits from the Distribution class.

Usage : Main parameters set : $Uniform(a, b)$

Arguments :

a : a real value, the lower bound

b : a real value, the upper bound, constraint : $b \geq a$

Value : TruncatedNormal

Some methods :

getA

Usage : $getA()$
Arguments : none
Value : a real value, the lower bound

getB

Usage : $getB()$
Arguments : none
Value : a real value, the upper bound

Details :

probability density function :

$$f(x) = \frac{1}{b-a}\mathbf{1}_{[a,b]}(x)$$

Links : see docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.2.18 UserDefined

This class inherits from the Distribution class.

Usage : $UserDefined(Coll)$

Arguments : $Coll$: a UserDefinedPairCollection, constraint : the collection of UserDefinedPair of the UserDefinedPairsCollection must be such that $\sum_1^n p_i = 1.0$

Value : a UserDefined

Some methods :

getPairCollection

Usage : $getPairCollection()$

Arguments : none

Value : a UserDefinedPairCollection, the $Coll$ parameter of the considered distribution

Details :

probability function :

$$\mathbb{P}(x_i) = p_i, \quad i = 1, \ldots, n$$

where

(x_i, p_i) and $i = 1, \ldots, n$ are respectively a NumericalPoint and its associated probability

n is the size of the UserDefinedPairCollection

One must have

$$\sum_{i=1}^{n} p_i = 1$$

Each $getMethod$ is associated to a $setMethod$.

©2007 EDF - EADS - PhiMeca

2.2.19 Weibull

This class inherits from the Distribution class.

Usage :

Main parameters set : $Weibull(\alpha, \beta, \gamma)$

Second parameter set : $Weibull(\mu, \sigma, \gamma, 1)$

Default construction : $Weibull()$

Arguments :

α : a real value, the shape parameter, constraint : $\alpha > 0$

β : a real value, the scale parameter, constraint : $\beta > 0$

γ : a real value, the location parameter

μ : a real value, the mean value,

σ : a real value, the standard deviation value, constraint : $\sigma > 0$

Value : a Weibull. In the default construction, we use the $Weibull(\alpha, \beta, \gamma) = Weibull(1.0, 1.0, 0.0)$ definition.

Some methods :

getAlpha

Usage : $getAlpha()$
Arguments : none
Value : a real value, the α of the considered distribution

getBeta

Usage : $getBeta()$
Arguments : none
Value : a real value, the β of the considered distribution

getGamma

Usage : $getGamma()$
Arguments : none
Value : a real value, the γ parameter of the considered distribution

getMu

Usage : $getMu()$
Arguments : none
Value : a real value, the μ parameter of the considered distribution

getSigma

Usage : $getSigma()$

©2007 EDF - EADS - PhiMeca

Arguments : none
Value : a real value, the σ parameter of the considered distribution

Details :

density function :

$$f(x) = \frac{\beta}{\alpha}\left(\frac{x-\gamma}{\alpha}\right)^{\beta-1} \exp\left[-\left(\frac{x-\gamma}{\alpha}\right)^{\beta}\right] \mathbf{1}_{[\gamma,+\infty[}(x)$$

relation between parameters set :

$$\begin{aligned} \mu &= \alpha\,\Gamma\left(1+\frac{1}{\beta}\right)+\gamma \\ \sigma &= \alpha\sqrt{\Gamma\left(1+\frac{2}{\beta}\right)-\Gamma^2\left(1+\frac{2}{\beta}\right)} \end{aligned}$$

where Γ is the Γ-function and

$$\mu = \mathrm{E}[X] \qquad\qquad \sigma = \sqrt{\mathrm{Var}[X]}$$

Links : see docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.3 Truncated distribution

2.3.1 TruncatedDistribution

This class enables to truncate any distribution whithin a specified range which one of the bounds may be infinite. It offers the methods of the Distribution class.

Usage :

$TruncatedDistribution(distribution, lowerBound, upperBound)$

$TruncatedDistribution(distribution, bound, TruncatedDistribution.UPPER)$

$TruncatedDistribution(distribution, bound, TruncatedDistribution.LOWER)$

Arguments :

distribution : a Distribution

lowerBound : a real, the new lower bound of the distribution : the distribution range is $[lowerBound, \infty[$ or $[lowerBound, max[$ if the distribution is already bounded by max

upperBound : a real, the new upper bound of the distribution : the distribution range is $[-\infty, upperBound[$ or $[min, upperBound[$ if the distribution is already bounded by min

Value : a Distribution

Links : see docref_B122_Copules_en

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.4 Composed distribution

2.4.1 Copula

This class inherits from the Distribution class.

Usage : *Copula*(*distributionImplementation*)

Arguments : *distributionImplementation* : a DistributionImplementation, which must verify the properties of a copula. This distribution can be a IndependentCopula, NormalCopula, ClaytonCopula, GumbelCopula or FrankCopula.

Value : a Copula

Links : see docref_B122_Copules_en

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.4.2 ClaytonCopula

This class inherits from the Distribution class.

Usage :

ClaytonCopula()
ClaytonCopula(*theta*)

Arguments : *theta* : a real, the only parameter of the Clayton copula, which PDF is : $\left(u_1^{-\theta} + u_2^{-\theta} - 1\right)^{-1/\theta}$, for $u_i \in [0, 1]$

Value :

In the first usage, a ClaytonCopula of dimension 2 with $\theta = 2.0$,
In the second usage, a ClaytonCopula of dimension 2 with the θ specified.

Links : see docref_B122_Copules_en

©2007 EDF - EADS - PhiMeca

2.4.3 FrankCopula

This class inherits from the Distribution class.

Usage :

$FrankCopula()$
$FrankCopula(theta)$

Arguments : *theta* : a real, the only parameter of the Gumbel copula, which PDF is :

$$-\frac{1}{\theta}\log\left(1+\frac{(e^{-\theta u_1}-1)(e^{-\theta u_2}-1}{e^{-\theta}-1}\right)$$

, for $u_i \in [0,1]$

Value :

In the first usage, a FrankCopula of dimension 2 with $\theta = 2.0$,
In the second usage, a FrankCopula of dimension 2 with the θ specified.

Links : see docref_B122_Copules_en

©2007 EDF - EADS - PhiMeca

2.4.4 GumbelCopula

This class inherits from the Distribution class.

Usage :

GumbelCopula()
GumbelCopula(*theta*)

Arguments : *theta* : a real, the only parameter of the Gumbel copula, which PDF is :

$$\exp\left(-\left((-\log(u_1))^\theta + (-\log(u_2))^\theta\right)^{1/\theta}\right)$$

, for $u_i \in [0,1]$

Value :

In the first usage, a GumbelCopula of dimension 2 with $\theta = 2.0$,
In the second usage, a GumbelCopula of dimension 2 with the θ specified.

Links : see docref_B122_Copules_en

©2007 EDF - EADS - PhiMeca

2.4.5 IndependentCopula

This class inherits from the Distribution class.

Usage :

IndependentCopula()
IndependentCopula(*dim*)

Arguments : *dim* : an integer, the dimension of the copula

Value :

In the first usage, a IndependentCopula of dimension 1,
In the second usage, a IndependentCopula of the dimension *dim* specified.

Links : see docref_B122_Copules_en

©2007 EDF - EADS - PhiMeca

2.4.6 NormalCopula

This class inherits from the Distribution class.

Usage :

$NormalCopula()$

$NormalCopula(R)$

Arguments : R : a CorrelationMatrix which is not the Kendall nor the Spearman rank correlation matrix of the distribution. The R matrix can be evaluated from the Spearman or Kendall correlation matrix

Value :

In the first usage, a NormalCopula of dimension 1

In the second usage, a NormalCopula with the correlation matrix R specified.

Some methods :

getNormalCorrelationFromKendallCorrelation

Usage : $NormalCopulagetNormalCorrelationFromKendallCorrelation(K)$

Arguments : K : a CorrelationMatrix, it must be the Kendall correlation matrix of the considered random vector

Value : a CorrelationMatrix, the correlation matrix of the normal copula evaluated from the Kendall correlation matrix K

getNormalCorrelationFromSpearmanCorrelation

Usage : $NormalCopula.getNormalCorrelationFromSpearmanCorrelation(S)$

Arguments : S : a CorrelationMatrix, it must be the Spearman correlation matrix of the considered random vector

Value : a CorrelationMatrix, the correlation matrix of the normal copula evaluated from the Spearman correlation matrix S

Links : see docref_B122_Copules_en

©2007 EDF - EADS - PhiMeca

2.4.7 ComposedCopula

This class inherits from the Distribution class.

Usage : *ComposedCopula*(*copulaCollection*)

Arguments : *copulaCollection* : a CopulaCollection

Value : a ComposedCopula, defined as the product of the initial copulas. For example, if C_1 and C_2 are two copulas respectively of $\mathcal{R}^{n_1}$ and $\mathcal{R}^{n_2}$, we can create the copula of a random vector of $\mathcal{R}^{n_1+n_2}$, noted C as follows :

$$C(u_1, \cdots, u_n) = C_1(u_1, \cdots, u_{n_1}) C_2(u_{n_1+1}, \cdots, u_{n_1+n_2})$$

It means that both subvectors $(u_1, \cdots, u_{n_1}$ and $(u_{n_1+1}, \cdots, u_{n_1+n_2})$ of $\mathcal{R}^{n_1}$ and $\mathcal{R}^{n_2}$ are independent.

Some methods :

getCopulaCollection

Usage : *getCopulaCollection*()

Arguments : none

Value : a CopulaCollection, the collection of copulas from which the ComposedCopula is built

Links : see docref_B_JoinedCDF_en

©2007 EDF - EADS - PhiMeca

2.4.8 ComposedDistribution

This class inherits from the Distribution class.

Usage : *ComposedDistribution*(*distributionCollection*, *copula*)

Arguments :

distributionCollection : a DistributionCollection

copula : a Copula

Value : a ComposedDistribution

Some methods :

getDistributionCollection

Usage : *getDistributionCollection*()

Arguments : none

Value : a DistributionCollection, the collection of distributions from which the ComposedDistribution is built

Links : see docref_B_JoinedCDF_en

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.5 Mixture

2.5.1 Mixture

A Mixture is a distribution such that its probability density function is a linear combination of probability density functions, with the linear combination coefficients greater or equal to zero such that their sum is equal to 1.
It is important to note that the linear combination coefficients are given through the *weight* attribute of each component of the DistributionCollection, thanks to the command *DistributionCollection[i].setWeight(coefficient)*.

Usage : *Mixture*(*distributionCollection*)

Arguments : *distributionCollection* : a DistributionCollection, the collection of the distributions which composes the linear combination

Value : a Mixture

Some methods :

getDistributionCollection

Usage : *getDistributionCollection*()
Arguments : none
Value : a DistributionCollection the collection of distribution from which the Mixture is built

Details :

probability density function :

$$f(x) = \sum \alpha_i p_i(x)$$

with $\sum \alpha_i = 1$

Links : see docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.6 KernelMixture

2.6.1 KernelMixture

A KernelMixture is a distribution built from a NumericalSample, such that its probability density function is a linear combination of the kernel specified by the User, centered on each point of the NumericalSample, which standard deviation is the bandwith specified by the User. It is important to note that the linear combination coefficients are all equal.

Usage : $KernelMixture(kernel, bandwidth, sample)$

Arguments : *distributionCollection* : a DistributionCollection, the collection of the distributions which composes the linear combination

Value : a KernelMixture

Some methods :

getBandwidth

Usage : $getBandwidth()$

Arguments : none

Value : a NumericalPoint, the bandwith of the kernel mixture, (see equation below for dimension 1). The bandwith is the same at each point of the NumericalSample

getKernel

Usage : $getKernel()$

Arguments : none

Value : a Distribution, the kernel K of the mixture, (see equation below for dimension 1)

getSample

Usage : $getSample()$

Arguments : none

Value : a NumericalSample, the NumericalSample of the mixture, (see equation below for dimension 1)

Details :

Probability density function in dimension 1 :

$$f(x) = \sum \frac{1}{nh} K(\frac{X_i - x}{h}), x \in \mathbb{R}$$

where $(X_1, \ldots, X_n)$ is a NumericalSample

Links : see docref_B121_ChoixLoi

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

2.6.2 KernelSmoothing

The class KernelSmoothing enables to build some kernels used to fit a distribution to a numerical sample.

Usage :

$KernelSmoothing()$

$KernelSmoothing(Distribution(myDistribution))$

$KernelSmoothing(DistributionImplentation())$

Arguments :

myDistribution : a 1D Distribution of any kind

$DistributionImplentation()$: default constructor of the 1D UsualDistribution. For example, $Uniform()$, $Triangular()$, ...

Value : a Distribution

In the first usage, the kernel is the kernel product of 1D Normal(1.0, 0.0). The dimension of the product is detected from the numerical sample.

In the second usage, the kernel is the kernel product of 1D distributions specified by *myDistribution*. Care : the kernel smoothing method is all the more efficient than the kernel is symetric with respect to 0.0. The dimension of the product is detected from the numerical sample.

In the third usage, the kernel is the kernel product of the default constructions of the 1D UsualDistributions. Note that the default constructor of a UsualDistribution builds a distribution which is symetric with respect to 0.0 when it is possible. The dimension of the product is detected from the numerical sample.

Some methods :

buildImplementation

Usage :

$buildImplementation(sample)$

$buildImplementation(sample, boundaryCorrection)$

$buildImplementation(sample, bandwidth)$

$buildImplementation(sample, bandwidth, boundaryCorrection)$

Arguments :

sample : a NumericalSample, the numerical sample from wich the kernel mixture is built

boundaryCorrection : a Bool which indicates if it is necessary to make a boundary treatment (according to the miroring technique)

bandwidth : a NumericalPoint, the bandwith of the kernel product. The dimension is detected from the numerical sample *sample* and evaluated according to the Scott rule.

Value : a Distribution

computeSilvermanBandwidth

Usage : $computeSilvermanBandwidth(sample)$

Arguments : *sample* : a NumericalSample, the numerical sample from wich the kernel mixture is built

©2007 EDF - EADS - PhiMeca

Value : a NumericalPoint, the bandwith automaticaly evaluated by Open TURNS from the numerical sample according to the Silverman rule

getBandwidth

Usage : *getBandwidth*()

Arguments : none

Value : a NumericalPoint, the bandwith of the kernel mixture, (see equation below for dimension 1). The bandwith is the same at each point of the NumericalSample

getKernel

Usage : *getKernel*()

Arguments : none

Value : a Distribution, the kernel adopted for the kernel smoothing

Details :

Probability density function in dimension 1 :

$$p_n(x) = \sum \frac{1}{nh} K(\frac{x - X^i}{h}), x \in \mathbb{R}$$

where $(X^1, \ldots, X^n)$ is a NumericalSample and K the kernel PDF,

Probability density function in dimension N :

$$p_n(\vec{x}) = \frac{1}{n} \sum_{i=1}^{i=N} \prod_{j=1}^{j=N} \frac{1}{h_j} K(\frac{x^j - X_i^j}{h^j})$$

where $\prod_{j=1}^{j=N} K(x^j)$ is the kernel product and $\underline{h} = (h^1, \cdots, h^N)$ the vector of bandwidth.

Links : see docref_B121_ChoixLoi

©2007 EDF - EADS - PhiMeca

2.7 Random vector

2.7.1 RandomVector

Usage :

$RandomVector(distribution)$

$RandomVector(function, distribution)$

Arguments :

distribution : a Distribution

$function$: a NumericalMathFunction

Value : a RandomVector, which is of type :

Usual : if created thanks to the first usage. In that case, the RandomVector has for distribution the one specified through *distribution*

Composite : if created thanks to the second usage. In that case, the RandomVector is defined as the function of a RandomVector X which distribution is *distribution* : $Y = function(X)$

Some methods :

getAntecedent

Usage : $getAntecedent()$

Arguments : no argument

Value : a RandomVector, only in the case of Composite RandomVector : the RandomVector X such that $Y = function(X)$.

getCovariance

Usage : $getCovariance()$

Arguments : no argument

Value : a CovarianceMatrix, only in the case of Usual RandomVector : the covariance of the considered RandomVector

getDistribution

Usage : $getDistribution()$

Arguments : no argument

Value : a Distribution, only in the case of Usual RandomVector : the distribution of the RandomVector

getDescription

Usage : $getDescription()$

Arguments : no argument

Value : a Description, the description of the Randomvector

getDimension

©2007 EDF - EADS - PhiMeca

Usage : *getDimension*()

Arguments : no argument

Value : an integer, the dimension of the RandomeVector

getMarginal

Usage :

getMarginal(i)

getMarginal(*indices*)

Arguments :

i : an integer which indicates the component concerned

indices : a Indices which regroups all the components concerned

Value : a RandomVector restricted to the concerned components.

Details : Let's note $\underline{Y} = (Y_1, \cdots, Y_n)^t$ a random vector and $I \in [1, n]$ a set of indices. If $\underline{Y}$ is a UsualRandomvector, the subvector is defined by $\underline{\tilde{Y}} = (Y_i)_{i \in I}{}^t$. If $\underline{Y}$ is a CompositeRandomVector, defined by $\underline{Y} = f(\underline{X})$ with $f = (f_1, \cdots, f_n)$, f_i some scalar functions, the sub vector is $\underline{\tilde{Y}} = (f_i(\underline{X}))_{i \in I}$.

getMean

Usage : ()

Arguments : no argument

Value : a NumericalPoint, only in the case of Usual RandomVector : the mean vector of the associated distribution

getName

Usage : *getName*()

Arguments : no argument

Value : a string, the name of the RandomVector

getNumericalSample

Usage : *getNumericalSample*()

Arguments : no argument

Value : a NumericalSample a sample of the random vector

isComposite

Usage : *isComposite*()

Arguments : no argument

Value : a boolean which indicates if the RandomVector is of type Composite or Usual.

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

3 Experiment planes

3.1 Experiment

Usage : *Experiment*(*expPlaneImplentation*)

Arguments : *expPlaneImplentation* : an ExperimentImplementation, which is Axial, Factorial, Composite or Box

Value : an Experiment

Some methods :

generate

Usage : *generate*()
Arguments : none
Value : a NumericalSample, the points which constitute the experiment plane

getLevels

Usage : *getLevels*()
Arguments : none
Value : a NumericalPoint, the levels of the experiment of the plane

getCenter

Usage : *getCenter*()
Arguments : none
Value : a NumericalPoint, the center of the experiment plane

.str()

Usage : *str*()
Arguments : no argument
Value : a string which describes the Experiment

Links see docref_C11_MinMaxPlanExp

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

3.2 Axial

This class inherits from the Experiment class.

Usage :

$Axial(center, levels)$
$Axial(dimension, levels)$

Arguments :

center : a NumericalPoint, the center of the experiment plane

levels : a NumericalPoint, the levels of the experiment of the plane

dimension : an integer, the dimension of the space where the experiment plane is created

Value : a Axial

if defined with the first usage, the experiment plane is centered on *center*

if defined with the second usage, the experiment plane is centered on the $center = 0$

Details :

Number of points generated : $1 + 2 * levels.getDimension() * dimension$

The axial plane generates a NumericalSample where :

the first point is the vector (*center*),

the following points are : each coordinate one at a time is equal to +/- levels[i], for each direction so on, until the last level.

It is possible to use the *scale, translate* methods of the NumericalSample in order to scale each direction and translate the grid structure onto the right center

Links see docref_C11_MinMaxPlanExp

©2007 EDF - EADS - PhiMeca

3.3 Factorial

This class inherits from the Experiment class.

Usage :

$Factorial(center, level)$

$Factorial(dimension, level)$

Arguments :

center : a NumericalPoint, the center of the experiment plane

level : a NumericalPoint, the levels of the experiment of the plane

dimension : an integer, the dimension of the space where the experiment plane is created

Value : a Factorial

if defined with the first usage, the experiment plane is centered on *center*

if defined with the second usage, the experiment plane is centered on the $center = 0$

Details :

Number of points generated : $1 + levels.getDimension() * 2^{dimension}$

The factorial plane generates a NumericalSample where :

the first point is the vector (*center*),

the following points are : all coordinates are equal to +/- levels[i] for each direction

It is possible to use the *scale, translate* methods of the NumericalSample in order to scale each direction and translate the grid structure onto the right center

Links see docref_C11_MinMaxPlanExp

©2007 EDF - EADS - PhiMeca

3.4 Composite

This class inherits from the Experiment class.

Usage :

$Composite(center, level)$
$Composite(dimension, level)$

Arguments :

center : a NumericalPoint, the center of the experiment plane

level : a NumericalPoint, the levels of the experiment of the plane

dimension : an integer, the dimension of the space where the experiment plane is created

Value : a Composite

if defined with the first usage, the experiment plane is centered on *center*

if defined with the second usage, the experiment plane is centered on the $center = 0$

Details :

A composite plane is the union of an axial and a factorial one

Number of points generated : $1 + levels.getDimension() * (2 * dimension + 2^{dimension})$

The composite plane generates a NumericalSample where :

the first point is the vector (*center*),

the following points are : one coordinate at a time is equal to +/- levels[i] for each direction

It is possible to use the *scale, translate* methods of the NumericalSample in order to scale each direction and translate the grid structure onto the right center

Links see docref_C11_MinMaxPlanExp

©2007 EDF - EADS - PhiMeca

3.5 Box

This class inherits from the Experiment class.

Usage :

$Box(levelsBox)$

Arguments :

levelsBox : a NumericalPoint, which specifies the number of intermediate points on each direction (one per direction) which regularly discretize the unit pavement $[-0.5, 0.5]^n$

Value : a Box, which regularly discretizes the unit pavement $[-0.5, 0.5]^n$ whith the specified number of intermediate points for each direction

Details :

Number of points generated : $\prod_{i=1}^{n}(2 + n_{level}(direction\ i))$

The box plane generates a NumericalSample where :

It is possible to use the *scale, translate* methods of the NumericalSample in order to scale each direction and translate the grid structure onto the right center

Links see docref_C11_MinMaxPlanExp

©2007 EDF - EADS - PhiMeca

4 Statistics on sample

4.1 Numerical Sample

4.1.1 NumericalSample

Usage :

$NumericalSample(size, dim)$
$NumericalSample(size, numericalPoint)$

Arguments :

$size$: an integer, the size of the NumericalSample
dim : an integer, the dimension each NumericalPoint of the NumericalSample
$numericalPoint$: a NumericalPoint

Value : a NumericalSample, containing $size$ NumericalPoint, each one equal to :

$\underline{0} \in \mathbb{R}^{dim}$ in the first usage
$numericalPoint$ in the second usage

Some methods :

[]
Usage : $NumericalSample[i]$
Arguments : i : an integer, constraint : $0 \leq i \leq size - 1$
Value : a NumericalPoint, the ith NumericalPoint of the NumericalSample

add
Usage : $add(x)$
Arguments : x : a NumericalPoint
Value : a NumericalSample, of size $size + 1$ where the last NumericalPoint has been added, equal to x

computeCovariance
Usage : $computeCovariance()$
Arguments : none
Value : a CovarianceMatrix, the covariance matrix of the NumericalSample (a dim^2 matrix)

computeEmpiricalCDF
Usage : $computeEmpiricalCDF(x)$
Arguments : x : a NumericalPoint

©2007 EDF - EADS - PhiMeca

Value : a numerical scalar, the Empirical Cumulative Distribution Function value of the NumericalSample at x

computeKendallTau

Usage : *computeKendallTau*()

Arguments : none

Value : a CorrelationMatrix, the Kendall rank correlation matrix of the NumericalSample

computeKurtosisPerComponent

Usage : *computeKurtosisPerComponent*()

Arguments : none

Value : a NumericalPoint, the value of the kurtosis of each component of the NumericalSample

computeMean

Usage : *computeMean*()

Arguments : none

Value : a NumericalPoint, the mean value vector of each component of the NumericalSample

computeMedianPerComponent

Usage : *computeMedianPerComponent*()

Arguments : none

Value : a NumericalPoint, the median value vector of each component of the NumericalSample

computePearsonCorrelation

Usage : *computePearsonCorrelation*()

Arguments : none

Value : a CorrelationMatrix, the Pearson correlation matrix of the NumericalSample (a dim^2 matrix)

computeQuantile

Usage : *computeQuantile*(p)

Arguments : p, a real value, constraint $0 \leq p \leq 1$, the value of a probability

Value : a NumericalPoint, the empirical quantile value associated to probability p, determined from the empirical CDF of the NumericalSample

computeQuantilePerComponent

Usage : *computeQuantilePerComponent*(p)

Arguments : p, a real value, constraint $0 \leq p \leq 1$, the value of a probability

Value : a NumericalPoint, the empirical quantile value associated to probability p for each component, determined from the empirical CDF of each component of the NumericalSample

computeSkewnessPerComponent

©2007 EDF - EADS - PhiMeca

Usage : *computeSkewnessPerComponent*()

Arguments : none

Value : a NumericalPoint, the skewness of each component of the NumericalSample

computeSpearmanCorrelation

Usage : *computeSpearmanCorrelation*()

Arguments : none

Value : a CorrelationMatrix, the Spearman correlation matrix of the NumericalSample (a dim^2 matrix)

computeStandardDeviation

Usage : *computeStandardDeviation*()

Arguments : none

Value : a SquareMatrix, the Cholesky factor $\underline{\underline{L}}$ of the covariance matrix $\underline{\underline{\Lambda}}$: $\underline{\underline{LL^t}} = \underline{\underline{\Lambda}}$, with $\underline{\underline{L}}$ triangular inferior

computeStandardDeviationPerComponent

Usage : *computeStandardDeviationPerComponent*()

Arguments : none

Value : a NumericalPoint, the standard Deviation value of each component of the NumericalSample

computeVariancePerComponent

Usage : *computeVariancePerComponent*()

Arguments : none

Value : a NumericalPoint, the variance value of each component of the NumericalSample

getDimension

Usage : *getDimension*()

Arguments : none

Value : an integer, the dimension of each point which constitutes the NumericalSample (it returns *dim*)

getMin

Usage : *getMin*()

Arguments : none

Value : a NumericalPoint, each element of the NumericalPoint corresponds to the minimum of each component of the NumericalSample

getMax

Usage : *getMax*()

Arguments : none

©2007 EDF - EADS - PhiMeca

Value : a NumericalPoint, each element of the NumericalPoint corresponds to the maximum of each component of the NumericalSample

getMarginal

Usage :

$getMarginal(i)$

$getMarginal(i)$

Arguments :

i : a UnsignedLong, (integer)

indices : a Indices (table of intergers)

Value :

a NumericalSample : the NumericalSample of same size as the initial NumericalSample, of dimension 1, corresponding to the $i+1$ coordinate of the NumericalPoints which constitute the initial NumericalSample

a NumericalSample : the NumericalSample of same size as the initial NumericalSample, of dimension $indices.getSize()$, corresponding to the associated coordinates of the NumericalPoints which constitute the initial NumericalSample. Care : indices are initialized to 0.

getSize

Usage : $getSize()$

Arguments : none

Value : an integer, the size of the NumericalSample (which means the number of points which constitute the NumericalSample (it returns *size*)

rank

Usage : $rank()$

Arguments : none

Value : a NumericalSample, where each value has been replaced by the value of its rank (order set component by component)

scale

Usage : $scale(factor)$

Arguments : $factor$: a NumericalPoint

Value : a NumericalSample, where the components [i] of each NumericalPoint have been multiplied by the corresponding value $factor[i]$

sort

Usage : $sort(i)$

Arguments : i : UnsignedLong (an integer)

Value : a NumericalSample of same size as the initial NumericalSample, of dimension 1, constitued by the $i+1$th component of the NumericalPoints that constitute the initial NumericalSample, all sorted in ascending order

©2007 EDF - EADS - PhiMeca

sortAccordingAComponent

Usage : *sortAccordingAComponent*(i)

Arguments : i : UnsignedLong (an integer)

Value : a NumericalSample of same size and dimension as the initial NumericalSample, where the NumericalPoints have been reordered such that the $(i + 1)$ component is sorted in ascending order

split

Usage : *split*(i)

Arguments : i : UnsignedLong (an integer)

Value : a NumericalSample which contains only the NumericalPoints of the initial NumericalSample corresponding to the indices k, where $k \geq i$. Care : The initial NumericalSample is modified and only contains the first i NumericalPoints (which means the ones corresponding to the indices k, where $k < i$).

str

Usage : *str*()

Arguments : none

Value : a string giving a brief description of the considered NumericalSample,

translate

Usage : *translate*(*translation*)

Arguments : *translation* : a NumericalPoint

Value : a NumericalSample, where the components [i] of each NumericalPoint have been added the corresponding value *translation*[i]

Details :

when two elements of the sample are equal, the rank of the first element appearing in the sample will be considered as the lower one (for computing Spearman correlation matrix)

©2007 EDF - EADS - PhiMeca

4.2 Distribution factory

4.2.1 DistributionImplementationFactory

Usage : *DistributionImplementationFactory*()

Arguments : none

Value : a DistributionImplementationFactory is the implementation of a Factory and is one of the following classes : BetaFactory, ExponentialFactory, GammaFactory, GeometricFactory, GumbelFactory, HistogramlFactory, LogisticFactory, LogNormalFactory, MultiNomialFactory, NormalFactory, PoissonFactory, StudentFactory, TriangularFactory, TruncatedNormalFactory, UserDefinedFactory, UniformFactory, WeibullFactory.

Some methods :

buildImplementation

Usage : *buildImplementation*(*sample*)

Arguments : *sample* : a NumericalSample, of dimension $n \geq 1$

Value : a DistributionImplementation, which is a Beta, Exponential, Gamma, Geometric, Gumbel, Histogram, Logistic, LogNormal, MultiNomial, Normal, Poisson, Student, Triangular, TruncatedNormal, UserDefined, Uniform, Weibull.

©2007 EDF - EADS - PhiMeca

4.3 Correlation analysis

4.3.1 CorrelationAnalysis

Usage : $CorrelationAnalysis()$

Arguments : none

Some methods :

PCC

Usage : $PCC(sample1, sample2)$

Arguments :

$sample1$: a NumericalSample, of dimension $n \geq 1$

$sample2$: a NumericalSample, of dimension $=1$

Value : a NumericalPoint, the PCC (partial Pearson Correlation Coefficient) coefficients evaluated between the $sample2$ and each coordinate of $sample1$

PRCC

Usage : $PRCC(sample1, sample2)$

Arguments :

$sample1$: a NumericalSample, of dimension $n \geq 1$

$sample2$: a NumericalSample, of dimension $=1$

Value : a NumericalPoint, the PRCC (Pearson Rank Correlation Coefficient) coefficients evaluated between the $sample2$ and each coordinate of $sample1$ (based on the rank values)

PearsonCorrelation

Usage : $PearsonCorrelation(sample1, sample2)$

Arguments :

$sample1$: a NumericalSample, of dimension $= 1$

$sample2$: a NumericalSample, of dimension $=1$

Value : a real value, the Pearson Correlation coefficient evaluated between the $sample2$ and $sample1$

SRC

Usage : $SRC(sample1, sample2)$

Arguments :

$sample1$: a NumericalSample, of dimension $n \geq 1$

$sample2$: a NumericalSample, of dimension $=1$

Value : a NumericalPoint, the SRC (Standard Regression Coefficient) coefficients evaluated between the $sample2$ and each coordinate of $sample1$

SRCC

Usage : $PRCC(sample1, sample2)$

Arguments :

$sample1$: a NumericalSample, of dimension $n \geq 1$

©2007 EDF - EADS - PhiMeca

*sample*2 : a NumericalSample, of dimension =1

Value : a NumericalPoint, the SRCC (Standard Rank Regression Coefficient)coefficients evaluated between the *sample*2 and each coordinate of *sample*1 (based on the rank values)

SpearmanCorrelation

Usage : *SpearmanCorrelation*(*sample*1, *sample*2)

Arguments :

*sample*1 : a NumericalSample, of dimension =1

*sample*2 : a NumericalSample, of dimension =1

Value : a real value, the Spearman Correlation coefficient evaluated between the *sample*2 and *sample*1 (based on the rank values)

©2007 EDF - EADS - PhiMeca

4.4 Fitting test

4.4.1 TestResult

Usage : a TestResult is the result of a fitting test, of type NormalityTest or HypothesisTest.

Some methods :

getBinaryQualityMeasure

Usage : *getBinaryQualityMeasure*()
Arguments : none
Value : a boolean value, indicating the succes of the test : 1 for succes and 0 for failure

getPValue

Usage : *getPValue*()
Arguments : none
Value : a real positive value, <1, the p-value of the test

getTestType

Usage : *getTestType*()
Arguments : none
Value : a string describing the type of the test

getThreshold

Usage : *getThreshold*()
Arguments : none
Value : a real positive value, < 1, the p-value threshold

©2007 EDF - EADS - PhiMeca

4.4.2 FittingTest

This class is used through its static methods in order to evaluate some hypothesis on samples : independence or monotonous relation.

Some methods :

BestModelBIC

Usage :

$FittingTest().BestModelBIC(sample, factoryCollection)$

$FittingTest().BestModelBIC(sample, distributionCollection)$

Arguments :

sample : a NumericalSample, the sample which will be tested

$factoryCollection$: a FactoryCollection, the collection of factories which are the structures which build the distribution from a sample

distributionCollection : a DistributionCollection, a collection of the distributions which will be tested through the BIC criteria

Value : a Distribution, the best one according to the BIC criteria

BestModelChiSquared

Usage :

$FittingTest().BestModelChiSquared(sample, factoryCollection)$

$FittingTest().BestModelChiSquared(sample, distributionCollection)$

Arguments :

sample : a NumericalSample, the sample which will be tested

$factoryCollection$: a FactoryCollection, a collection of the factories which are the structures which build distributions from a sample

distributionCollection : a DistributionCollection, the collection of the distributions which will be tested through the Chi Squared criteria

Value : a Distribution, the best one according to the ChiSquared criteria

BestModelKolmogorov

Usage :

$FittingTest().BestModelKolmogorov(sample, factoryCollection)$

$FittingTest().BestModelKolmogorov(sample, distributionCollection)$

Arguments :

sample : a NumericalSample, the sample which will be tested

$factoryCollection$: a FactoryCollection, a collection of the factories which are the structures which build distributions from a sample

distributionCollection : a DistributionCollection, the collection of the distributions which will be tested through the Kolmogorov criteria

Value : a Distribution, the best one according to the Kolmogorov criteria

BIC

©2007 EDF - EADS - PhiMeca

Usage :

$FittingTest().BIC(sample, factory)$

$FittingTest().BIC(sample, distribution)$

Arguments :

sample : a NumericalSample, the sample which will be tested

$factory$: a Factory, the structure which builds the distribution from the sample which will be tested

distribution : a Distribution, which will be tested through the BIC criteria

Value : a real value, the BIC value of the distribution tested evaluated on the sample

ChiSquared

Usage :

$FittingTest().ChiSquared(sample, factory, level)$

$FittingTest().ChiSquared(sample, distribution, level)$

Arguments :

sample : a NumericalSample, the sample which will be tested

$factory$: a Factory, the structure which builds the distribution from the sample which will be tested

distribution : a Distribution, which will be tested through the ChiSquared criteria

$level$: a real value, constraint : $0 < level < 1$, such as $1 - level$ be the first type error of the fitting test (the probability you reject the distribution tested whereas you should not have). If not fulfilled, by default, $level = 0.95$.

Value : a TestResult, the structure which contains the result of the ChiSquared Test : the first usage tests a type of distribution, the second one tests a particular distribution

Kolmogorov

Usage :

$FittingTest().Kolmogorov(sample, factory, level)$

$FittingTest().Kolmogorov(sample, distribution, level)$

Arguments :

sample : a NumericalSample, the sample which will be tested

$factory$: a Factory, the structure which builds the distribution from the sample which will be tested

distribution : a Distribution, which will be tested through the Kolmogorov criteria

$level$: a real value, constraint : $0 < level < 1$, such as $1 - level$ be the first type error of the fitting test (the probability you reject the distribution tested whereas you should not have). If not fulfilled, by default, $level = 0.95$.

Value : a TestResult, the structure which contains the result of the Kolmogorov Test : the first usage tests a type of distribution, the second one tests a particular distribution

©2007 EDF - EADS - PhiMeca

4.4.3 VisualTest

This class is used through its static methods in order to graphically evaluate some hypothesis on samples : independence or monotonous relation.

Some methods :

DrawClouds

Usage :

$VisualTest().DrawClouds(sample1, dist)$

$VisualTest().DrawClouds(sample1, sample2)$

Arguments :

$sample1$: a NumericalSample, drawn on the graph

$sample2$: a NumericalSample, drawn on the graph

$dist$: a Distribution, the distribution which pdf is drawn

Value : a Graph, the structure wich contains : one curve (pdf) and a cloud for the first usage or two clouds in the second usage

DrawEmpiricalCDF

Usage : $VisualTest().DrawEmpiricalCDF(sample, xMin, xMax)$

Arguments :

$sample$: a NumericalSample, drawn on the graph

$xMin$: a real value, the lower boundary of the graph

$xMax$: a real value, the upper boundary of the graph, must be $> xMin$

Value : a Graph, the structure wich contains : one staircase curve which is the empirical cdf of the sample

DrawHenryLine

Usage : $VisualTest().DrawHenryLine(sample)$

Arguments : $sample$: a NumericalSample, drawn on the graph

Value : a Graph, the structure wich contains one histogram

DrawHistogram

Usage :

$VisualTest().DrawHistogram(sample, barNumber)$

$VisualTest().DrawHistogram(sample)$

Arguments :

$sample$: a NumericalSample, which histogtram is drawn

$barNumber$: an integer, the number of barplots used to draw the histogram. If not mentionned, the number of barplots is automatically determined by Open TURNS according to the Silverman rule

Value : a Graph, the structure wich contains the graph : one curve and a cloud for the first usage or two clouds in the second usage

©2007 EDF - EADS - PhiMeca

DrawLMResidualTest

Usage : $VisualTest().DrawLMResidualTest(sample1, sample2, linearModel)$

Arguments :

$sample1$: a NumericalSample, X, of dimension 1

$sample2$: a NumericalSample, Y, of dimension 1, which is scalar described by a linear model from X : $Y = aX + b$, a and b real values

$linearModel$: a LinearModel, the regression model

Value : a Graph, the structure wich contains the cloud of the residual values, couples (residual i, residual i+1)

DrawLMVisualTest

Usage : $VisualTest().DrawLMVisualTest(sample1, sample2, linearModel)$

Arguments :

$sample1$: a NumericalSample, X, of dimension 1

$sample2$: a NumericalSample, Y, of dimension 1, which is described by a linear model from X : $Y = aX + b$, a and b real values

$linearModel$: a LinearModel, the regression model

Value : a Graph, the structure wich contains the cloud of points (X_i, Y_i) and the linear model line $Y = aX + b$

DrawQQplot

Usage :

$VisualTest().DrawQQplot(sample1, sample2)$

$VisualTest().DrawQQplot(sample1, sample2, pointNumber)$

$VisualTest().DrawQQplot(sample1, distribution)$

$VisualTest().DrawQQplot(sample1, distribution, pointNumber)$

Arguments :

$sample1$: a NumericalSample, used to build on the graph

$sample2$: a NumericalSample, used to build the graph

$distribution$: a Distribution, the distribution which pdf is drawn

$pointNumber$: an integer, the number of points used to build the graph, equal to 20 by default

Value : a Graph, the structure wich contains the corresponding empirical fractiles between the two samples in the two first usages, or between the sample and the distribution in the two last usages

©2007 EDF - EADS - PhiMeca

4.4.4 NormalityTest

This class is used through its static methods in order to evaluate whether the sample follows a normal distribution. These two tests give more importance to extreme values.

Some methods :

AndersonDarlingNormal

Usage :

$NormalityTest().AndersonDarlingNormal(sample)$

$NormalityTest().AndersonDarlingNormal(sample, level)$

Arguments :

sample : a NumericalSample, of dimension 1 : the sample tested

level : a positive real value, the threshold p-value of the test (= 1- first type risk), must be < 1, equal to 0.95 by default

Value : a TestResult, the structure which contains the result of the test.

Details : the AndersonDarlingNormal Test is used to check whether the sample follows a normal distribution. This test gives more importance to extreme values

CramerVonMisesNormal

Usage :

$NormalityTest().CramerVonMisesNormal(sample)$

$NormalityTest().CramerVonMisesNormal(sample, level)$

Arguments :

sample : a NumericalSample, of dimension 1 : the sample tested

level : a positive real value, the threshold p-value of the test (= 1- first type risk), must be < 1, equal to 0.95 by default

Value : a TestResult, the structure which contains the result of the test.

Details : the CramerVonMisesNormal Test is used to check whether the sample follows a normal distribution. This test gives more importance to extreme values

©2007 EDF - EADS - PhiMeca

4.4.5 HypothesisTest

This class is used through its static methods in order to evaluate some hypothesis on samples : independence or monotonous relation.

Some methods :

ChiSquared

Usage :

$HypothesisTest().ChiSquared(firstSample, secondSample)$

$HypothesisTest().ChiSquared(firstSample, secondSample, level)$

Arguments :

$firstSample$: a NumericalSample, of dimension 1 : the first sample tested

$secondSample$: a NumericalSample, of dimension 1 : the second sample tested

$level$: a positive real value, the threshold p-value of the test (= 1- first type risk), must be < 1, equal to 0.95 by default

Value : a TestResult, the structure which contains the result of the test

Details : the ChiSquared Test is used to check whether two discrete samples are independent

FullPearson

Usage :

$HypothesisTest().FullPearson(firstSample, secondSample)$

$HypothesisTest().FullPearson(firstSample, secondSample, level)$

Arguments :

$firstSample$: a NumericalSample, of dimension $n \geq 1$: the first sample tested

$secondSample$: a NumericalSample, of dimension 1 : the second sample tested

$level$: a positive real value, the threshold p-value of the test (= 1- first type risk), must be < 1, equal to 0.95 by default

Value : a TestResultCollection, the structure which contains the results of the successive tests.

Details : the FullPearson Test is the independence Pearson test between 2 samples : firstSample of dimension n and secondSample of dimension 1. If firstSample[i] is the numerical sample extracted from firstSample (ith coordinate of each point of the numerical sample), FullPearson performs the Independence Pearson test simultaneously on firstSample[i] and secondSample. For all i, it is supposed that the couple (firstSample[i] and secondSample) is issued from a gaussian vector.

FullRegression

Usage :

$HypothesisTest().FullRegression(firstSample, secondSample)$

$HypothesisTest().FullRegression(firstSample, secondSample, level)$

Arguments :

$firstSample$: a NumericalSample, of dimension $n \geq 1$: the first sample tested

$secondSample$: a NumericalSample, of dimension 1 : the second sample tested

$level$: a positive real value, the threshold p-value of the test (= 1- first type risk), must be < 1, equal to 0.95 by default.

©2007 EDF - EADS - PhiMeca

Value : a TestResultCollection, the structure which contains the results of the successive tests.

Details : the FullRegression Test is used to check the quality of the regression model between two samples : firstSample of dimension n and secondSample of dimension 1. If firstSample[i] is the numerical sample extracted from firstSample (ith coordinate of each point of the numerical sample), FullRegression performs the Regression test simultaneously on all firstSample[i] and secondSample. The Regression test tests if the regression model between two scalar numerical samples is significant. It is based on the deviation analysis of the regression. The Fisher distribution is used.

FullSpearman

Usage :

$HypothesisTest().FullSpearman(firstSample, secondSample)$

$HypothesisTest().FullSpearman(firstSample, secondSample, level)$

Arguments :

$firstSample$: a NumericalSample, of dimension $n \geq 1$: the first sample tested

$secondSample$: a NumericalSample, of dimension 1 : the second sample tested

$level$: a positive real value, the threshold p-value of the test (= 1- first type risk), must be < 1, equal to 0.95 by default.

Value : a TestResultCollection, the structure which contains the results of the successive tests.

Details : the FullSpearman Test is used to check the hypothesis of monotonous relation between samples : firstSample of dimension n and secondSample of dimension 1. If firstSample[i] is the numerical sample extracted from firstSample (ith coordinate of each point of the numerical sample), FullSpearman performs the Independence Spearman test simultaneously on all firstSample[i] and secondSample.

PartialPearson

Usage :

$HypothesisTest().PartialPearson(firstSample, secondSample, selection)$

$HypothesisTest().PartialPearson(firstSample, secondSample, selection, level)$

Arguments :

$firstSample$: a NumericalSample, of dimension $n \geq 1$: the first sample tested

$secondSample$: a NumericalSample, of dimension 1 : the second sample tested

$selection$: a Indices, array of integers selecting the indices of the coordinates of the first sample which will successively be tested with the second sample through the Pearson Test

$level$: a positive real value, the threshold p-value of the test (= 1- first type risk), must be < 1, equal to 0.95 by default.

Value : a TestResultCollection, the structure which contains the results of the successive tests.

Details : the PartialPearson Test is the independence Pearson test between 2 samples : firstSample of dimension n and secondSample of dimension 1. If firstSample[i] is the numerical sample extracted from firstSample (ith coordinate of each point of the numerical sample), PartialPearson performs the Independence Pearson test simultaneously on firstSample[i] and secondSample, for i in the selection. For all i, it is supposed that the couple (firstSample[i] and secondSample) is issued from a gaussian vector.

©2007 EDF - EADS - PhiMeca

PartialRegression

Usage :

$HypothesisTest().PartialRegression(firstSample, secondSample, selection)$

$HypothesisTest().PartialRegression(firstSample, secondSample, selection, level)$

Arguments :

$firstSample$: a NumericalSample, of dimension $n \geq 1$: the first sample tested

$secondSample$: a NumericalSample, of dimension 1 : the second sample tested

$selection$: a Indices, array of integers selecting the indices of the coordinates of the first sample which will successively be tested with the second sample through the Regression Test

$level$: a positive real value, the threshold p-value of the test (= 1- first type risk), must be < 1, equal to 0.95 by default.

Value : a TestResult, the structure which contains the result of the test.

Details : the PartialRegression Test is used to check the quality of the regression model between two samples : firstSample of dimension n and secondSample of dimension 1. If firstSample[i] is the numerical sample extracted from firstSample (ith coordinate of each point of the numerical sample), PartialRegression performs the Regression test simultaneously on all firstSample[i] and secondSample, for i in the selection. The Regression test tests ifthe regression model between two scalar numerical samples is significant. It is based on the deviation analysis of the regression. The Fisher distribution is used.

PartialSpearman

Usage :

$HypothesisTest().PartialSpearman(firstSample, secondSample, selection)$

$HypothesisTest().PartialSpearman(firstSample, secondSample, selection, level)$

Arguments :

$firstSample$: a NumericalSample, of dimension $n \geq 1$: the first sample tested

$secondSample$: a NumericalSample, of dimension 1 : the second sample tested

$selection$: a Indices, array of integers selecting the indices of the coordinates of the first sample which will successively be tested with the second sample through the Spearman Test

$level$: a positive real value, the threshold p-value of the test (= 1- first type risk), must be < 1, equal to 0.95 by default.

Value : a TestResultCollection, the structure which contains the results of the successive tests.

Details : the PartialSpearman Test is used to check the hypothesis of monotonous relation between samples : firstSample of dimension n and secondSample of dimension 1. If firstSample[i] is the numerical sample extracted from firstSample (ith coordinate of each point of the numerical sample), PartialSpearman performs the Independence Spearman test simultaneously on firstSample[i] and secondSample, for i in the selection.

Pearson

Usage :

$HypothesisTest().Pearson(firstSample, secondSample)$

$HypothesisTest().Pearson(firstSample, secondSample, level)$

Arguments :

$firstSample$: a NumericalSample, of dimension 1 : the first sample tested

©2007 EDF - EADS - PhiMeca

secondSample : a NumericalSample, of dimension 1 : the second sample tested

level : a positive real value, the threshold p-value of the test (= 1- first type risk), must be < 1, equal to 0.95 by default.

Value : a TestResult, the structure which contains the result of the test.

Details : the Test is used to check whether two samples which form a gaussian vector are independent (based on the evaluation of the linear correlation coefficient).

Smirnov

Usage :

$HypothesisTest().Smirnov(firstSample, secondSample)$

$HypothesisTest().Smirnov(firstSample, secondSample, level)$

Arguments :

$firstSample$: a NumericalSample, of dimension 1 : the first sample tested

secondSample : a NumericalSample, of dimension 1 : the second sample tested

level : a positive real value, the threshold p-value of the test (= 1- first type risk), must be < 1, equal to 0.95 by default.

Value : a TestResult, the structure which contains the result of the test.

Details : the Smirnov Test is used to check whether two continuous scalar samples (of sizes not necessarily equal) follow the same distribution.

Spearman

Usage :

$HypothesisTest().Spearman(firstSample, secondSample)$

$HypothesisTest().Spearman(firstSample, secondSample, level)$

Arguments :

$firstSample$: a NumericalSample, of dimension 1 : the first sample tested

secondSample : a NumericalSample, of dimension 1 : the second sample tested

level : a positive real value, the threshold p-value of the test (= 1- first type risk), must be< 1, equal to 0.95 by default.

Value : a TestResult, the structure which contains the result of the test.

Details : the Spearman Test is used to check whether two scalar samples have a monotonous relation.

©2007 EDF - EADS - PhiMeca

4.5 Linear model

4.5.1 LinearModelFactory

This class is used in order to create a linear model from numerical samples.

Usage : $LinearModelFactory()$

Arguments : none

Some methods :

buildLM

Usage :

$buildLM(sampleX, sampleY)$

$buildLM(sampleX, sampleY, level)$

Arguments :

$sampleX$: a NumericalSample, of dimension $n \geq 1$

$sampleY$: a NumericalSample, of dimension 1

$level$: the level value of the confidence intervals of each coefficient of the linear model, equal to 0.95 by default

Value : a LinearModel, the linear model built from the samples $(sampleX, sampleY)$: $Y = a_0 + \sum_i a_i X_i + \varepsilon$, where ε is the aleatory residual with zero mean.

©2007 EDF - EADS - PhiMeca

4.5.2 LinearModel

Usage : A LinearModel is created through the method *buildLM* of a LinearModelFactory.

Some methods :

getConfidenceIntervals

Usage : *getConfidenceIntervals*()

Arguments : none

Value : a ConfidenceIntervalPersistentCollection, the collection of the confidence intervals of the linear model coefficients, corresponding to the level precised when the LinearModel class has been created through the method *buildLM*

getPValues

Usage : *getPValues*()

Arguments : none

Value : a NumericalScalarPersistentCollection, the collection of the p-values of the linear model coefficients

getPredict

Usage : *getPredict*(*sampleX*)

Arguments : *sampleX* : a NumericalSample, the sample we want to evaluate the response Y on

Value : a NumericalSample, of dimension 1, the response Y evaluated through the linear model on the sample *sampleX*

getRegression

Usage : *getRegression*()

Arguments : none

Value : a NumericalPoint, the coefficients of the linear model : $(a_0, a_1, \cdots, a_n)$

getResidual

Usage : *getResidual*(*sampleX*, *sampleY*)

Arguments :

sampleX : a NumericalSample, the *sampleX* on which the linear model has been built

sampleY : a NumericalSample, the *sampleY* on which the linear model has been built

Value : a NumericalPoint, the residuals

©2007 EDF - EADS - PhiMeca

5 Threshold exceedance probability evaluation with reliability algorithms

5.1 Optimisation

5.1.1 NearestPointAlgorithm

Usage :

NearestPointAlgorithm(*levelFunction*)

NearestPointAlgorithm(*nearestPointAlgorithmImplementation*)

Arguments :

levelFunction : a NumericalMathFunction, the constraint function of the constrained optimisation problem

nearestPointAlgorithmImplementation : a NearestPointAlgorithmImplementation, the implementatin of the nearest point algorithm, which is *Cobyla* , *AbdoRackwitz* or *SQP*.

Some methods :

getLevelFunction

Usage : *getLevelFunction*()

Arguments : none

Value : a NumericalMathFunction, the constraint function of the constrained optimisation problem

getLevelValue

Usage : *getLevelFunction*()

Arguments : none

Value : a real value, the level value of the constraint function in the constrained optimisation problem

getMaximumAbsoluteError

Usage : *getMaximumAbsoluteError*()

Arguments : none

Value : a positive real value, the maximum absolute error : maximum distance between two successive iterates

getMaximumConstraintError

Usage : *getMaximumConstraintError*()

Arguments : none

Value : a positive real value, the maximum absolute value of the constraint function minus the level value

getMaximumIterationsNumber

Usage : *getMaximumIterationsNumber*()

Arguments : none

©2007 EDF - EADS - PhiMeca

Value : an integer, the maximum number of iterations of the algorithm

getMaximumRelativeError

Usage : *geMaximumtRelativeError*()

Arguments : none

Value : a real value, the maximum relative distance between two successive iterates (with regards the second iterate)

getMaximumResidualError

Usage : *getMaximumResidualError*()

Arguments : none

Value : a real value, the maximum orthogonality error (lack of orthogonality between the vector Center - Iterate and the constraint surface)

getResult

Usage : *getResult*()

Arguments : none

Value : a NearestPointAlgorithmImplementationResult, the structure containing all the results of the constrained optimisation problem

getStartingPoint

Usage : *getStartingPoint*()

Arguments : none

Value : a NumericalPoint, the starting point of the constrained optimisation research

run

Usage : *run*()

Arguments : none

Value : none

Role : it creates a NearestPointAlgorithmImplementationResult, the optimisation result wich is accesible with the method getResult().

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

5.1.2 Cobyla

Usage :

$Cobyla()$

$Cobyla(specificParameters, levelFunction)$

Arguments :

specificParameters : a CobylaSpecificParameters, the list of the parameters specific to the Cobyla algorithm

levelFunction : a NumericalMathFunction, the constraint function of the constrained optimisation problem

Details : When no argument is specified, the parameters will have to be fulfilled after, for example, when used in a FORM algorithm (see the corresponding Use Case).

Some methods :

getSpecificParameters

Usage : $getSpecificParameters()$

Arguments : none

Value : a CobylaSpecificParameters, the list of the parameters specific to the Cobyla algorithm

This *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

5.1.3 CobylaSpecificParameters

Usage :

CobylaSpecificParameters()

CobylaSpecificParameters(*rhoBeg*)

Arguments : *rhoBeg* : a real positive strictly value, a reasonable initial step to the variables. When not fulfilled, by default equal to 0.1

Some methods :

getRhoBeg

Usage : *getRhoBeg*()

Arguments : none

Value : a real value >0, a reasonable initial step to the variables

This *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

5.1.4 AbdoRackwitz

Usage :

AbdoRackwitz()

AbdoRackwitz(*specificParameters*, *levelFunction*)

Arguments :

specificParameters : a AbdoRackwitzSpecificParameters, the list of the parameters specific to the AbdoRackwitz algorithm

levelFunction : a NumericalMathFunction, the constraint function of the constrained optimisation problem

Details : When no argument is specified, the parameters will have to be fulfilled after, for example, when used in a FORM algorithm (see the corresponding Use Case).

The AbdoRackwitz algorithm is a gradient-based constrained optimization method, thus the NumericalMathFunction has to provide its gradient:

- If the function is a meta-model generated by OpenTURNS, the analytical gradient is automatically provided.
- If it is an analytical function, OpenTURNS provides a gradient based on the centered finite difference method, that the user can change to better fit ots will by another GradientImplementation (e.g. constructed by another finite difference method).
- If the function is loaded thanks to a wrapper and if the wrapper provide an implementation of the gradient, OpenTURNS uses it. If there is no gradient provided, OpenTURNS provides a gradient based on centered finite difference method, but with a parameterization different from the case of analytical functions (assuming a lower precision for the function evaluation than in the analytical case).

Be aware of the potential pitfalls associated with the use of finite differences and check the value of the finite difference epsilon for each dimension if AbdoRackwitz algorithm fails to converge.

Some methods :

getSpecificParameters

Usage : *getSpecificParameters*()

Arguments : none

Value : a AbdoRackwitzSpecificParameters, list of the parameters specific to the AbdoRackwitz algorithm

This *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

5.1.5 AbdoRackwitzSpecificParameters

Usage :

$AbdoRackwitzSpecificParameters()$

$AbdoRackwitzSpecificParameters(Omega, Smooth, Tau)$

Arguments :

$Omega$: a real strictly positive value, the Armijo factor. It must be < 1 and should be rather small. When not fulfilled, by default equal to 10^{-4}

$Smooth$: a real value > 1, the increasing rate of the penalisation coefficient for the line search. It must be > 1 and should be rather near 1. When not fulfilled, by default equal to 1.2

Tau : a real positive value, the multiplicative decrease of the linear step. It must be < 1. When not fulfilled, by default equal to 0.5

Some methods :

getOmega

Usage : $getOmega()$

Arguments : none

Value : a real value between 0 and 1, the Armijo factor

getSmooth

Usage : $getSmooth()$

Arguments : none

Value : a real value > 1, the increasing rate of the penalisation coefficient for the line search

getTau

Usage : $getTau()$

Arguments : none

Value : a real value between 0 and 1, the multiplicative decrease of the linear step

This $getMethod$ is associated to a $setMethod$.

©2007 EDF - EADS - PhiMeca

5.1.6 SQP

Usage : $SQP(specificParameters, levelFunction)$

Arguments :

$specificParameters$: a SQPSpecificParameters, list of the parameters specific to the SQP algorithm

$levelFunction$: a NumericalMathFunction, the constraint function of the constrained optimisation problem

Some methods :

$getSpecificParameters$

Usage : $getSpecificParameters()$

Arguments : none

Value : a SQPSpecificParameters, list of the parameters specific to the SQP algorithm

This $getMethod$ is associated to a $setMethod$.

©2007 EDF - EADS - PhiMeca

5.1.7 SQPSpecificParameters

Usage :

$SQPSpecificParameters()$

$SQPSpecificParameters(Omega, Smooth, Tau)$

Arguments :

$Omega$: a real value by default equal to 10^{-4}, Armijo factor, must be > 0 and <1 but rather small

$Smooth$: a real value by default equal to 1.2, the increasing rate of the penalisation coefficient for the line search, must be >1 but rather near 1

Tau : a real value by default equal to 0.5, multiplicative decrease of the linear step, must be > 0 and <1

Some methods :

getOmega

Usage : $getOmega()$
Arguments : none
Value : a real value between 0 and 1, Armijo factor

getSmooth

Usage : $getSmooth()$
Arguments : none
Value : a real value > 1, the increasing rate of the penalisation coefficient for the line search

getTau

Usage : $getTau()$
Arguments : none
Value : a real value between 0 and 1, multiplicative decrease of the linear step

This $getMethod$ is associated to a $setMethod$.

©2007 EDF - EADS - PhiMeca

5.1.8 NearestPointAlgorithmImplementationResult

Usage : structure created by the method run() of a NearestPointAlgorithm and obtained thanks to the method *getResult*()

Some methods :

getAbsoluteError

Usage : *getAbsoluteError*()

Arguments : none

Value : a NumericalScalar, the absolute error : the distance between the two last successif iterates when the algorithm stops

getConstraintError

Usage : *getConstraintError*()

Arguments : none

Value : a real value, the absolute value of the constraint function minus the level value at the last iterate point when the algorithm stops

getIterationsNumber

Usage : *getIterationsNumber*()

Arguments : none

Value : an integer, the number of performed iterations of the algorithm when the algorithm stops

getMinimizer

Usage : *getMinimizer*()

Arguments : none

Value : a NumericalPoint, the last iterate when the algorithm stops (the solution of the optimisation problem)

getRelativeError

Usage : *getRelativeError*()

Arguments : none

Value : a real value, the relative distance between the two last successif iterates (with regards the last iterate)

getResidualError

Usage : *getResidualError*()

Arguments : none

Value : a real value, the orthogonality error of the solution point (lack of orthogonality between the vector Center - Last Iterate and the constraint surface)

©2007 EDF - EADS - PhiMeca

5.2 Reliability Algorithms

5.2.1 Analytical

Usage : *Analytical*(*nearestPointAlgorithm*, *event*, *physicalStartingPoint*)

Arguments :

nearestPointAlgorithm : a NearestPointAlgorithm, the optimisation algorithm which will be used to research the design point

event : a Event, the event we want to evaluate the probability

physicalStartingPoint : a NumericalPoint, the starting point of the optimisation research, declared in the physical space

Some methods :

getAnalyticalResult

Usage : *getAnalyticalResult*()

Arguments : none

Value : a AlgorithmAnalyticalResult, the result structure which contains results

getEvent

Usage : *getEvent*()

Arguments : none

Value : a Event, the event we want to evaluate the probability

getNearestPointAlgorithm

Usage : *getNearestPointAlgorithm*()

Arguments : none

Value : a NearestPointAlgorithm, the optimisation algorithm which will be used to research the design point

getPhysicalStartingPoint

Usage : *getPhysicalStartingPoint*()

Arguments : none

Value : a NumericalPoint, the starting point of the optimisation research, declared in the physical space

run

Usage : *run*()

Arguments : none

Value : it performs the research design point and creates a AnalyticalResult, the structure result wich is accesible with the method getAnalyticalResult().

The methods *getEvent*, *getNearestPointAlgorithm* and *getPhysicalStartingPoint* are associated to a *setMethod*.

Derivative Classes : FORM ans SORM

©2007 EDF - EADS - PhiMeca

5.2.2 AnalyticalResult

Usage : structure created by the method run() of a Analytical and obtained thanks to the method *getAnalyticalResult*()

Some methods :

drawHasoferReliabilityIndexSensitivity

Usage : *drawHasoferReliabilityIndexSensitivity*()

Arguments : none

Value : a GraphCollection (the collection of two barplots) drawing the sensitivity of the Hasofer Reliability Index to the parameters of the marginals of the probabilistic input vector (first graph) and to the parameters of the dependence structure of the probabilistic input vector (second graph).

drawImportanceFactors

Usage : *drawImportanceFactors*()

Arguments : none

Value : a Graph, the pie of the importance factors of the probabilistic variables

getHasoferReliabilityIndex

Usage : *getHasoferReliabilityIndex*()

Arguments : none

Value : a real positive value, the Hasofer Reliability Index

getHasoferReliabilityIndexSensitivity

Usage : *getHasoferReliabilityIndexSensitivity*()

Arguments : none

Value : a Sensitivity, the sensitivities of the Hasofer Reliability Index to the parameters of the probabilistic input vector (marginals and dependence structure)

getImportanceFactors

Usage : *getImportanceFactors*()

Arguments : none

Value : a NumericalPoint, the importance factors of probabilistic variables

getIsStandardPointOriginInFailureSpace

Usage : *getIsStandardPointOriginInFailureSpace*()

Arguments : none

Value : a boolean which indicates whether the origine of the standard space is in the failure space

getLimitStateVariable

Usage : *getLimitStateVariable*()

Arguments : none

©2007 EDF - EADS - PhiMeca

Value : a Event, the event we evaluated the probability

getPhysicalSpaceDesignPoint

Usage : *getPhysicalSpaceDesignPoint*()

Arguments : none

Value : a NumericalPoint, the starting point of the optimisation research, declared in the physical space

getStandardSpaceDesignPoint

Usage : *getStandardSpaceDesignPoint*()

Arguments : none

Value : a NumericalPoint, the starting point of the optimisation research, declared in the standard space

Derivative Classes : FORMResult and SORMResult

©2007 EDF - EADS - PhiMeca

5.2.3 Event

Usage :

Event(*antecedent*, *comparisonOperator*, *threshold*)

Event(*antecedent*, *comparisonOperator*, *threshold*, *name*)

Arguments :

antecedent : a RandomVector, of dimension 1 : the output variable of interest

comparisonOperator : a ComparisonOperator, the comparison operator which is equal to *Less*, *Greater*, *LessOrEqual* or *GreaterOrEqual*

threshold : a real value, the threshold we want to compare to *antecedent*

name : a string, the name of the event

Some methods :

getDimension

Usage : *getDimension*()

Arguments : none

Value : an integer, the dimension of the probabilistic input vector

getNumericalSample

Usage : *getNumericalSample*(*size*)

Arguments : *size* : an integer, the size of the numerical sample generated

Value : a NumericalSample filled with boolean values (1 for the realisation of the event and 0 else) : *size* realisations of the event (considered as a Bernoulli variable)

getOperator

Usage : *getOperator*()

Arguments : none

Value : a ComparisonOperator, the comparison operator of the event

getRealization

Usage : *getRealization*()

Arguments : none

Value : a NumericalPoint of dimension 1, filled with a boolean value (1 for the realisation of the event and 0 else) : one realisation of the event (considered as a Bernoulli variable)

getThreshold

Usage : *getThreshold*()

Arguments : none

Value : a real value, the threshold of the event

Derivative Class : StandardEvent

©2007 EDF - EADS - PhiMeca

5.2.4 StandardEvent

This class inherits from Event.

Usage :

$StandardEvent(antecedent, comparisonOperator, threshold)$

$StandardEvent(antecedent, comparisonOperator, threshold, name)$

$StandardEvent(event)$

Arguments :

antecedent : a RandomVector, of dimension 1 : the output variable of interest

comparisonOperator : a ComparisonOperator, the comparison operator which is equal to $Less$, $Greater$, $LessOrEqual$ or $GreaterOrEqual$

threshold : a real value, the threshold we want to compare to *antecedent*

name : a string, the name of the event

event : a Event, the physical event associated to the standard event

©2007 EDF - EADS - PhiMeca

5.2.5 FORM

This class inherits from Analytical.

Usage : $FORM(nearestPointAlgorithm, event, physicalStartingPoint)$

Arguments :

nearestPointAlgorithm : a NearestPointAlgorithm, the optimisation algorithm which will be used to research the design point

event : a Event, the event in the physical space we want to evaluate the probability

physicalStartingPoint : a NumericalPoint, the starting point of the optimisation research, declared in the physical space

Some methods :

getResult

Usage : $getResult()$

Arguments : none

Value : a FORMResult, structure containing all the results of the FORM analysis

run

Usage : $run()$

Arguments : none

Value : it creates a FORMResult, the optimisation result wich is accesible with the method getResult().

©2007 EDF - EADS - PhiMeca

5.2.6 FORMResult

This class inherits from AnalyticalResult.

Usage : structure created by the method run() of a FORM and obtained thanks to the method *getResult*()

Some methods :

drawEventProbabilitySensitivity

Usage : *drawEventProbabilitySensitivity*()

Arguments : none

Value : a GraphCollection (the collection of two barplots) drawing the sensitivities of the FORM Probability with regards to the parameters of the probabilistic input vector (first graph) and to parameters of the dependence structure of the probabilistic input vector (second graph)

getEventProbability

Usage : *getEventProbability*()

Arguments : none

Value : a positive real value, the FORM probability of the event

getEventProbabilitySensitivity

Usage : *getEventProbabilitySensitivity*()

Arguments : none

Value : a Sensitivity, the sentivities of the FORM Probability with regards to the parameters of the probabilistic input vector and to parameters of the dependence structure of the probabilistic input vector

getGeneralisedReliabilityIndex

Usage : *getGeneralisedReliabilityIndexHohenBichler*()

Arguments : none

Value : a real value, the generalised reliability index evaluated from the FORM Probability. The generalised reliability index from the FORM probability is equal to $\pm$ the Hasofer reliability index according to the fact the standard space center fulfills the event or not

K

©2007 EDF - EADS - PhiMeca

5.2.7 SORM

This class inherits from Analytical.

Usage : $SORM(nearestPointAlgorithm, event, physicalStartingPoint)$

Arguments :

nearestPointAlgorithm : a NearestPointAlgorithm, the optimisation algorithm which will be used to research the design point

event : a Event, the event in the physical space we want to evaluate the probability

physicalStartingPoint : a NumericalPoint, the starting point of the optimisation research, declared in the physical space

Some methods :

getResult

Usage : $getResult()$

Arguments : none

Value : a SORMResult, structure containing all the results of the SORM analysis

run

Usage : $run()$

Arguments : none

Value : it creates a SORMResult, the optimisation result wich is accesible with the method getResult().

©2007 EDF - EADS - PhiMeca

5.2.8 SORMResult

This class inherits from AnalyticalResult.

Usage : structure created by the method run() of a SORM and obtained thanks to the method *getResult*()

Some methods :

getEventProbabilityBreitung

Usage : *getEventProbabilityBreitung*()
Arguments : none
Value : a positive real value, the SORM Probability according to the Breitung approximation

getEventProbabilityHohenBichler

Usage : *getEventProbabilityHohenBichler*()
Arguments : none
Value : a positive real value, the SORM Probability according to the Hohen Bichler approximation

getEventProbabilityTvedt

Usage : *getEventProbabilityTvedt*()
Arguments : none
Value : a positive real value, the SORM Probability according to the Tvedt approximation

getGeneralisedReliabilityIndexBreitung

Usage : *getGeneralisedReliabilityIndexBreitung*()
Arguments : none
Value : a real value, the generalised reliability index evaluated from the Breitung SORM Probability (positive or negative according to the fact the standard space center fulfills the event or not)

getGeneralisedReliabilityIndexHohenBichler

Usage : *getGeneralisedReliabilityIndexHohenBichler*()
Arguments : none
Value : a real value, the generalised reliability index evaluated from the Hohen Bichler SORM Probability (positive or negative according to the fact the standard space center fulfills the event or not)

getGeneralisedReliabilityIndexTvedt

Usage : *getGeneralisedReliabilityIndexTvedt*()
Arguments : none
Value : a real value, the generalised reliability index evaluated from the Tvedt SORM Probability (positive or negative according to the fact the standard space center fulfills the event or not)

©2007 EDF - EADS - PhiMeca

5.3 The Strong Max Test

5.3.1 StrongMaxTest

Usage :

$StrongMaximumTest(event, standardSpaceDesignPoint, importanceLevel, \cdots$
$accuracyLevel, confidenceLevel)$
$StrongMaximumTest(event, standardSpaceDesignPoint, importanceLevel, \cdots$
$accuracyLevel, pointNumber)$

Arguments :

event : a StandardEvent,

standardSpaceDesignPoint : a NumericalPoint,

importanceLevel : a positive real value, the importance level ε.

accuracyLevel : a positive real value, the accuracy Level τ. It is recommanded to take $\tau \leq 4$.

confidenceLevel : a positive real value, the confidenceLevel $(1 - q)$, must be < 1.

pointNumber : the number of points used to perform the Strong Maximum Test on which the limit state function is evaluated

Some methods :

getAccuracyLevel

Usage : *getAccuracyLevel*()
Arguments : none
Value : a positive real value, the accuracy Level τ

getConfidenceLevel

Usage : *getConfidenceLevel*()
Arguments : none
Value : a positive real value, the confidenceLevel $(1 - q)$

getEvent

Usage : *getEvent*()
Arguments : none
Value : a StandardEvent, the event in the standard space on which is based te Strong Maximum Test

getFarDesignPointVerifyingEventPoints

Usage : *getFarDesignPointVerifyingEventPoints*()
Arguments : none
Value : a NumericalSample, the list of points of the discretised sphere which are out of the vicinity of the standard design point and which verify the event

©2007 EDF - EADS - PhiMeca

getFarDesignPointVerifyingEventValues

Usage : *getFarDesignPointVerifyingEventValues*()

Arguments : none

Value : a NumericalSample, the list of the values of the limit state function on the points of the discretised sphere which are out of the vicinity of the standard design point and which verify the event

getFarDesignPointViolatingEventPoints

Usage : *getFarDesignPointViolatingEventPoints*()

Arguments : none

Value : a NumericalSample, the list of points of the discretised sphere which are out of the vicinity of the standard design point and which don't verify the event

getFarDesignPointViolatingEventValues

Usage : *getFarDesignPointViolatingEventValues*()

Arguments : none

Value : a NumericalSample, the list of the values of the limit state function on the points of the discretised sphere which are out of the vicinity of the standard design point and which don't verify the event

getImportanceLevel

Usage : *getImportanceLevel*()

Arguments : none

Value : a positive real value, the importance level ε

getNearDesignPointVerifyingEventPoints

Usage : *getNearDesignPointVerifyingEventPoints*()

Arguments : none

Value : a NumericalSample, the list of points of the discretised sphere which are inside the vicinity of the standard design point and which verify the event

getNearDesignPointVerifyingEventValues

Usage : *getNearDesignPointVerifyingEventValues*()

Arguments : none

Value : a NumericalSample, the list of the values of the limit state function on the points of the discretised sphere which are inside the vicinity of the standard design point and which verify the event

getNearDesignPointViolatingEventPoints

Usage : *getNearDesignPointViolatingEventPoints*()

Arguments : none

Value : a NumericalSample, the list of points of the discretised sphere which are out of the vicinity of the standard design point and which don't verify the event

©2007 EDF - EADS - PhiMeca

getNearDesignPointViolatingEventValues

Usage : *getNearDesignPointViolatingEventValues*()

Arguments : none

Value :

a NumericalSample, the list of the values of the limit state function on the points of the discretised sphere which are inside the vicinity of the standard design point and which don't verify the event

getPointNumber

Usage : *getPointNumber*()

Arguments : none

Value : an integre, the number of points used to perform the Strong Maximum Test, evaluated by the limit state function

getStandardSpaceDesignPoint

Usage : *getStandardSpaceDesignPoint*()

Arguments : none

Value : a NumericalPoint, the standard space design point

run

Usage : *run*()

Arguments : none

Value : it performs the Strong Maximum Test.

©2007 EDF - EADS - PhiMeca

6 Threshold exceedance probability evaluation with simulation

©2007 EDF - EADS - PhiMeca

6.1 HistoryStrategy

Four storgae strategies are proposed by Open TURNS, in order to store the numerical sample (in and out) used to evaluate the probability estimator, and to store the values of the estimator and its standard deviation used to draw the convergence graph.

Usage :

$HistoryStrategy(Null())$

$HistoryStrategy(Full())$

$HistoryStrategy(Compact(Ncompact))$

$HistoryStrategy(Last(Nlast))$

Arguments :

$Ncompact$: an integre, the number of points stored is in $[Ncompact, 2 * Ncompact]$. If the sample size is smaller than $Ncompact$, the entire sample is stored.

$Nlast$: an integer, the number of points stored. If the sample size is smaller than $Nlast$, the entire sample is stored.

Details :

$HistoryStrategy(Null())$: nothing is stored.

$HistoryStrategy(Full())$: each point is stored. Be careful! The memory will be exhausted for huge samples.

$HistoryStrategy(Compact(Ncompact))$: a regularly spaced sub-sample is stored, with a maximum size if the sample is large enough.

$HistoryStrategy(Last(Nlast))$: only the $Nlast$ last points are stored.

Some methods :

getSample

Usage : $getSample()$

Arguments : none

Value : a NumericalSample containing the object stored (the in and out numerical sample or the values of the probability estimator and its standard deviation)

6.2 RandomGenerator

Usage : $RandomGenerator()$

Arguments : none

Some methods :

Generate

Usage :

$Generate()$

$Generate(size)$

©2007 EDF - EADS - PhiMeca

Arguments : *size* : an integer, the number of realisations required. When not fulfilled, by default equal to 1

Value : a NumericalPoint, the list of the required realisations of a uniform distribution on [0,1]

GetState

Usage : *GetState*()

Arguments : : none

Value : a RandomGeneratorState, the state of the random generator

SetSeed

Usage : *SetSeed*(n)

Arguments : n : an integer which enables an easy initialisation of the random generator

Value : none. It initialises the state of the random generator

SetState

Usage : *SetState*(*state*)

Arguments : *state* : a RandomGeneratorState, the state of the random generator

Value : none. It initialises the state of the random generator

The *GetState* method is associated to a *SetState* one.

©2007 EDF - EADS - PhiMeca

6.3 Wilks

This class is a static class which enables the evaluation of the Wilks number : the minimal sample size $N_{\alpha,\beta,i}$ to perform in order to garantee that the empirical quantile α, noted $\tilde{q}_\alpha(N_{\alpha,\beta,i})$ evaluated with the $(n-i)th$ maximum of the sample, noted X_{N-i} be greater than the theoretical quantile q_α with a probability at least β :

$$Prob(\tilde{q}_\alpha(N_{\alpha,\beta,i}) = X_{n-i} > q_\alpha) > \beta$$

This class proposes one method :

ComputeSampleSize

Usage : $ComputeSampleSize(alpha, beta, i)$

Arguments :

alpha : a real value, order of the quantile we want the evaluate, must be > 1 and > 0

beta : confidence on the evaluation of the empirical quantile, must be > 1 and > 0

i : rank of the maximum which will evaluate the empirical quantile, by default $i = 0$ (maximum of the sample)

Value : an integer, the Wilks number.

©2007 EDF - EADS - PhiMeca

6.4 Simulation

Usage : *NearestPointAlgorithm*(*levelFunction*)

Arguments : *levelFunction* : a NumericalMathFunction, the constraint function of the constrained optimisation problem

Some methods :

getBlockSize

Usage : *getBlockSize*()

Arguments : none

Value : an integer, the number of terms in the probability simulation estimator grouped together

Details : for Monte Carlo, LHS and Importance Sampling methods, we recommend to use BlockSize = number of available CPU ; for the Directional Sampling, we recommend to use BlockSize = 1

getConvergenceStrategy

Usage : *getConvergenceStrategy*()

Arguments : none

Value : a HistoryStrategy, the storage strategy used to store the values of the probability estimator and its variance during the simulation algorithm

getEvent

Usage : *getEvent*()

Arguments : none

Value : an Event, which we want to evaluate the probability

getInputStrategy

Usage : *getInputStrategy*()

Arguments : none

Value : a HistoryStrategy, the storage strategy used to store the input random vector sample and the output random vector sample used to evaluate the probability estimator of the event probabilty

getMaximumCoefficientOfVariation

Usage : *getMaximumCoefficientOfVariation*()

Arguments : none

Value : a real value, the maximum coefficient of variation of the simulated sample

getMaximumOuterSampling

Usage : *getMaximumOuterSampling*()

Arguments : none

Value : an integer, the maximum number of groups of terms in the probability simulation estimator

Details : for Monte Carlo, LHS and Importance Sampling methods, the maximum number of evaluations of the limit state function defining the event is : MaximumOuterSampling * BlockSize

©2007 EDF - EADS - PhiMeca

getOutputStrategy

Usage : *getOutputStrategy*()

Arguments : none

Value : a HistoryStrategy, the storage strategy used to store the input random vector sample and the output random vector sample used to evaluate the probability estimator of the event probabilty

getResult

Usage : *getResult*()

Arguments : none

Value : a SimulationResult, the structure containing all the results obtained after simulation and created by the method run()

run

Usage : *run*()

Arguments : none

Value : it launches the simulation and creates a SimulationResult, structure containing all the results obtained after simulation

setConvergenceStrategy

Usage : *setConvergenceStrategy*(*myHistoryStrategy*)

Arguments : *myHistoryStrategy* : a HistoryStrategy, the storage strategy used to store the values of the probability estimator and its variance during the simulation algorithm.

Value : none

setInputOutputStrategy

Usage : *setInputOutputStrategy*(*myHistoryStrategy*)

Arguments : *myHistoryStrategy* : a HistoryStrategy, the storage strategy used to store the input random vector sample and the output random vector sample used to evaluate the probability estimator of the event probabilty

Value : none

Only the $getBlockSize$, $getMaximumCoefficientOfVariation$, $getMaximumOuterSampling$ methods have a $setMethod$ associated.

Derivative Classes :

MonteCarlo

LHS

DirectionnalSampling

ImportanceSampmling

©2007 EDF - EADS - PhiMeca

6.5 MonteCarlo

This class inherits from Simulation.

Usage : $MonteCarlo(event)$

Arguments : *event* : an Event, the event we want to evaluate the probability

©2007 EDF - EADS - PhiMeca

6.6 LHS

This class inherits from Simulation.

Usage : $LHS(event)$

Arguments : *event* : an Event, the event we want to evaluate the probability

©2007 EDF - EADS - PhiMeca

6.7 DirectionalSampling

6.7.1 DirectionalSampling

This class inherits from Simulation.

The Directional Sampling simulation operates in the standard space.

Usage :

DirectionalSampling(*event*)

DirectionalSampling(*event*, *rootStrategy*, *samplingStrategy*)

Arguments :

event : an Event, the event we want to evaluate the probability

rootStrategy : a RootStrategy, the strategy adopted to evaluate the intersections of each direction with the limit state function and take into account the contribution of the direction to the event probability. By default, *rootStrategy* = *RootStrategy*(*SafeAndSlow*)

samplingStrategy : a SamplingStrategy, the strategy adopted to sample directions. By default, *samplingStrategy* *SamplingStrategy*(*RandomDirection*).

Some methods :

getRootStrategy

Usage : *getRootStrategy*()
Arguments : none
Value : a RootStrategy, the root strategy adopted

getSamplingStrategy

Usage : *getSamplingStrategy*()
Arguments : none
Value : a SamplingStrategy, the direction sampling strategy adopted

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

6.7.2 RootStrategy

Usage :

RootStrategy()

RootStrategy(*rootStrategyImplementation*)

Arguments :

rootStrategyImplementation : a RootStrategyImplementation, the implementation of the root strategy adopted, which is $RiskyAndFast$, $MediumSafe$ or $SafeAndSlow$

When not fulfilled, by default, $rootStrategyImplementation = SafeAndSlow$.

Some methods :

getMaximumDistance

Usage : *getMaximumDistance*()

Arguments : none

Value : a positive real value, the distance from the center of the standard space until which we research an intersection with the limit state function along each direction. By default, the maximum distance is equal to 8

getOriginValue

Usage : *getOriginValue*()

Arguments : none

Value : a real value, the value of the limit state functino at the center of the standard space

getStepSize

Usage : *getStepSize*()

Arguments : none

Value : a real value, the lenght of each segment inside which the root research is performed.

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

6.7.3 RiskyAndFast

The RiskyAndFast strategy is the following : for each direction, we check whether there is a sign changement of the standard limit state function between the maximum distant point (at distance *MaximumDistance* from the center of the standard space) and the center of the standard space.
In case of sign changement, we research one root in the segment [origine, maximum distant point] with the selectionned non linear solver.
As soon as founded, the segment [root, infinity point] is considered within the failure space.

It inherits from the methods of the RootStrategy class.

Usage :

$RiskyAndFast()$

$RiskyAndFast(solver)$

$RiskyAndFast(solver, maximumDistance, stepSize)$

Arguments :

solver : a Solver, the non linear solver used to research the intersection of the limit state function with the direction, on each segment of length *stepSize*, between the center of the space and *maximumDistance* (root research)

maximumDistance : a real strictly positive value, the maximum distance within wich the root research is perfomed along each direction

stepSize : a real value, the length of each segment along a direction inside which the root research is performed

By default, $solver = Brent$, $maximumDistance = 8$, $stepSize = 1$

Some methods :

getSolver

Usage : $getSolver()$

Arguments : none

Value : a Solver, the non linear solver wich will research the root in a segment

solve

Usage : $solve(function, value)$

Arguments :

$function$: a NumericalMathFunction, from $\mathbb{R}$ into $\mathbb{R}$

$value$: a real value

Value : a ScalarCollection of dimension 1, the real value x such as $function(x) = value$ researched within $[origine, maximumDistance]$

Each $getMethod$ is associated to a $setMethod$.

©2007 EDF - EADS - PhiMeca

6.7.4 MediumSafe

The MediumSafe strategy is the following : for each direction, we go along the direction by step of length *stepSize* from the origin to the maximum distant point (at distance *MaximumDistance* from the center of the standard space) and we check whether there is a sign changement on each segment so formed.
At the first sign changement, we research one root in the concerned segment with the selectionned non linear solver. Then, the segment [root, maximum distant point] is considered within the failure space.
If *stepSize* is small enough, this strategy garantees us to find the root which is the nearest from the origine.

It inherits from the methods of the RootStrategy class.

Usage :

$MediumSafe()$

$MediumSafe(solver)$

$MediumSafe(solver, maximumDistance, stepSize)$

Arguments :

solver : a Solver, the non linear solver used to research the intersection of the limit state function with the direction, on each segment of length *stepSize*, between the center of the space and *maximumDistance* (root research),

maximumDistance : a real strictly positive value, the maximum distance within wich the root research is perfomed along each direction

stepSize : a real value. CARE : this value is not takjen into account in the root research : $stepSize = maximumDistance$ authomatically on the algorithm according to this root strategy

By default, $solver = Brent$, $maximumDistance = 8$

Some methods :

getSolver

Usage : $getSolver()$
Arguments : none
Value : a Solver, the non linear solver wich will research the root in a segment

solve

Usage : $solve(function, value)$
Arguments :
$function$: a NumericalMathFunction, from $\mathbb{R}$ into $\mathbb{R}$
value : a real value
Value : a ScalarCollection of dimension 1 (one root) : the real value x such as $function(x) = value$ researched the first segment of length *stepsize*, within $[origine, maximumDistance]$ where a sign changement of $function$ has been detected

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

6.7.5 SafeAndSlow

The SafeAndSlow strategy is the following : for each direction, we go along the direction by step of length *stepSize* from the origine to the maximum distant point(at distance *MaximumDistance* from the center of the standard space) and we check whether there is a sign changement on each segment so formed.
We go until the maximum distant point. Then, for all the segments where we detected a the presence of a root, we research the root with the selectionned non linear solver. We evaluate the contribution to the failure probability of each segment.
If *stepSize* is small enough, this strategy garantees us to find all the roots in the direction and the contribution of this direction to the failure probability is precisely evaluated.

It inherits from the methods of the RootStrategy class.

Usage :

$SafeAndSlow()$

$SafeAndSlow(solver)$

$SafeAndSlow(solver, maximumDistance, stepSize)$

Arguments :

solver : a Solver, the non linear solver used to research the intersection of the limit state function with the direction, on each segment of length *stepSize*, between the center of the space and *maximumDistance* (root research),

maximumDistance : a real strictly positive value, the maximum distance within wich the root research is perfomed along each direction

stepSize : a real value, the length of each segment along a direction inside which the root research is performed.

By default, $solver = Brent$, $maximumDistance = 8$, $stepSize = 1$

Some methods :

getSolver

Usage : $getSolver()$
Arguments : none
Value : a Solver, the non linear solver wich will research the root in a segment

solve

Usage : $solve(function, value)$
Arguments :
$function$: a NumericalMathFunction, from $\mathbb{R}$ into $\mathbb{R}$
$value$: a real value
Value : a ScalarCollection, all the real values x such as $function(x) = value$ researched in each segment of length *stepsize*, within $[origine, maximumDistance]$

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

6.7.6 SamplingStrategy

Usage :

SamplingStrategy()

SamplingStrategy(*samplingStrategyImplementation*)

SamplingStrategy(*dimension*)

Arguments :

samplingStrategyImplementation : a SamplingStrategyImplementation, the implementation of the sampling strategy adopted, which is *RandomDirection*, or *OrthogonalDirection*

dimension : an integer, the dimension of the standard space

By default, *samplingStrategyImplementation* = *RandomDirection* and *dimension* = 0 but the dimension automatically updated by the calling class

Some methods :

getDimension

Usage : *getDimension*()

Arguments : none

Value : an integer, the dimension of the standard space

Each *getMethod* is associated to a *setMethod*.

©2007 EDF - EADS - PhiMeca

6.7.7 RandomDirection

The RandomDirection strategy is the following : we generate some points on the sphere unity in the standard space according to the uniform distribution and we consider both opposite directions so built.

It inherits from the methods of the SamplingStrategy class.

Usage :

RandomDirection()

RandomDirection(*dimension*)

Arguments :

dimension : an integer, the dimension of the standard space

By default, $dimension = 0$ but automatically updated by the calling class

Some methods :

generate

Usage : *generate*()

Arguments : none

Value : a NumericalSample of size 2, two opposite random directions generated

getUniformUnitVectorRealization

Usage : $getUniformUnitVectorRealization(dimension)$

Arguments : *dimension* : an interger, the dimension of the sphere unity (which is the dimension of the standard space)

Value : a NumericalPoint, a realisation of a vector on the sphere unity, according to the uniform distribution

Each $getMethod$ is associated to a $setMethod$.

©2007 EDF - EADS - PhiMeca

6.7.8 OrthogonalDirection

The OrthogonalDirection strategy is the following : this strategy is parametered by $k \in \mathbb{N}$. We generate one direct orthonormalised base $(e_1, \ldots, e_n)$ within the set of orthonormalised bases. We consider all the renormalised linear combinations of k vectors within the n vectors of the base, where the coefficients of the linear combinations are equal to $+1, -1$. There are $C_n^k 2^k$ new vectors v_i. We consider each direction defined by each vector v_i.
If $k = 1$, we consider all the axes of the standard space.

It inherits from the methods of the SamplingStrategy class.

Usage :

$OrthogonalDirection()$

$OrthogonalDirection(dimension, size)$

Arguments :

$dimension$: an integer, dimension of the standard space

$size$: an integer, the number of elements in the linear combinations described here above

By default, $size = 1$ and $dimension = 0$ but automatically updated by the calling class.

Some methods :

generate

Usage : $generate()$

Arguments : none

Value : a NumericalSample, a realisation of a random direction according to the algorithm described here above.

$getUniformUnitVectorRealization$

Usage : $getUniformUnitVectorRealization(dimension)$

Arguments : $dimension$: an ineteger, the dimension of the sphere unity (dimension of the standard space)

Value : a NumericalPoint, a realisation of a vector on the sphere unity, according to the uniform distribution

Each $getMethod$ is associated to a $setMethod$.

©2007 EDF - EADS - PhiMeca

6.7.9 Solver

This class enables to solve 1D non linear equations :

$$f(x) = value,\ for\ x \in]infPoint, supPoint[$$

if f is a function from $\mathbb{R}$ in $\mathbb{R}$, $(infPoint, supPoint) \in \mathbb{R}^2$ and if f has a sign changement between the two bounds of the interval $]infPoint, supPoint[$, which means that $f(infPoint) * f(supPoint) < 0$.

In particular, it is used in the root research of a directional sampling simulation.

Usage :

$Solver(solverImplementation)$

$Solver(absoluteError, relativeError, maximumFunctionEvaluation)$

Arguments :

solverImplementation : a SolverImplementation, the implementation of particular solver which is *Bisection*, *Brent* or *Secant*,

absoluteError : a real positive value, absolute error : distance between two successif iterates at the end point

relativeError : a real positive value, relative distance between the two last successif iterates (with regards the last iterate)

maximumFunctionEvaluation : an integer, the maximum number of evaluations of the function

Some methods :

getAbsoluteError

Usage : $getAbsoluteError()$

Arguments : none

Value : a real positive value, the absolute error : distance between two successive iterates at the end point

getMaximumFunctionEvaluation

Usage : $getMaximumFunctionEvaluation()$

Arguments : none

Value : an integer, the maximum number of evaluations of the function

getRelativeError

Usage : $getRelativeError()$

Arguments : none

Value : a real positive value, the relative distance between the two last successive iterates (with regards the last iterate)

©2007 EDF - EADS - PhiMeca

6.7.10 Bisection

The Bisection solver is a bisection algorithm.

Usage :

$Bisection()$

$Bisection(absoluteError, relativeError, maximumFunctionEvaluation)$

Arguments :

$absoluteError$: a real positive value, absolute error : distance between two successif iterates at the end point

$relativeError$: a real positive value, relative distance between the two last successif iterates (with regards the last iterate)

$maximumFunctionEvaluation$: an integer, the maximum number of evaluations of the function

By default, $absoluteError = 1.e-5$, $relativeErro = 1.e-5$, $maximumFunctionEvaluation = 100$

Some methods :

solve

Usage :

$solve(function, value, infPoint, supPoint)$

$solve(function, value, infPoint, supPoint, infValue, supValue)$

Arguments :

$function$: a NumericalMathFunction, the function of the equation $function(x) = value$ we want to solve on $]infPoint, supPoint[$

$value$: a real value, the value of the equation $function(x) = value$ we want to solve on $]infPoint, supPoint[$

$infPoint$: a real value, the lower bound of the interval where we want to solve the equation

$supPoint$: a real value, the upper bound of the interval where we want to solve the equation

$infValue$: a real value, the value of $function$ on the point $infPoint$: $function(infPoint)$, must be of opposite sign of $supValue$

$supValue$: a real value, a real value, the value of $function$ on the point $supPoint$: $function(supPoint)$, must be of opposite sign of $infValue$

Value : a real value, the result of the root research, in $]infPoint, supPoint[$

Details : If the function f is continuous, the Bisection solver will converge towards a root of the equation $function(x) = value$ on $]infPoint, supPoint[$. If not, it will converge towards either a root or a discontinuity point of f on $]infPoint, supPoint[$. Bisection garantees a convergence.

Bisection may fail.

©2007 EDF - EADS - PhiMeca

6.7.11 Brent

The Brent solver is a mix of Bisection, Secant and inverse quadratic interpolation.

Usage :

$Brent()$

$Brent(absoluteError, relativeError, maximumFunctionEvaluation)$

Arguments :

$absoluteError$: a real positive value, the absolute error : distance between two successive iterates at the end point

$relativeError$: a real positive value,the relative distance between the two last successive iterates (with regards the last iterate)

$maximumFunctionEvaluation$: an integer, the maximum number of evaluations of the function

By default, $absoluteError = 1.e-5$, $relativeErro = 1.e-5$, $maximumFunctionEvaluation = 100$

Some methods :

solve

Usage :

$solve(function, value, infPoint, supPoint)$

$solve(function, value, infPoint, supPoint, infValue, supValue)$

Arguments :

$function$: a NumericalMathFunction, the function of the equation $function(x) = value$ we want to solve on $]infPoint, supPoint[$

$value$: a real value, the value of the equation $f(x) = value$ we want to solve on $]infPoint, supPoint[$

$infPoint$: a real value, the lower bound of the interval where we want to solve the equation

$supPoint$: a real value, the upper bound of the interval where we want to solve the equation

$infValue$: a real value, the value of $function$ on the point $infPoint$: $function(infPoint)$, must be of opposite sign of $supValue$

$supValue$: a real value, a real value, the value of $function$ on the point $supPoint$: $function(supPoint)$, must be of opposite sign of $infValue$

Value : the result of the root research, in $]infPoint, supPoint[$

Details : If the function f is continuous, the Brent solver will converge towards a root of the equation $function(x) = value$ on $]infPoint, supPoint[$. If not, it will converge towards either a root or a discontinuity point of f on $]infPoint, supPoint[$. Brent garantees a convergence.

©2007 EDF - EADS - PhiMeca

6.7.12 Secant

The Secant solver is based on the evaluation of a segment between the two last iterated points.

Usage :

$Secant()$

$Secant(absoluteError, relativeError, maximumFunctionEvaluation)$

Arguments :

$absoluteError$: a real positive value, absolute error : distance between two successif iterates at the end point

$relativeError$: a real positive value, relative distance between the two last successif iterates (with regards the last iterate)

$maximumFunctionEvaluation$: an integer, the maximum number of evaluations of the function

By default, $absoluteError = 1.e-5$, $relativeErro = 1.e-5$, $maximumFunctionEvaluation = 100$

Some methods :

solve

Usage :

$solve(function, value, infPoint, supPoint)$

$solve(function, value, infPoint, supPoint, infValue, supValue)$

Arguments :

$function$: a NumericalMathFunction, the function of the equation $function(x) = value$ we want to solve on $]infPoint, supPoint[$

$value$: a real value, the value of the equation $function(x) = value$ we want to solve on $]infPoint, supPoint[$

$infPoint$: a real value, the lower bound of the interval where we want to solve the equation

$supPoint$: a real value, the upper bound of the interval where we want to solve the equation

$infValue$: a real value, the value of $function$ on the point $infPoint$: $function(infPoint)$, must be of opposite sign of $supValue$

$supValue$: a real value, a real value, the value of $function$ on the point $supPoint$: $function(supPoint)$, must be of opposite sign of $infValue$

Value : the result of the root research, in $]infPoint, supPoint[$

Details : Secant might fail and not converge.

©2007 EDF - EADS - PhiMeca

6.8 ImportanceSampling

This class inherits from Simulation.

Usage : *ImportanceSampling*(*event*, *importanceDistribution*)

Arguments :

event : a Event, the event we want to evaluate the probability

importanceDistribution : a Distribution, the importance distribution of the Importance Sampling simulation method.

Some methods :

getImportanceDistribution

Usage : *getImportanceDistribution*()

Arguments : none

Value : a Distribution, the importance distribution of the Importance Sampling simulation method

©2007 EDF - EADS - PhiMeca

6.9 SimulationResult

Usage : structure created by the method run() of a Simulation, and obtained thanks to the method *getResult*()

Some methods :

getBlockSize

Usage : *getBlockSize*()

Arguments : none

Value : an integer, the number of terms in the probability simulation estimator grouped together

getCoefficientOfVariation

Usage : *getCoefficientOfVariation*()

Arguments : none

Value : a real value, the coefficient of variation of the simulated sample

getConfidenceLength

Usage : *getConfidenceLength*()

Arguments : none

Value : a positive real value, the length of any confidence interval equal to the double of the variance of the Monte Carlo estimator

getOuterSampling

Usage : *getOuterSampling*()

Arguments : none

Value : an integer, the number of groups of terms in the probability simulation estimator

Details : for Monte Carlo, LHS and Importance Sampling methods, the number of evaluations of the limit state function defining the event is : OuterSampling * BlockSize

getProbabilityEstimate

Usage : *getProbabilityEstimate*()

Arguments : none

Value : a positive real value, the Monte Carlo estimate of the event probability

getVarianceEstimate

Usage : *getVarianceEstimate*()

Arguments : none

Value : a positive real value, the variance of the Monte Carlo estimator, equal to the Mn(1-Mn) / n if Mn is the Monte Carlo probability estimator and n the size of the simulated sample

©2007 EDF - EADS - PhiMeca

7 QuadraticCumul

Usage : *QuadraticCumul*(*randVect*)

Arguments : *randVect* : a RandomVector, constraint : this RandomVector must be oy type Composite, which means it must have been defined with the second usage of declaration of a RandomVector (from a NumericalMathFunction and an antecedent Distribution)

Value : a QuadraticCumul

Some methods :

drawImportanceFactors

Usage : *drawImportanceFactors*()

Arguments : none

Value : a Graph, the structure containing the pie corresponding to the importance factors of the probabilistic variables

getCovariance

Usage : *getCovariance*()

Arguments : none

Value : a CovarianceMatrix, approximation of first order of the covariance matrix of the random vector

getImportanceFactors

Usage : *getImportanceFactors*()

Arguments : none

Value : a NumericalPoint, the importance factors of the inputs : only when *randVect* is of dimension 1

getMeanFirstOrder

Usage : *getMeanFirstOrder*()

Arguments : none

Value : a NumericalPoint, approximation at the first order of the mean of the random vector

getMeanSecondOrder

Usage : *getMeanSecondOrder*()

Arguments : none

Value : a NumericalPoint, approximation at the second order of the mean of the random vector (it requires that the hessian of the NumericalMathFunction has been defined)

©2007 EDF - EADS - PhiMeca

8 Response Surface Approximation

8.1 LinearTaylor

Usage : $LinearTaylor(center, function)$

Arguments :

$center$: a NumericalPoint, the point where the Taylor expansion of the function $function$ is performed

$function$: a NumericalMathFunction, the function to be approximated.

Value : a LinearTaylor

Some methods :

getInputFunction

Usage : $getInputFunction$
Arguments : none
Value : a NumericalMathFunction, the function $function$

getName

Usage : $getName()$
Arguments : none
Value : a string, the name of the LinearTaylor

getCenter

Usage : $getCenter()$
Arguments : none
Value : a NumericalPoint, around which the approximation has been made : $center$

run

Usage : $run()$
Arguments : none
Value : it performs the linear Taylor expansion around $center$ (while this method has not been executed, only $getInputFunction$, $getName$ and $setName$ methods can be used)

getConstant

Usage : $getConstant()$
Arguments : none
Value : a NumericalPoint, the constant vector of the approximation, equal to $function(center)$

getLinear

Usage : $getLinear()$
Arguments : none

©2007 EDF - EADS - PhiMeca

Value : a Matrix, the gradient of the function $function$ at the point *center* (the transposition of the jacobian matrix)

getResponseSurface

Usage : $getResponseSurface()$

Arguments : none

Value : a NumericalMathFunction, an approximation of the function $function$ by a linear Taylor expansion at the *center*

Links : see docref_SurfRep_Taylor

The methods $getInputFunction$, $getName$, $getCenter$ have their associated $setMethod$.

©2007 EDF - EADS - PhiMeca

8.2 QuadraticTaylor

Usage : $QuadraticTaylor(center, function)$

Arguments :

$center$: a NumericalPoint, the point where the quadratic Taylor expansion of the function $function$ is performed

$function$: a NumericalMathFunction, the function to be approximated : the gradient and hessian of the NumericalMathFunction must be defined.

Value : a QuadraticTaylor

Some methods :

getInputFunction

Usage : $getInputFunction$

Arguments : none

Value : a NumericalMathFunction ,the function $function$

getName

Usage : $getName()$

Arguments : none

Value : a string, the name of the Quadratic

getCenter

Usage : $getCenter()$

Arguments : none

Value : a NumericalPoint, around which the approximation has been made : $center$

run

Usage : $run()$

Arguments : none

Value : it performs the Quadratic Taylor expansion around $center$ (while this method has not been executed, only $getInputFunction$, $getName$ and $setName$ methods can be used)

getConstant

Usage : $getConstant()$

Arguments : none

Value : a NumericalPoint, the constant vector of the approximation, equal to $function(center)$

getLinear

Usage : $getLinear()$

Arguments : none

©2007 EDF - EADS - PhiMeca

Value : a Matrix, the gradient of the function $function$ at the point *center* (the transposition of the jacobian matrix)

getQuadratic

Usage : $getQuadratic()$

Arguments : none

Value : a SymmetricTensor which contains the 0.5 * transposition of the hessian values of $function$ at *center*

getResponseSurface

Usage : $getResponseSurface()$

Arguments : none

Value : a NumericalMathFunction, an approximation of the function $function$ by a Quadratic Taylor expansion at *center*

Links : see docref_SurfRep_Taylor

The methods $getInputFunction$, $getName$, $getCenter$ have their associated $setMethod$.

©2007 EDF - EADS - PhiMeca

8.3 LinearLeastSquares

Usage :

$LinearLeastSquares(dataIn, function)$
$LinearLeastSquares(dataIn, dataOut)$

Arguments :

$dataIn$: a NumericalSample, the input variables
$function$: a NumericalMathFunction, the function to be approximated
$dataOut$: a NumericalSample, the output variables

Value : a LinearLeastSquares, the linear least squares approximation between :

the two samples $dataIn$ and $dataOut$ in the case of the second usage
the two samples $dataIn$ and $function(dataIn)$ in the case of the first usage

Some methods :

getInputFunction
Usage : $getInputFunction()$
Arguments : none
Value : a NumericalMathfunctiontion the $function$ parameter in the case of the first usage

getDataIn
Usage : $getDataIn()$
Arguments : none
Value : a NumericalSample, the $dataIn$ parameter

getName
Usage : $getName()$
Arguments : none
Value : a string, the name of the LinearLeastSquares

run
Usage : $run()$
Arguments : none
Value : it performs the linear least squares approximation (while this method has not been executed, only $getInputfunctiontion$, $getDataIn$, $getName$ and $setName$ methods can be used)

getDataOut
Usage : $getDataIn()$
Arguments : none
Value : a NumericalSample, it returns the ouput variable :

©2007 EDF - EADS - PhiMeca

in the case of the first usage, it corresponds to the values of the function $function$ at the input variables $dataIn$: $function(dataIn)$

in the case of the second usage, it corresponds to $dataOut$

getLinear

Usage : $getLinear()$

Arguments : none

Value : a Matrix, the gradient of the function $function$ at the point *center* (the transposition of the jacobian matrix)

getResponseSurface

Usage : $getResponseSurface()$

Arguments : none

Value : a NumericalMathFunction, an approximation of the function $function$ by Linear Least Squares

Links : see docref_SurfRep_LeastSquare

The methods $getInputFunction$, $getName$, $getDataIn$ have their associated $setMethod$.

©2007 EDF - EADS - PhiMeca

8.4 QuadraticLeastSquares

Usage : $QuadraticLeastSquares(dataIn, function)$

$QuadraticLeastSquares(dataIn, dataOut)$

Arguments :

$dataIn$: a NumericalSample, the input variables

$function$: a NumericalMathFunction, the function to be approximated

$dataOut$: a NumericalSample, the output variables

Value : a QuadraticLeastSquares, the quadratic least squares approximation between :

the two samples $dataIn$ and $dataOut$ in the case of the second usage

the two samples $dataIn$ and $function(dataIn)$ in the case of the first usage

Some methods :

getInputFunction

Usage : $getInputFunction$

Arguments : none

Value : a NumericalMathFunction, the function $function$

getName

Usage : $getName()$

Arguments : none

Value : a string, the name of the QuadraticLeastSquares

getCenter

Usage : $getCenter()$

Arguments : none

Value : a NumericalPoint, around which the approximation has been made : *center*

run

Usage : $run()$

Arguments : none

Value : it performs the quadratic least squares approximation (while this method has not been executed, only $getInputFunction$, $getName$ and $setName$ methods can be used)

getDataOut

Usage : $getDataIn()$

Arguments : none

Value : a NumericalSample, it returns the ouput variable :

in the case of the first usage, it corresponds to the values of the function $function$ at the input variables $dataIn$: $function(dataIn)$

©2007 EDF - EADS - PhiMeca

in the case of the second usage, it corresponds to *dataOut*

getConstant

Usage : *getConstant*()
Arguments : none
Value : a NumericalPoint, the constant vector of the approximation, equal to $function(center)$

getLinear

Usage : *getLinear*()
Arguments : none
Value : a Matrix, the linear matrix of the approximation

getQuadratic

Usage : *getQuadratic*()
Arguments : none
Value : a SymmetricTensor, the quadratic term of the approximation

getResponseSurface

Usage : *getResponseSurface*()
Arguments : none
Value : a NumericalMathFunction, an approximation of the function *function* by Quadratic Least Squares

Links : see docref_SurfRep_LeastSquare

The methods *getInputFunction*, *getName*, *getDataIn* have their associated *setMethod*.

©2007 EDF - EADS - PhiMeca

9 Graphs

9.1 Graph

The class Graph is the structure which contains :

- the drawable elements (may be several drawables elements) : class Drawable
- the graphical context : the potential axes and labels, the bounding box, the global title, the global legend and its position

Usage :

$Graph(title, xTitle, yTitle, showAxes)$

$Graph(title, xTitle, yTitle, showAxes, legendPosition)$

$Graph(title, xTitle, yTitle, showAxes, legendPosition, legendFontSize)$

Arguments :

title : a String, the title of the graph

xTitle : a String, the legend of the X axe

yTitle : a String, the legend of the Y axe

showAxes : a boolean which indicates if the axes are drawn (yes = 1, no = 0)

legendPosition : a String which indicates the position of the legend. If *legendPosition* is not specified, the Graph has no legend

legendFontSize : an interger, the font size of the legend. If not specified, the default width wil be used

Some methods :

addDrawable

Usage : $addDrawable(aDrawable)$

Arguments : *aDrawable* : a Drawable, a drawable element we want to add on the graph

Value : none, it adds the new graph on the first one, with its legend. It keeps the graphical context of the first graph

draw

Usage :

$draw(path, file, width, height)$

$draw(file, width, height)$

$draw(file)$

Arguments :

path : a String which indicates the adress where the created file will be put. When not specified, the files is created in the cururent repertory

file : a String wich indicates the name of the created file (without the suffixe). The files created will be file.png and file.ps

width, *height* : two real positive values, number of pixels fixing the width and the height of the graph. When not specified, the couple (640,480) is taken into account

©2007 EDF - EADS - PhiMeca

Value : none : it generates the files file.png and file.ps

getAxes

Usage : *getAxes*()

Arguments : none

Value : a boolean which indicates if the axes are drawn (yes = 1, no = 0)

getBitmap

Usage : *getBitmap*()

Arguments : none

Value : a String, the adress of the file file.png created by the method draw

getBoundingBox

Usage : *getBoundingBox*()

Arguments : none

Value : a NumericalPoint of dimension 4, the bounding box of the drawable element, wich is a rectangle determined by its low and left corner (P1) and its high and right corner (P2). The BoundingBox is (XP1,YP1, XP2, YP2).

getDrawables

Usage : *getDrawables*()

Arguments : none

Value : a DrawableCollection, the collection of the Drawables included in the graph

getFileName

Usage : *getFileName*()

Arguments : none

Value : a String, the name of the files containing the graph

getLegendFontSize

Usage : *getLegendFontSize*()

Arguments : none

Value : a positive real, the legend font size

getLegendPosition

Usage : *getLegendPosition*()

Arguments : none

Value : a String, the position of the legend.

getPath

Usage : *getPath*()

Arguments : none

©2007 EDF - EADS - PhiMeca

Value : a String, the adress where the files file.png and file.ps are put

getPostscript

Usage : *getPostscript*()
Arguments : none
Value : a String, the adress of the file file.ps created by the method .draw()

getTitle

Usage : *getTitle*()
Arguments : none
Value : a String, the title of the graph

getXTitle

Usage : *getXTitle*()
Arguments : none
Value : a String, the title of the X axe

getYTitle

Usage : *getYTitle*()
Arguments : none
Value : a String, the title of the Y axe

setBoundingBox

Usage : *setBoundingBox*(*myBoundingBox*)
Arguments : *myBoundingBox* : a BoundingBox, which is a Numericalpoint(4) composed by $[x_{min}, x_{max}, y_{min}, y_{max}]$ if we want to impose the x-range to $[x_{min}, x_{max}]$ and the y-range to $[y_{min}, y_{max}]$.
Value : none

The methods *getAxes*, *getDrawables*, *getLegendPosition*, *getTitle*, *getXTitle*, *getYTitle*have their corresponding *setMethod*.

Here is the list of legend positions accepted by Open TURNS : "bottomright", "bottom", "bottomleft", "left", "topleft", "topright", "right", "center".

©2007 EDF - EADS - PhiMeca

9.2 Drawable

A Drawable is a drawable element described by :

- its data,
- their attributes : color, line stype, point style, fill style
- the specific legend of the drawable element.

Usage : *Drawable*(*drawableImplementation*)

Arguments : *drawableImplementation* : a DrawableImplementation, the implementation of Drawable, which is *Curve*, *Cloud*, *BarPlot*, *Staircase*, *Pie*

Some methods :

getBoundingBox

Usage : *getBoundingBox*()

Arguments : none

Value : a NumericalPoint of dimension 4, the bounding bow of the drawable element, wich is a rectangle determined by its low and left corner (P1)and its high and right corner (P2). The BoundingBox is (XP1,YP1, XP2, YP2).

getColor

Usage : *getColor*()

Arguments : none

Value : a String which describes the color of the lines within the drawable element none

getData

Usage : *getData*()

Arguments : none

Value : a NumericalSample, from which the Drawable is built

getFillStyle

Usage : *getFillStyle*()

Arguments : none

Value : a String which describes the fill style of the surfaces within the drawable element

getLabels

Usage : *getLabels*()

Arguments : none

Value : a Description, the labels of both axes

getLegendName

Usage : *getLegendName*()

©2007 EDF - EADS - PhiMeca

Arguments : none
Value : a String which is the legend of the drawable element

getLineStyle

Usage : *getLineStyle*()
Arguments : none
Value : a String which describes the style of the lines within the drawable element

getLineWidth

Usage : *getLineWidth*()
Arguments : none
Value : an interger, the width of the line included in the Drawable (if such the case)

getPointCode

Usage : *getPointCode*()
Arguments : none
Value : an integer which describes the style of the points within the drawable element

getPointStyle

Usage : *getPointStyle*()
Arguments : none
Value : a string which describes the style of the points within the drawable element

All the methods *getColor*, *getFillStyle*, *getLineStyle*, *getPointCode* and *getPointStyle* have their corresponding *setMethod*.

Here is the list of codes, styles, width accepted by Open TURNS :

- map matching keys with R codes for point symbols :

Point Style	Point Code
square	0
circle	1
triangleup	2
plus	3
times	4
diamond	5
triangledown	6
star	8
fsquare	15
fcircle	16
ftriangleup	17
fdiamond	18
bullet	20

©2007 EDF - EADS - PhiMeca

- pssible colors : "green", "red", "blue", "yellow", "darkblue", "orange", "lightgreen", "darkcyan", "cyan", "magenta", "darkgreen", "violet", "brown", "darkred", "pink", "ivory", "gold", "darkgrey", "grey", "white", "aliceblue", "antiquewhite", "antiquewhite1", "antiquewhite2", "antiquewhite3", "antiquewhite4", "aquamarine", "aquamarine1", "aquamarine2", "aquamarine3", "aquamarine4", "azure", "azure1", "azure2", "azure3", "azure4", "beige", "bisque", "bisque1", "bisque2", "bisque3", "bisque4", "black", "blanchedalmond", "blue1", "blue2", "blue3", "blue4", "blueviolet", "brown1", "brown2", "brown3", "brown4", "burlywood", "burlywood1", "burlywood2", "burlywood3", "burlywood4", "cadetblue", "cadetblue1", "cadetblue2", "cadetblue3", "cadetblue4", "chartreuse", "chartreuse1", "chartreuse2", "chartreuse3", "chartreuse4", "chocolate", "chocolate1", "chocolate2", "chocolate3", "chocolate4", "coral", "coral1", "coral2", "coral3", "coral4", "cornflowerblue", "cornsilk", "cornsilk1", "cornsilk2", "cornsilk3", "cornsilk4", "cyan1", "cyan2", "cyan3", "cyan4", "darkgoldenrod", "darkgoldenrod1", "darkgoldenrod2", "darkgoldenrod3", "darkgoldenrod4", "darkgray", "darkkhaki", "darkmagenta", "darkolivegreen", "darkolivegreen1", "darkolivegreen2", "darkolivegreen3", "darkolivegreen4", "darkorange", "darkorange1", "darkorange2", "darkorange3", "darkorange4", "darkorchid", "darkorchid1", "darkorchid2", "darkorchid3", "darkorchid4", "darksalmon", "darkseagreen", "darkseagreen1", "darkseagreen2", "darkseagreen3", "darkseagreen4", "darkslateblue", "darkslategray", "darkslategray1", "darkslategray2", "darkslategray3", "darkslategray4", "darkslategrey", "darkturquoise", "darkviolet", "deeppink", "deeppink1", "deeppink2", "deeppink3", "deeppink4", "deepskyblue", "deepskyblue1", "deepskyblue2", "deepskyblue3", "deepskyblue4", "dimgray", "dimgrey", "dodgerblue", "dodgerblue1", "dodgerblue2", "dodgerblue3", "dodgerblue4", "firebrick", "firebrick1", "firebrick2", "firebrick3", "firebrick4", "floralwhite", "forestgreen", "gainsboro", "ghostwhite", "gold1", "gold2", "gold3", "gold4", "goldenrod", "goldenrod1", "goldenrod2", "goldenrod3", "goldenrod4", "gray", "gray0", "gray1", "gray2", "gray3", "gray4", "gray5", "gray6", "gray7", "gray8", "gray9", "gray10", "gray11", "gray12", "gray13", "gray14", "gray15", "gray16", "gray17", "gray18", "gray19", "gray20", "gray21", "gray22", "gray23", "gray24", "gray25", "gray26", "gray27", "gray28", "gray29", "gray30", "gray31", "gray32", "gray33", "gray34", "gray35", "gray36", "gray37", "gray38", "gray39", "gray40", "gray41", "gray42", "gray43", "gray44", "gray45", "gray46", "gray47", "gray48", "gray49", "gray50", "gray51", "gray52", "gray53", "gray54", "gray55", "gray56", "gray57", "gray58", "gray59", "gray60", "gray61", "gray62", "gray63", "gray64", "gray65", "gray66", "gray67", "gray68", "gray69", "gray70", "gray71", "gray72", "gray73", "gray74", "gray75", "gray76", "gray77", "gray78", "gray79", "gray80", "gray81", "gray82", "gray83", "gray84", "gray85", "gray86", "gray87", "gray88", "gray89", "gray90", "gray91", "gray92", "gray93", "gray94", "gray95", "gray96", "gray97", "gray98", "gray99", "gray100", "green1", "green2", "green3", "green4", "greenyellow", "grey0", "grey1", "grey2", "grey3", "grey4", "grey5", "grey6", "grey7", "grey8", "grey9", "grey10", "grey11", "grey12", "grey13", "grey14", "grey15", "grey16", "grey17", "grey18", "grey19", "grey20", "grey21", "grey22", "grey23", "grey24", "grey25", "grey26", "grey27", "grey28", "grey29", "grey30", "grey31", "grey32", "grey33", "grey34", "grey35", "grey36", "grey37", "grey38", "grey39", "grey40", "grey41", "grey42", "grey43", "grey44", "grey45", "grey46", "grey47", "grey48", "grey49", "grey50", "grey51", "grey52", "grey53", "grey54", "grey55", "grey56", "grey57", "grey58", "grey59", "grey60", "grey61", "grey62", "grey63", "grey64", "grey65", "grey66", "grey67", "grey68", "grey69", "grey70", "grey71", "grey72", "grey73", "grey74", "grey75", "grey76", "grey77", "grey78", "grey79", "grey80", "grey81", "grey82", "grey83", "grey84", "grey85", "grey86", "grey87", "grey88", "grey89", "grey90", "grey91", "grey92", "grey93", "grey94", "grey95", "grey96", "grey97", "grey98", "grey99", "grey100", "honeydew", "honeydew1", "honeydew2", "honeydew3", "honeydew4", "hotpink", "hotpink1", "hotpink2", "hotpink3", "hotpink4", "indianred", "indianred1", "indianred2", "indianred3", "indianred4", "ivory1", "ivory2", "ivory3", "ivory4", "khaki", "khaki1", "khaki2", "khaki3", "khaki4", "lavender", "lavenderblush", "lavenderblush1", "lavenderblush2", "lavenderblush3", "lavenderblush4", "lawngreen", "lemonchiffon", "lemonchiffon1", "lemonchiffon2", "lemonchiffon3", "lemonchiffon4", "lightblue", "lightblue1", "lightblue2", "lightblue3", "lightblue4", "lightcoral", "lightcyan", "lightcyan1", "lightcyan2", "lightcyan3", "lightcyan4",

"lightgoldenrod", "lightgoldenrod1", "lightgoldenrod2", "lightgoldenrod3", "lightgoldenrod4", "lightgoldenrodyellow", "lightgray", "lightgrey", "lightpink", "lightpink1", "lightpink2", "lightpink3", "lightpink4", "lightsalmon", "lightsalmon1", "lightsalmon2", "lightsalmon3", "lightsalmon4", "lightseagreen", "lightskyblue", "lightskyblue1", "lightskyblue2", "lightskyblue3", "lightskyblue4", "lightslateblue", "lightslategray", "lightslategrey", "lightsteelblue", "lightsteelblue1", "lightsteelblue2", "lightsteelblue3", "lightsteelblue4", "lightyellow", "lightyellow1", "lightyellow2", "lightyellow3", "lightyellow4", "limegreen", "linen", "magenta1", "magenta2", "magenta3", "magenta4", "maroon", "maroon1", "maroon2", "maroon3", "maroon4", "mediumaquamarine", "mediumblue", "mediumorchid", "mediumorchid1", "mediumorchid2", "mediumorchid3", "mediumorchid4", "mediumpurple", "mediumpurple1", "mediumpurple2", "mediumpurple3", "mediumpurple4", "mediumseagreen", "mediumslateblue", "mediumspringgreen", "mediumturquoise", "mediumvioletred", "midnightblue", "mintcream", "mistyrose", "mistyrose1", "mistyrose2", "mistyrose3", "mistyrose4", "moccasin", "navajowhite", "navajowhite1", "navajowhite2", "navajowhite3", "navajowhite4", "navy", "navyblue", "oldlace", "olivedrab", "olivedrab1", "olivedrab2", "olivedrab3", "olivedrab4", "orange1", "orange2", "orange3", "orange4", "orangered", "orangered1", "orangered2", "orangered3", "orangered4", "orchid", "orchid1", "orchid2", "orchid3", "orchid4", "palegoldenrod", "palegreen", "palegreen1", "palegreen2", "palegreen3", "palegreen4", "paleturquoise", "paleturquoise1", "paleturquoise2", "paleturquoise3", "paleturquoise4", "palevioletred", "palevioletred1", "palevioletred2", "palevioletred3", "palevioletred4", "papayawhip", "peachpuff", "peachpuff1", "peachpuff2", "peachpuff3", "peachpuff4", "peru", "pink1", "pink2", "pink3", "pink4", "plum", "plum1", "plum2", "plum3", "plum4", "powderblue", "purple", "purple1", "purple2", "purple3", "purple4", "red1", "red2", "red3", "red4", "rosybrown", "rosybrown1", "rosybrown2", "rosybrown3", "rosybrown4", "royalblue", "royalblue1", "royalblue2", "royalblue3", "royalblue4", "saddlebrown", "salmon", "salmon1", "salmon2", "salmon3", "salmon4", "sandybrown", "seagreen", "seagreen1", "seagreen2", "seagreen3", "seagreen4", "seashell", "seashell1", "seashell2", "seashell3", "seashell4", "sienna", "sienna1", "sienna2", "sienna3", "sienna4", "skyblue", "skyblue1", "skyblue2", "skyblue3", "skyblue4", "slateblue", "slateblue1", "slateblue2", "slateblue3", "slateblue4", "slategray", "slategray1", "slategray2", "slategray3", "slategray4", "slategrey", "snow", "snow1", "snow2", "snow3", "snow4", "springgreen", "springgreen1", "springgreen2", "springgreen3", "springgreen4", "steelblue", "steelblue1", "steelblue2", "steelblue3", "steelblue4", "tan", "tan1", "tan2", "tan3", "tan4", "thistle", "thistle1", "thistle2", "thistle3", "thistle4", "tomato", "tomato1", "tomato2", "tomato3", "tomato4", "turquoise", "turquoise1", "turquoise2", "turquoise3", "turquoise4", "violetred", "violetred1", "violetred2", "violetred3", "violetred4", "wheat", "wheat1", "wheat2", "wheat3", "wheat4", "whitesmoke", "yellow1", "yellow2", "yellow3", "yellow4", "yellowgreen"

- line styles : "blank", "solid", "dashed", "dotted", "dotdash", "longdash", "twodash"
- some fill styles (patterns) : "solid", "shaded"

The default values are the following ones :

- $Color = "blue"$
- $SurfaceColor = "white"$
- $FillStyle = "solid"$
- $PointStyle = "plus"$
- $LineWidth = 1$
- $LineStyle = "solid"$
- $Pattern = "s"$

©2007 EDF - EADS - PhiMeca

9.3 Curve

It inherits from the methods of the Drawable class.

Usage :

$Curve(data, legend)$

$Curve(data, color, lineStyle, lineWidth, legend)$

Arguments :

data : a NumericalSample, the points from which the curve is built, must be of dimension 2

legend : a String, the legend

color : a String, the color of the curve

lineStyle : a String, the style of the curve

lineWidth : an integer, the line width of the curve

Some methods :

isConformData

Usage : $isConformData(data)$

Arguments : *data* : a NumericalSample

Value : a boolean which indicates if the type of data is conform to the type of the drawable (here a Curve) : a NumericalSample of dimension 2

getData

Usage : $getData()$

Arguments : none

Value : a NumericalSample of dimension 2, the data from which the curve is built

getLineWidth

Usage : $getLineWidth()$

Arguments : none

Value : an integer, the line width of the curve

All the methods *getColor*, *getLegendName*, *getLineStyle* and *getLineWidth* have their corresponding *setMethod*.

©2007 EDF - EADS - PhiMeca

9.4 Cloud

It inherits from the methods of the Drawable class.

Usage :

$Cloud(data, legend)$

$Cloud(data, color, pointStyle, legend)$

Arguments :

data : a NumericalSample, the points from which the cloud is built, must be of dimension 2

legend : a String, the legend

color : a String, the color of the curve . If not specified, by default equal to "blue"

pointStyle : a String, the style of the points. If not specified, by default equal to "plus"

Some methods :

$isConformData$

Usage : $isConformData(data)$

Arguments : *data* : a NumericalSample

Value : a boolean which indicates if the type of data is conform to the type of the drawable : a NumericalSample of dimension 2

$getData$

Usage : $getData()$

Arguments : none

Value : a NumericalSample of dimension 2, the data from which the cloud is built

All the methods $getColor$, $getLegendName$, $getPointCode$ and $getPointStyle$ have their corresponding $setMethod$.

©2007 EDF - EADS - PhiMeca

9.5 BarPlot

It inherits from the methods of the Drawable class.

Usage :

$BarPlot(data, origin, legend)$

$BarPlot(data, origin, color, fillStyle, lineStyle, legend)$

Arguments :

data : a NumericalSample, the data from which the BarPlot is built, must be of dimension 2 : the discontinuous points and their corresponding height

origin : a real value which is where the BarPlot begins

legend : a String, the legend

color : a String, the color of the curve . If not specified, by default equal to "blue"

lineStyle : a String, the style of the curve. If not specified, by default equal to "solid"

$fillStyle$: a String, the fill style of the surfaces. If not specified, by default equal to "solid"

Some methods :

getData

Usage : $getData()$

Arguments : none

Value : a NumericalSample, of dimension 2, giving the discontinuous points and their corresponding height

getOrigin

Usage : $getOrigin()$

Arguments : none

Value : a real value which is where the BarPlot begins

$isConformData$

Usage : $isConformData(data)$

Arguments : *data* : a NumericalSample

Value : a boolean which indicates if the type of data is conform to the type of the drawable (here a BarPlot) : a NumericalSample, of dimension 2

All the methods *getColor*, *getLegendName*, *getOrigin*, *getFillStyle* and *getLineStyle* have their corresponding *setMethod*.

©2007 EDF - EADS - PhiMeca

9.6 Staircase

It inherits from the methods of the Drawable class.

Usage :

$StairCase(data, legend)$

$StairCase(data, color, lineStyle, pattern, legend)$

Arguments :

data : a NumericalSample, the points from which the Staircase is built, must be of dimension 2 : the discontinuous points and their corresponding height

legend : a String, the legend

color : a String, the color of the curve If not specified, by default equal to "blue"

lineStyle : a String, the style of the curve. If not specified, by default equal to "solid"

pattern : a String, the pattern of the surfaces. If not specified, by default equal to "s"

Some methods :

getData

Usage : $getData()$

Arguments : none

Value : a NumericalSample, of dimension 2, giving the discontinuous points and their corresponding heigth

isConformData

Usage : $isConformData(data)$

Arguments : *data* : a NumericalSample

Value : a boolean which indicates if the type of data is conform to the type of the drawable (here a Staircase) : a NumericalSampl of dimension 2, giving the discontinuous points and their corresponding height

getPattern

Usage : $getPattern()$

Arguments : none

Value :

All the methods *getColor*, *getLegendName*, *getLineStyle* and *getPattern* have their corresponding *setMethod*.

©2007 EDF - EADS - PhiMeca

9.7 Pie

It inherits from the methods of the Drawable class.

Usage :

$Pie(data)$

$Pie(data, labels, center, radius, palette)$

Arguments :

data : a NumericalSample, of dimension 1, giving the percentiles of the pie

labels : a StringCollection, the names of each group. If not specified, by default equal to the descrition of the probabilistic input vector

center : a NumericalPoint, the center of the pie inside the bounding box. If not specified, by default equal to $(0, 0)$

radius : a real positive value, the radius of the pie. If not specified, by default equal to 1

palette : a StringCollection, the names of the colors. If not specified, colors are successively taken from the list given below, in the same order

Some methods :

getCenter

Usage : $getCenter()$
Arguments : none
Value : a NumericalPoint, the center of the pie inside the bounding box

getData

Usage : $getData()$
Arguments : none
Value : a NumericalSample of dimension 1, giving the percentiles of the pie

getLabels

Usage : $getLabels()$
Arguments : none
Value : a StringCollection, the names of each group

getPalette

Usage : $getPalette()$
Arguments : none
Value : a StringCollection, the names of the colors used for the pie

getRadius

Usage : $getRadius()$
Arguments : none

©2007 EDF - EADS - PhiMeca

Value : a real positive value, the radius of the pie

isConformData

Usage : *isConformData*(*data*)

Arguments : *data* : a NumericalSample

Value : a boolean which indicates if the type of data is conform to the type of the drawable (here a Pie) : a NumericalSample of dimension 1

getPattern

Usage : *getPattern*()

Arguments : none

Value :

All the methods *getColor*, *getLegendName*, *getLineStyle* and *getPattern* have their corresponding *setMethod*.

9.8 Show

The command *Show* enables to visualise a Graph whithin the TUI whithout creating the files .EPS, .PNG or .FIG.

Usage : *Show*(*graph*)

Arguments : *graph* : a Graph

Value : It shows the graph whithin the TUI whitoouht saving it in any file.

Details : For example, *Show*(*myDistribution.drawPDF*()) where *myDistribution* is a Distribution.

©2007 EDF - EADS - PhiMeca