Package ‘mtk’

February 20, 2015

Type Package

Title Mexico ToolKit library (MTK)

Encoding UTF-8

Version 1.0

Date 2014-07-15

Author Juhui WANG [aut, cre] (Software and Engineering), Hervé MONOD [aut] (Applications and Statistical Methods), Robert FAIVRE [ctb] (Applications and Statistical Methods), Hervé RICHARD [ctb] (Software and Engineering)

Maintainer Juhui WANG <Juhui.Wang@jouy.inra.fr>

Description MTK (Mexico ToolKit) is a generic platform for the sensitivity and uncertainty analysis of complex models. It provides functions and facilities for experimental design, model simulation, sensitivity and uncertainty analysis, methods integration and data reporting, etc.

License GPL-3

LazyLoad yes

Depends R (>= 2.15.0), base, stringr, graphics, methods, XML, sensitivity, lhs, rgl

Suggests MASS

Collate 'mtkAllGenerics.R' 'globalsMtkFuncts.R' 'mtkValue.R'
 'mtkFeature.R' 'mtkLevels.R' 'mtkParameter.R' 'mtkDomain.R'
 'mtkFactor.R' 'mtkExpFactors.R' 'mtkProcess.R'
 'mtkExpWorkflow.R' 'mtkExperiment.R' 'mtkParsor.R'
 'mtkResult.R' 'mtkDesignerResult.R' 'mtkDesigner.R'
 'mtkMorrisDesigner.R' 'mtkBasicMonteCarloDesigner.R'
 'mtkRandLHSDesigner.R' 'mtkNativeDesigner.R'
 'mtkSobolDesigner.R' 'mtkFastDesigner.R' 'mtkEvaluatorResult.R'
 'mtkEvaluator.R' 'mtkNativeEvaluator.R'
 'mtkIshigamiEvaluator.R' 'mtkWWDMEvaluator.R'
 'mtkSystemEvaluatorResult.R' 'mtkSystemEvaluator.R'
 'mtkAnalyserResult.R' 'mtkAnalyser.R' 'mtkNativeAnalyser.R'
 'mtkDefaultAnalyser.R' 'mtkRegressionAnalyser.R'
 'mtkMorrisAnalyser.R' 'mtkSobolAnalyser.R' 'mtkPLMMAnalyser.R'
 'mtkFastAnalyser.R' 'mtk.addons.R' 'mtkPackage.R'
R topics documented:

mtk-package .............................. 5
addProcess-methods ........................ 8
ANY .......................................... 9
BasicMonteCarlo .......................... 10
deleteProcess-methods .................. 11
eextractData-methods .................. 13
Fast .......................................... 14
getData-methods .................. 17
generateDiscreteDistributionLevels-methods .................. 18
generateDiscreteDistributionType-methods ................. 19
generateDiscreteDistributionWeights-methods ........... 20
generateDistributionName-methods ................. 20
generateDistributionNames-methods ................. 21
generateDistributionNominalValue-methods .......... 22
generateDistributionNominalValues-methods .......... 23
generateDistributionNominalValue-type-methods .......... 24
generateDistributionNominalValues-methods .......... 24
generateDistributionParameters-methods .......... 25
generateDomain-methods ................. 26
generateFactorFeatures-methods ................. 27
generateFactorNames-methods ................. 28
generateFactors-methods ................. 29
generateFeatures-methods ................. 30
generateLevels-methods ................. 30
generateMTKFeatures-methods ............. 31
generateName-methods .................. 32
generateNames-methods .................. 32
generateNominalValue-methods ............. 33
generateNominalValue-type-methods ............. 34
generateParameters-methods ............. 35
generateProcess-methods ............. 36
generateResult-methods ............. 37
generateType-methods ............. 38
generateValue-methods ............. 39
generateWeights-methods ............. 40
is.finished-methods .................. 41
is.ready-methods .................. 42
Ishigami .................................. 43
Ishigami.factors .................. 45
make.mtkFactor .................. 46
make.mtkFeatureList ................. 47
topics documented:

- `make.mtkParameterList` ........................................... 48
- `Morris` .................................................................. 48
- `mtk.analyserAddons` ............................................... 51
- `mtk.designerAddons` ............................................... 54
- `mtk.evaluatorAddons` ............................................ 56
- `mtkAnalyser` .......................................................... 58
- `mtkAnalyser-class` ................................................... 59
- `mtkAnalyserResult` .................................................. 61
- `mtkAnalyserResult-class` ......................................... 62
- `mtkBasicMonteCarloDesigner` .................................. 63
- `mtkBasicMonteCarloDesigner-class` ......................... 64
- `mtkBasicMonteCarloDesignerResult` .......................... 66
- `mtkBasicMonteCarloDesignerResult-class` ............... 67
- `mtkDefaultAnalyser` ............................................... 68
- `mtkDefaultAnalyser-class` ...................................... 69
- `mtkDesigner` .......................................................... 71
- `mtkDesigner-class` ................................................. 72
- `mtkDesignerResult` .................................................. 74
- `mtkDesignerResult-class` ......................................... 75
- `mtkDomain` ............................................................ 76
- `mtkDomain-class` ..................................................... 77
- `mtkEvaluator` .......................................................... 78
- `mtkEvaluator-class` .................................................. 80
- `mtkEvaluatorResult` ............................................... 81
- `mtkEvaluatorResult-class` ....................................... 82
- `mtkExperiment` ........................................................ 83
- `mtkExperiment-class` ............................................... 85
- `mtkExpFactors` ......................................................... 87
- `mtkExpFactors-class` ................................................ 87
- `mtkExpWorkflow` ...................................................... 89
- `mtkExpWorkflow-class` ............................................ 91
- `mtkFactor` .............................................................. 93
- `mtkFactor-class` ...................................................... 94
- `mtkFastAnalyser` ...................................................... 96
- `mtkFastAnalyser-class` .......................................... 97
- `mtkFastAnalyserResult` .......................................... 99
- `mtkFastAnalyserResult-class` .................................. 100
- `mtkFastDesigner` ...................................................... 101
- `mtkFastDesigner-class` ........................................... 102
- `mtkFastDesignerResult` .......................................... 105
- `mtkFastDesignerResult-class` ................................ 106
- `mtkFeature` ............................................................ 107
- `mtkFeature-class` ..................................................... 108
- `mtkIshigamiEvaluator` ............................................ 109
- `mtkIshigamiEvaluator-class` .................................. 110
- `mtkLevels` ............................................................. 112
- `mtkLevels-class` ..................................................... 113
- `mtkMorrisAnalyser` .................................................. 114
<table>
<thead>
<tr>
<th>Topic Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>mtkMorrisAnalyser-class</td>
<td>115</td>
</tr>
<tr>
<td>mtkMorrisAnalyserResult</td>
<td>117</td>
</tr>
<tr>
<td>mtkMorrisAnalyserResult-class</td>
<td>118</td>
</tr>
<tr>
<td>mtkMorrisDesigner</td>
<td>119</td>
</tr>
<tr>
<td>mtkMorrisDesigner-class</td>
<td>121</td>
</tr>
<tr>
<td>mtkMorrisDesignerResult</td>
<td>123</td>
</tr>
<tr>
<td>mtkMorrisDesignerResult-class</td>
<td>124</td>
</tr>
<tr>
<td>mtkNativeAnalyser</td>
<td>125</td>
</tr>
<tr>
<td>mtkNativeAnalyser-class</td>
<td>126</td>
</tr>
<tr>
<td>mtkNativeDesigner</td>
<td>128</td>
</tr>
<tr>
<td>mtkNativeDesigner-class</td>
<td>129</td>
</tr>
<tr>
<td>mtkNativeEvaluator</td>
<td>131</td>
</tr>
<tr>
<td>mtkNativeEvaluator-class</td>
<td>134</td>
</tr>
<tr>
<td>mtkParameter</td>
<td>136</td>
</tr>
<tr>
<td>mtkParameter-class</td>
<td>137</td>
</tr>
<tr>
<td>mtkParsor</td>
<td>138</td>
</tr>
<tr>
<td>mtkParsor-class</td>
<td>139</td>
</tr>
<tr>
<td>mtkPLMMAnalyser</td>
<td>140</td>
</tr>
<tr>
<td>mtkPLMMAnalyser-class</td>
<td>141</td>
</tr>
<tr>
<td>mtkPLMMAnalyserResult</td>
<td>142</td>
</tr>
<tr>
<td>mtkPLMMAnalyserResult-class</td>
<td>144</td>
</tr>
<tr>
<td>mtkProcess</td>
<td>145</td>
</tr>
<tr>
<td>mtkProcess-class</td>
<td>146</td>
</tr>
<tr>
<td>mtkRandLHSDesigner</td>
<td>147</td>
</tr>
<tr>
<td>mtkRandLHSDesigner-class</td>
<td>149</td>
</tr>
<tr>
<td>mtkRandLHSDesignerResult</td>
<td>151</td>
</tr>
<tr>
<td>mtkRandLHSDesignerResult-class</td>
<td>152</td>
</tr>
<tr>
<td>mtkReadFactors-methods</td>
<td>153</td>
</tr>
<tr>
<td>mtkRegressionAnalyser</td>
<td>153</td>
</tr>
<tr>
<td>mtkRegressionAnalyser-class</td>
<td>155</td>
</tr>
<tr>
<td>mtkRegressionAnalyserResult</td>
<td>157</td>
</tr>
<tr>
<td>mtkRegressionAnalyserResult-class</td>
<td>158</td>
</tr>
<tr>
<td>mtkResult</td>
<td>159</td>
</tr>
<tr>
<td>mtkResult-class</td>
<td>160</td>
</tr>
<tr>
<td>mtkSobolAnalyser</td>
<td>161</td>
</tr>
<tr>
<td>mtkSobolAnalyser-class</td>
<td>162</td>
</tr>
<tr>
<td>mtkSobolAnalyserResult</td>
<td>163</td>
</tr>
<tr>
<td>mtkSobolAnalyserResult-class</td>
<td>164</td>
</tr>
<tr>
<td>mtkSobolDesigner</td>
<td>165</td>
</tr>
<tr>
<td>mtkSobolDesigner-class</td>
<td>166</td>
</tr>
<tr>
<td>mtkSobolDesignerResult</td>
<td>168</td>
</tr>
<tr>
<td>mtkSobolDesignerResult-class</td>
<td>168</td>
</tr>
<tr>
<td>mtkSystemEvaluator</td>
<td>169</td>
</tr>
<tr>
<td>mtkSystemEvaluator-class</td>
<td>170</td>
</tr>
<tr>
<td>mtkSystemEvaluatorResult</td>
<td>172</td>
</tr>
<tr>
<td>mtkSystemEvaluatorResult-class</td>
<td>173</td>
</tr>
<tr>
<td>mtkValue</td>
<td>174</td>
</tr>
<tr>
<td>mtkValue-class</td>
<td>174</td>
</tr>
</tbody>
</table>
### MTK (Mexico ToolKit) for Sensitivity Analysis and Numerical Experiments

**Description**

MTK is an R package for sensitivity analysis and numerical experiments. Three examples are provided:

- "Ishigami" model analysis with the "BasicMonteCarlo" and "Regression" methods.
- Using the "mtk" package from a XML file.
- "WWDM (Winter Wheat Dry Matter)" model analysis with the "Morris" methods.

---

<table>
<thead>
<tr>
<th>mtk-package</th>
<th>MTK (Mexico ToolKit) for Sensitivity Analysis and Numerical Experiments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
</tbody>
</table>
To run the examples, just load the package and type respectively:

- demo(demo1,package="mtk", ask=FALSE)
- demo(demo2,package="mtk", ask=FALSE)
- demo(demo3,package="mtk", ask=FALSE)

The following methods and models are available for the current release:

- The "Fast" methods for experiments design and sensitivity index calculation. see help(Fast).
- The "Morris" methods for experiments design and sensitivity index calculation. see help(Morris).
- The "Sobol" methods for experiments design and sensitivity index calculation. see help(Sobol).
- The "Monte-Carlo" methods for experiments design. see help(BasiMonteCarlo).
- The "LHS" methods for experiments design. see help(RandLHS).
- The "PLMM (Polynomial Linear Meta-Model)" methods for sensitivity analysis. see help(PLMM).
- The "Regression" methods for sensitivity index calculation. see help(Regression).
- The "Ishigami" model for model simulation. see help(Ishigami).
- The "WWDM (Winter Wheat Dry Matter)" model for model simulation. see help(WWDM).

Author(s)

The Mexico Group. Contact: Juhui WANG, MIA-Jouy, Juhui.Wang@jouy.inra.fr,

References


Examples

```r
### Example 1: Sensitivity analysis of the "Ishigami" model ###

# Specify the factors to analyze:
x1 <- make.mtkFactor(name="x1",
                    distribName="unif", distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
                    distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
                    distribPara=list(min=-pi, max=pi))
factors <- mtkExpFactors(list(x1,x2,x3))
# Build the processes:
# 1) the experimental design process with the method "Morris".
expl.Designer <- mtkMorrisDesigner(listParameters = list(r=20,type=“oat”,levels=4,grid.jump=2))
# 2) the model simulation process with the model "Ishigami".
expl.Evaluator <- mtkIshigamiEvaluator()
```
# 3) the analysis process with the method "Morris".
exp1.Analyser <- mtkMorrisAnalyser()

# Build the workflow with the processes defined previously.
exp1 <- mtkExpWorkflow(expFactors=factors,
 processesVector = c(
   design=exp1.Designer,
   evaluate=exp1.Evaluator,
   analyse=exp1.Analyser)
)

# Run the workflow and reports the results.
run(exp1)
print(exp1)

# Create a new process with the analysis method «Regression».
exp1.AnalyserReg <- mtkRegressionAnalyser(listParameters
 =list(nboot=20)
)

# Re-analyze the model "Ishigami" with the method "Regression":
# replace the process, run the workflow and report the results
setProcess(exp1, exp1.AnalyserReg, "analyse")
run(exp1)
print(exp1)

### Example 2 : Sensitivity analysis from a XML file ###

## XML file is held in the directory of the library: “inst/extdata/”

# Specify the XML file’s name
xmlFile <- "WWDM_morris.xml"

## Find where the examples are held.
xmlFilePath <- paste(path.package("mtk", quiet = TRUE),
 "/extdata/",xmlFile,sep = "")

## Create the workflow
## Nota: If your XML file is local file for example "/var/tmp/X.xml",
## you should create the workflow as follows:
## workflow <- mtkExpWorkflow(xmlFilePath="/var/tmp/X.xml")
workflow <- mtkExpWorkflow(xmlFilePath=xmlFilePath)

# Run the workflow and report the results
run(workflow)
summary(workflow)
addProcess-methods  

The addProcess method

Description

Adds a process to the workflow.

Usage

addProcess(this, p, name)

Arguments

this  
an object of the class mtkExpWorkflow.

p  
an object of the class mtkProcess.

name  
a string from "design", "evaluate", or "analyze" to specify the type of process to add.

Value

invisible()

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# Define the factors

x1 <- make.mtkFactor(name="x1", distribName="unif", 
distribPara=list(min=-pi, max=pi))

x2 <- make.mtkFactor(name="x2", distribName="unif", 
distribPara=list(min=-pi, max=pi))

x3 <- make.mtkFactor(name="x3", distribName="unif", 
distribPara=list(min=-pi, max=pi))

ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# Create a workflow to manager the processes used for the analysis task
ANY Reg

# Create a designer to generate the experiments design and
# put the designer under control of the workflow
designer <- mtkNativeDesigner("BasicMonteCarlo",
  information=list(size=20))
addProcess(ishiReg, designer, name="design")

# Creates an evaluator and add it to the workflow
model <- mtkNativeEvaluator("Ishigami")
addProcess(ishiReg, model, name="evaluate")

# Create a analyser and add it to the workflow
analyser <- mtkNativeAnalyser("Regression")
addProcess(ishiReg, analyser, name="analyze")

# Run the workflow and reports the results
run(ishiReg)
summary(ishiReg)

---

**ANY**

*The ANY class*

**Description**

ANY is a data type to represent any S4 class.

**Details**

S4 implements the ANY class, but does not document it.

**Examples**

# creates a new class with "ANY"
setClass(Class="mtkProcess",
  representation=representation(
    name="character",
    protocol="character",
    site="character",
    service="character",
    parameters="ANY",
BasicMonteCarlo

Description

A native \texttt{mtk} design method to generate Monte Carlo samples.

Usage

- \texttt{mtkBasisMonteCarloDesigner(listParameters=NULL)}
- \texttt{mtkNativeDesigner(design="BasicMonteCarlo", information=NULL)}

Parameters

\texttt{size} : the sample size.

Details

1. The \texttt{mtk} implementation of the Basic Monte-Carlo method includes the following classes:
   - \texttt{mtkBasisMonteCarloDesigner} for Basic Monte-Carlo design processes.
   - \texttt{mtkBasisMonteCarloDesignerResult} to store and manage the design.
2. Many ways to create a Basic Monte-Carlo designer are available in \texttt{mtk}, but we recommend the following class constructors: \texttt{mtkBasisMonteCarloDesigner} or \texttt{mtkNativeDesigner}.

References

Examples

```r
## Experiments design with the "Basic Monte-Carlo" method for the "Ishigami" model

# Example I: by using the class constructors: mtkBasicMonteCarloDesigner()

# 1) Create a designer process based on the Basic Monte-Carlo method
MCdesign <- mtkBasicMonteCarloDesigner(listParameters = list(size=20))

# 2) Import the input factors of the "Ishigami" model
data(Ishigami.factors)

# 3) Build and run the workflow
exp1 <- mtkExpWorkflow(expFactors = Ishigami.factors,
                        processesVector = c(design=MCdesign))
run(exp1)

# 4) Report and plot the design
show(exp1)
plot(exp1)

# Example II: by using the class constructors: mtkNativeDesigner()

# 1) Create a designer process based on the Basic Monte-Carlo method
MCdesign <- mtkNativeDesigner("BasicMonteCarlo", information = list(size=20))

# 2) Import the input factors of the "Ishigami" model
data(Ishigami.factors)

# 3) Build and run the workflow
exp1 <- mtkExpWorkflow(expFactors = Ishigami.factors,
                        processesVector = c(design=MCdesign))
run(exp1)

# 4) Print and plot the design
print(exp1)
plot(exp1)
```

---

**deleteProcess-methods**  
*The deleteProcess method*

Description

Deletes a process from the workflow.

Usage

```r
deleteProcess(this, name)
```
deleteProcess-methods

Arguments

this an object of the class mtkExpWorkflow.
name a string from "design", "evaluate", or "analyze" to specify the process to delete.

Value

invisible()

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# Create an analysis for the Ishigami model:

x1 <- make.mtkFactor(name="x1", distribName="unif", distribPara=list(min=pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif", distribPara=list(min=pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif", distribPara=list(min=pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

designer <- mtkNativeDesigner("BasicMonteCarlo", 
information=list(size=20))
model <- mtkNativeEvaluator("Ishigami")
analyser <- mtkNativeAnalyzer("Regression", information=list(nboot=20))
ishiReg <- mtkExpWorkflow( expFactors=ishi.factors, 
processesVector=c( design=designer, 
evaluate=model, 
analyze=analyser) 
)
run(ishiReg)
summary(ishiReg)

# Delete the analysis process from the workflow and 
# run only the model simulation:

deleteProcess(ishiReg, "analyze")
run(ishiReg)
extractData-methods

The extractData method

Description

Gets the results produced by the workflow as a data.frame.

Usage

extractData(this, name)

Arguments

this       an object of the class mtkExpWorkflow.
name       a vector of strings from "design", "evaluate", or "analyze" to specify the results
to return. i.e. name =c("design") returns the experimental design produced by
the designer, name=c("design", "evaluate") returns both the experimental design
produced by the designer and the model simulation produced by the evaluator.

Value

a data.frame

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References

l’exploration numérique des modèles. In: Analyse de sensibilité et exploration de modèles : Ap-
application aux sciences de la nature et de l’environnement (R. Faivre, B. Iooss, S. Mahévas, D.

Examples

# Build a workflow for sensitivity analysis with the model "Ishigami"

x1 <- make.mtkFactor(name="x1", distribName="unif",
  distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
  distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
  distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

summary(ishiReg)
Fast

Description

A mtk compliant implementation of the so-called extended=FAST or e-Fast method for experiments design and sensitivity analysis.

Usage

- mtkFastDesigner(listParameters = NULL)
- mtkNativeDesigner(design="Fast", information=NULL)
- mtkFastAnalyser()
- mtkNativeAnalyser(analyze="Fast", information=NULL)

Parameters used to manage the sampling method

n: (numeric) the number of iteration.

Parameters used to manage the analysis method

No parameter is necessary.

Details

1. The mtk implementation uses the fast99 function of the sensitivity package. For further details on the arguments and the behaviour, see help(fast99, sensitivity).
2. The mtk implementation of the Fast method includes the following classes:
**mtkFastDesigner**: for Fast design processes.
**mtkFastAnalyser**: for Fast analysis processes.
**mtkFastDesignerResult**: to store and manage the design.
**mtkFastAnalyserResult**: to store and manage the analysis results.

3. Many ways to create a Fast designer are available in mtk, but we recommend the following class constructors: `mtkFastDesigner` or `mtkNativeDesigner`.

4. Many ways to create a Fast analyser are available in mtk, but we recommend the following class constructors: `mtkFastAnalyser` or `mtkNativeAnalyser`.

5. The method `Fast` is usually used both to build the experiment design and to carry out the sensitivity analysis. In such case, we can use the `mtkDefaultAnalyser` instead of naming explicitly the method for sensitivity analysis (see example III in the examples section)

**References**


**See Also**

`help(fast99, sensitivity)`

**Examples**

```r
## Sensitivity analysis of the "Ishigami" model with the "Fast" method

# Example I: by using the class constructors: mtkFastDesigner() and mtkFastAnalyser()

# Input the factors
data(Ishigami.factors)

# Build the processes and workflow:

# 1) the design process
expl1.designer <- mtkFastDesigner(listParameters = list(n=1000))

# 2) the simulation process
expl1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process
expl1.analyser <- mtkFastAnalyser()

# 4) the workflow
expl1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
```
processesVector = c(design=exp1.designer, 
evaluate=exp1.evaluator, analyze=exp1.analyser))

# Run the workflow and reports the results. 
run(exp1) 
print(exp1) 
plot(exp1)

## Example II: by using the class constructors: mtkNativeDesigner() and mtkFastAnalyser()

# Generate the factors 
data(Ishigami.factors)

# Build the processes and workflow:

# 1) the design process 
exp1.designer <- mtkNativeDesigner(design = "Fast", information=list(n=1000))

# 2) the simulation process 
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process with the default method 
exp1.analyser <- mtkFastAnalyser()

# 4) the workflow 
exp1 <- mtkExpWorkflow(expFactors = Ishigami.factors, 
procesesVector = c(design=exp1.designer, 
evaluate=exp1.evaluator, analyze=exp1.analyser))

# Run the workflow and reports the results. 
run(exp1) 
plot(exp1)

## Example III: by using the class constructors: mtkFastDesigner() and mtkDefaultAnalyser()

# Generate the factors 
data(Ishigami.factors)

# Build the processes and workflow:

# 1) the design process 
exp1.designer <- mtkFastDesigner( listParameters = list(n=2000))

# 2) the simulation process 
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process with the default method 
exp1.analyser <- mtkDefaultAnalyser()

# 4) the workflow
getData-methods

```r
exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
                        processesVector = c(design=exp1.designer,
                                             evaluate=exp1.evaluator, analyze=exp1.analyzer))

# Run the workflow and reports the results.
run(exp1)
plot(exp1)
```

---

### The `getData` method

**Description**

Returns the results produced by the process as a data.frame.

**Usage**

```r
dataFromThis <- getData(this)
```

**Arguments**

- `this` an object of the class `mtkProcess` or its sub-classes

**Value**

a data.frame.

**Author(s)**

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

**References**


**Examples**

```r
## Example: Sensitivity analysis for the Ishigami model

# Define the factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
                     distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
                     distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
                     distribPara=list(min=-pi, max=pi))
```
getDiscreteDistributionLevels-methods

The getDiscreteDistributionLevels method

description

Returns the levels of the discrete distribution associated with the factor's domain.

Usage

getDiscreteDistributionLevels(this)

Arguments

this the underlying object of the class to proceed (mtkFactor).

Value

a list.

Author(s)

JuHui WANG, MIA-jouy, INRA
Examples

# Create a discrete domain
x1 <- make.mtkFactor(name="x1", distribName="discrete",
  distribPara= list(type='categorical',
  levels = c(1,2,3,4,5), weights=rep(0.2, 5)))

# Returns the levels of the associated discrete distribution
getDiscreteDistributionLevels(x1)

getDiscreteDistributionType-methods

The getDiscreteDistributionType method

Description

Returns the type of the discrete distribution associated with the factor’s domain.

Usage

getDiscreteDistributionType(this)

Arguments

this the underlying object of the class to proceed (mtkFactor).

Value

a string.

Author(s)

Juhui WANG, MIA-jouy, INRA

Examples

# Create a discrete domain
x1 <- make.mtkFactor(name="x1", distribName="discrete",
  distribPara= list(type='categorical',
  levels = c(1,2,3,4,5), weights=rep(0.2, 5)))

# Returns the type of the associated discrete distribution
getDiscreteDistributionType(x1)
The `getDiscreteDistributionWeights` method

**Description**

Returns the weights of the discrete distribution associated with the factor's domain.

**Usage**

```r
getDiscreteDistributionWeights(this)
```

**Arguments**

- `this` the underlying object of the class to proceed (`mtkFactor`).

**Value**

a list of numeric values.

**Author(s)**

Juhui WANG, MIA-jouy, INRA

**Examples**

```r
# Create a discrete domain
x1 <- make.mtkFactor(name="x1", distribName="discrete",
                   distribPara=list(type='categorical',
                                    levels = c(1,2,3,4,5), weights=rep(0.2, 5)))
# Returns the weights of the associated discrete distribution
getDiscreteDistributionWeights(x1)
```

The `getDistributionName` method

**Description**

Returns the name of the distribution associated with a domain or a factor.
getDistributionNames-methods

Usage

getDistributionName(this)

Arguments

this the underlying object of the class to proceed (mtkDomain or mtkFactor).

Value

a string.

Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

Examples

# Create a domain and get the name of its distribution
d <- mtkDomain(distributionName="unif", domainNominalValue=0)
distribution <- getDistributionName(d)

# For more information, see examples for the mtkDomain or
# mtkFactor classes.

getDistributionNames-methods

The getDistributionNames method

Description

Returns the names of the distributions associated with an object of the class mtkExpFactors.

Usage

getDistributionNames(this)

Arguments

this an object of the mtkExpFactors class.

Value

a list.
Author(s)
Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

Examples

```r
# Define three factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
                   distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
                   distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
                   distribPara=list(min=-pi, max=pi))

# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# Get the names of the distributions managed by all the factors
names <- getDistributionNames(ishi.factors)
```

Description
Returns the nominal value associated with the uncertainty domain of a factor.

Usage
getDistributionNominalValue(this)

Arguments

this  
an object of the class mtkFactor.

Value

ANY

Author(s)
Juhui WANG, MIA-jouy, INRA
getDistributionNominalValues-methods

Examples

# Create a factor with a nominal value

x1 <- make.mtkFactor(name="x1", type='numeric', nominal=0.0, distribName="unif", distribPara=list(min=-pi, max=pi))

getAddressNominalValue(x1)

getDistributionNominalValues-methods

The `getDistributionNominalValues` method

Description

Gets the nominal values associated with the managed factors.

Usage

`getDistributionNominalValues(this)`

Arguments

- `this` an object of the class `mtkExpFactors`

Value

a named list

Author(s)

Juhui WANG, MIA-jouy, INRA

Examples

# Define three factors
x1 <- make.mtkFactor(name="x1", distribName="unif", distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif", distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif", distribPara=list(min=-pi, max=pi))

# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# Return the nominal values
nValues <- getDistributionNominalValues(ishi.factors)
The `getDistributionNominalValueTypes` method

**Description**

Returns the nominal values associated with the managed factors.

**Usage**

`getDistributionNominalValueTypes(this)`

**Arguments**

- `this` an object of the class `mtkFactor`.

**Value**

string

**Author(s)**

Juhui WANG, MIA-jo, INRA

**Examples**

```r
# Create a factor with a nominal value
x1 <- make.mtkFactor(name="x1", type='numeric', nominal=0.0, distribName="unif",
                      distribPara=list(min=-pi, max=pi))
getDistributionNominalValueTypes(x1)
```

The `getDistributionNominalValueTypes` method

**Description**

Gets the nominal values associated with the managed factors.

**Usage**

`getDistributionNominalValueTypes(this)`
The `getDistributionParameters` method

Description

Gets the parameters of the distribution(s) associated with an object (`mtkDomain`, `mtkFactor` or `mtkExpFactors`).

Usage

```r
getDistributionParameters(this)
```

Arguments

- `this` an object of the underlying class (`mtkDomain`, `mtkFactor` or `mtkExpFactors`)

Value

- a named list or a nested list

Examples

```r
# Define three factors
x1 <- make.mtkFactor(name="x1", distribName="unif", distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif", distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif", distribPara=list(min=-pi, max=pi))

# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# Return the nominal values
nTypes <- getDistributionNominalValueTypes(ishi.factors)
```
Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

Examples

# Define three factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
                      distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
                      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
                      distribPara=list(min=-pi, max=pi))

# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# Return the parameters of the distributions managed by all the factors as a nested list
names <- getDistributionParameters(ishi.factors)

getDomain-methods The getDomain method

Description

Returns the domain associated with the factor.

Usage

getDomain(this)

Arguments

this an object of the class mtkFactor.

Value

an object of the class mtkDomain

Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA
Examples

# Define a factor
x1 <- make.mtkFactor(name="x1", distribName="unif",
  distribPara=list(min=-pi, max=pi))

# Return the uncertainty domain associated with the factor
dom <- getDomain(x1)

getFactorFeatures-methods

Description

Returns the features associated with the managed factors.

Usage

getFactorFeatures(this)

Arguments

this

an object of the mtkExpFactors class

Value

a named list.

Author(s)

Juhui WANG, MIA-jouy, INRA

Examples

# Define three factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
  distribPara=list(min=-pi, max=pi))

# Define a list of features and associate it with the factor x1
features <- make.mtkFeatureList(list(pre=5, post=60))
setFeatures(x1, features)

x2 <- make.mtkFactor(name="x2", distribName="unif",
  distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
  distribPara=list(min=-pi, max=pi))

# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors(list(x1,x2,x3))
# Get the features of the managed factors as a list
factors <- getFactorFeatures(ishi.factors)


getFactorNames-methods

The getFactorNames method

Description

Returns the name of the managed factors.

Usage

gETCHFactorNames(this)

Arguments

this 

an object of the class mtkExpFactors.

Value

a list of strings

Author(s)

Juhui WANG, MIA-jouy, INRA

Examples

# Define three factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
                    distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
                    distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
                    distribPara=list(min=-pi, max=pi))

# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors(list(x1, x2, x3))

# Get the names of the factors managed by all the factors
factors <- getFactorNames(ishi.factors)
getFactors-methods

getFactors-methods

The getFactors method

Description

Retunrs the managed factors.

Usage

getFactors(this)

Arguments

this the underlying object of the class mtkExpFactors.

Value

a list of objects from the class mtkFactor.

Author(s)

Juhui WANG, MIA-jouy, INRA

Examples

# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors()

# Define the factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
                      distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
                      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
                      distribPara=list(min=-pi, max=pi))

# Assign and return the factors to the mtkExpFactors' object

setFactors(ishi.factors, list(x1,x2,x3))
getFactors(ishi.factors)
getFeatures-methods The getFeatures method

Description

Returns the features associated with the underlying factor.

Usage

getFeatures(this)

Arguments

this an object of the mtkFactor class

Value

a named list.

Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

Examples

# Define a factor
x1 <- make.mtkFactor(name="x1", distribName="unif", distribPara=list(min=-pi, max=pi))

# Define a list of features and associate it with the factor
features <- make.mtkFeatureList(list(pre=5, post=60))
setFeatures(x1, features)

# Return the features associated with the factor
f1 <- getFeatures(x1)

getLevels-methods The getLevels method

Description

Returns the levels associated with a discrete domain.

Usage

getLevels(this)
getMTKFeatures-methods

Arguments

this an object of the class mtkDomain or mtkLevels.

Value

a list

Examples

```r
l <- mtkLevels(type='categorical', levels=seq(1:10), weight=rep(0.1, 10))
getLevels(l)
```

getMTKFeatures-methods

*The getMTKFeatures method*

Description

Returns the features associated with the underlying factor as a list of mtkFeature objects.

Usage

getMTKFeatures(this)

Arguments

this an object of the mtkFactor class

Value

a list of objects of the class mtkFeature

Author(s)

Juhui WANG, MIA-jouy, INRA

Examples

```r
# Define a factor
x1 <- make.mtkFactor(name="x1", distribName="unif",
    distrPara=list(min=-pi, max=pi))

# Define a list of features and associate it with the factor
features <- make.mtkFeatureList(list(pre=5, post=60))
setFeatures(x1, features)

# Return the features associated with the factor
fl <- getMTKFeatures(x1)
```
### Description

Returns the name of the object or a process.

### Usage

```r
getName(this)
```

### Arguments

- `this` : the underlying object to proceed.

### Value

a string

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### Examples

```r
# just a method to access to the name of the underlying object or process
# Create an object of the 'mtkFeature' class.
f <- mtkFeature(name="x", type="double", val=0.0)
getName(f) # gives 'x'
```
getNominalValue-methods

Value

a list of strings

Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

Examples

# Define three factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
                      distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
                      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
                      distribPara=list(min=-pi, max=pi))

# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# Get the names of the factors managed by all the factors
factors <- getNames(ishi.factors)

getNominalValue-methods

The getNominalValue method

Description

Returns the nominal value associated with the uncertainty domain of a factor.

Usage

getNominalValue(this)

Arguments

this an object of the class mtkDomain.

Value

ANY

Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA
getNominalValueType-methods

Description

Returns the data type of the nominal value associated with the uncertainty domain of a factor.

Usage

getNominalValueType(this)

Arguments

this

an object of the class mtkDomain.

Value

a string

Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

Examples

# Create a domain and get the type of its nominal value
d <- mtkDomain(distributionName="unif", domainNominalValue=0.0)
mv <- getNominalValue(d)

# For more information, see examples for the mtkDomain or
# mtkFactor classes.
getParameters-methods  The getParameters method

Description

Returns the vector of parameters and converts it to a named list.

Usage

getParameters(this)

Arguments

this  

the underlying object of class mtkProcess or its sub-classes.

Value

a named list in which each element corresponds to a parameter. The vector of parameters is converted into a named list such as (name of parameter 1 = value of parameter 1, name of parameter 2 = value of parameter 2, ...).

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# Create a native designer avec the method "Morris"
# implemented in the package "mtk"

designer <- mtkNativeDesigner(design="Morris", information=list(size=20))

# Return the parameters as named list
getParameters(designer)
getProcess-methods

The getProcess method

Description

Gets a process from the workflow.

Usage

getProcess(this, name)

Arguments

this    the underlying object of class mtkExpWorkflow.
name    a string from "design", "evaluate", or "analyze" to specify the process to fetch.

Value

an object of the class mtkProcess.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# Build a workflow to do the sensitivity analysis for the model "Ishigami"
x1 <- make.mtkFactor(name="x1", distribName="unif",
        distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
        distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
        distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

designer <- mtkNativeDesigner("BasicMonteCarlo",
        information=list(size=20))
model <- mtkNativeEvaluator("Ishigami")
analyser <- mtkNativeAnalyser("Regression", information=list(nboot=20) )
ishireg <- mtkExpWorkflow(expFactors=ishi.factors, 
  processesVector=c( design=designer, 
    evaluate=model, 
    analyze=analyser) 
  )
run(ishiReg)

# Extract the process "design" or "evaluate" from the workflow for other uses
designer <- getProcess(ishiReg, "design")
evaluator <- getProcess(ishiReg, "evaluate")

---

**getResult-methods**  
*The getResult method*

**Description**

Returns the results produced by the process as an object of the class `mtkResult` or its sub-classes.

**Usage**

```r
getResult(this)
```

**Arguments**

- **this**  
  the underlying object of class `mtkProcess` or its sub-classes

**Details**

1. Sub-class of the class `mtkProcess` returns objects of different sub-class of the class `mtkResult`.  
   For instance, an object of the class `mtkDesigner` returns an object of the class `mtkDesignerResult`.
2. To fetch the results as a data.frame, please use the method `getData()`.

**Value**

an object of the class `mtkResult`.

**Author(s)**

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

**References**

Examples

# Create a designer and an analyser avec the method "Morris"
# to analyze the model "Ishigami":

# Specify the factors to analyze:
x1 <- make.mtkFactor(name="x1", distribName="unif",
distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
distribPara=list(min=-pi, max=pi))

factors <- mtkExpFactors(list(x1,x2,x3))

# Builds the processes:
#  1) the experimental design process with the method "Morris".
expl.designer <- mtkNativeDesigner(design="Morris",
    information=list(r=20,type="oat",levels=4,grid.jump=2))

#  2) the model simulation process with the model "Ishigami".
expl.evaluator <- mtkNativeEvaluator(model="Ishigami")

#  3) the analysis process with the default method.
#    Here, it is the Morris method.
expl.analyser <- mtkDefaultAnalyser()

# Build the workflow with the processes defined previously.
expl <- mtkExpWorkflow(expFactors=factors,
    processesVector = c(design=expl.designer,
    evaluate=expl.evaluator, analyze=expl.analyser))

# Run the workflow and report the results.
run(expl)

# Extracts the results produced by the analysis process as an objet of the class mtkAnalyserResult.
getResult(getProcess(expl, "analyze"))

---

**getDescription**

The `getDescription` method

Description

Returns a string indicating the data type associated with the underlying object.

Usage

gtype(this)
### getValue-methods

**Arguments**

- **this**: an object of the underlying class.

**Value**

- a string

**Author(s)**

Juhui WANG, MIA-jouy, INRA

**Examples**

```r
# Define a factor
x1 <- make.mtkFactor(name="x1", distribName="unif",
                      distribPara=list(min=-pi, max=pi))

# Return the data-type associated with the factor
t <- getClass(x1)

# Create an object of the 'mtkFeature' class.
f <- mtkFeature(name="x", type="double", val=0.0)

# Return the data-type associated with the feature
getClass(f) # gives 'double'
```

---

**Description**

Returns the name and the value managed by an object of the underlying class.

**Usage**

```r
getValue(this)
```

**Arguments**

- **this**: an object of the class `mtkValue` or its sub-classes.

**Value**

- a named variable
Author(s)
Juhui WANG, MIA-jouy, INRA

Examples

# Create an object of the 'mtkValue'
v <- mtkValue(name="x", type="string", va=2.2)

# Fetch the value of the object as a named variable: x = "2.2"
getValue(v)

getWeights-methods The getWeights method

Description
Returns the weights of the discrete distribution associated with the factor's domain.

Usage
getWeights(this)

Arguments
this the underlying object of the class to proceed (mtkLevels and mtkDomain).

Value
a list of numeric values.

Author(s)
Juhui WANG, MIA-jouy, INRA

Examples

# Create a discrete domain
x1 <- mtkDomain(distributionName="discrete", domainNominalValue=0,
distributionParameters=list(type='categorical',
levels = c(1,2,3,4,5), weights=rep(0.2, 5)))

# Returns the weights of the associated discrete distribution
getWeights(x1)
is.finished-methods

The is.finished method

Description
Tests if the process has run and the results produced by the process are available.

Usage
is.finished(this)

Arguments
this the underlying object of the class mtkProcess

Value
TRUE or FALSE.

Author(s)

References

Examples

# Build a workflow to do the sensitivity analysis for the model "Ishigami"
x1 <- make.mtkFactor(name="x1", distribName="unif",
distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1, x2, x3))

designer <- mtkNativeDesigner("BasicMonteCarlo",
information=list(size=20))

ishiReg <- mtkExpWorkflow(expFactors=ishi.factors,
 processesVector=c(design=designer) )
run(ishiReg)

# Extract the process "design" and test if it is correctly executed.

designer <- getProcess(ishiReg, "design")
is.finished(designer)

---

### is.ready-methods

#### The is.ready method

**Description**

Tests if the process is ready to run.

**Usage**

```r
is.ready(this)
```

**Arguments**

- `this` the underlying object of the class `mtkProcess`

**Value**

TRUE or FALSE.

**Author(s)**

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

**References**


**Examples**

```r
## This method is usually used only for the package's core programming!!

# creates an experimental design with the method "Morris"
# to analyze the model "Ishigami":

# Specify the factors to analyze:

x1 <- make.mtkFactor(name="x1", distribName="unif",
                      distribPara=list(min=-pi, max=pi))
```
Ishigami

Description

The Ishigami model is an example evaluator implemented in the native mtk. It corresponds to the Ishigami function described in Saltelli et al., 2000. The behavior of the model is influenced by three factors \(x_1, x_2, x_3\).

Usage

- \texttt{mtkIshigamiEvaluator()}
- \texttt{mtkNativeEvaluator(model="Ishigami")}
- \texttt{mtkEvaluator(protocol = "R", site = "mtk", service = "Ishigami")}

Details

1. The implementation of the Ishigami model includes the object \texttt{Ishigami.factors} on the input factors and the class \texttt{mtkIshigamiEvaluator} to run the simulations.
2. In mtk, there are a few ways to build an evaluator of the Ishigami model, but we usually recommend the following class constructors: \texttt{mtkIshigamiEvaluator}, \texttt{mtkNativeEvaluator}.

References

See Also

help(Ishigami.factors), help(ishigami.fun, sensitivity)

Examples

```r
### Run simulations of the "Ishigami" model
### for a random sample of input combinations

## Example I: by using the class constructor: mtkIshigamiEvaluator()

#
# Input the factors used in the "Ishigami" model
data(Ishigami.factors)

# Build the workflow:
# 1) specify the design process
exp1.designer <- mtkNativeDesigner(design = "BasicMonteCarlo",
                                   information = list(size=20) )

# 2) specify the evaluation process;
exp1.evaluator <- mtkIshigamiEvaluator()

# 3) specify the workflow
exp1 <- mtkExpWorkflow(expFactors = Ishigami.factors,
                        processesVector = c(design=exp1.designer,
                                             evaluate=exp1.evaluator) )

# Run the workflow and report the results.
run(exp1)
print(exp1)

## Example II: by using the class constructor: mtkNativeEvaluator()

# Generate the Ishigami input factors
data(Ishigami.factors)

# Build the workflow:
# 1) specify the design process
exp1.designer <- mtkNativeDesigner(design = "BasicMonteCarlo",
                                   information = list(size=20) )

# 2) specify the evaluation process;
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) specify the workflow
exp1 <- mtkExpWorkflow(expFactors = Ishigami.factors,
                        processesVector = c(design=exp1.designer,
                                             evaluate=exp1.evaluator) )

# Run the workflow and report the results.
run(exp1)
print(exp1)
```
## Example III: by using the generic class constructor: mtkEvaluator()

# Generate the Ishigami input factors
data(Ishigami.factors)

# Build the workflow:
# 1) specify the design process
exp1.designer <- mtkNativeDesigner(
  design = "BasicMonteCarlo", information = list(size=20) )

# 2) specify the evaluation process;
exp1.evaluator <- mtkEvaluator(protocol = "R", site = "mtk", service = "Ishigami")

# 3) specify the workflow
exp1 <- mtkExpWorkflow(expFactors = Ishigami.factors,
  processesVector = c/design=exp1.designer, evaluate=exp1.evaluator) )
# Run the workflow and report the results.
run(exp1)
print(exp1)

---

**Ishigami.factors**  
*Input factors of the Ishigami model*

### Description

The names and uncertainty distributions of the 3 input factors $x_1$, $x_2$, $x_3$ involved in the Ishigami function which is usually used as a model example for uncertainty and sensitivity analysis methods.

### Usage

data(Ishigami.factors)

### Format

an object of class `mtkExpFactors`.

### References


### See Also

help(Ishigami), help(ishigami.fun,sensitivity)
Examples

# The code used to generate the Ishigami.factors is as follows:

x1 <- make.mtkFactor(name="x1", distribName="unif",
  distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
  distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
  distribPara=list(min=-pi, max=pi))

Ishigami.factors <- mtkExpFactors(list(x1,x2,x3))

# To import the Ishigami.factors, just use the following line
data(Ishigami.factors)

---

**make.mtkFactor**

*The make.mtkFactor function*

**Description**

Creates a new input factor and specifies its uncertainty distribution.

**Usage**

```r
make.mtkFactor(name="unknown", id="unknown", unit="", type="",
  nominal=NA, distribName='unknown', distribPara=list(), features=list())
```

**Arguments**

- `name`: the name of the input factor.
- `id`: the name of the factor in the simulation code, if different from `name` (optional).
- `unit`: the measurement unit of the factor values (optional). This can be used in graphics or reports, for example.
- `type`: the data-type of the factor's values (optional).
- `nominal`: the nominal value of the factor.
- `distribName`: the name of the probability distribution describing the factor's uncertainty.
- `distribPara`: the list of distribution parameters.
- `features`: the list of factor's features.

**Details**

The `distribName` argument must use the R terminology, for example `norm` for the normal distribution or `unif` for the uniform one; see `help(distributions)`.
Value

an object of class `mtkFactor`.

Author(s)

Juhui WANG, MIA-jouy, INRA, Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod

Examples

```r
# Define a new continuous factor
make.mtkFactor("A", distribName="unif", distribPara=list(min=0,max=1))
# Define a new discrete factor
make.mtkFactor("D", distribName="discrete", distribPara =
  list(type='categorical', levels=c('a','b','c'),
    weights=rep(0.33,3))
)
```

make.mtkFeatureList  The `make.mtkFeatureList` function

Description

Creates a list of `mtkFeature` elements from a simple named list.

Usage

```r
make.mtkFeatureList(x=list())
```

Arguments

- **x**: a named list.

Value

a list of objects from the class `mtkFeature`.

Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

Examples

```r
# Create a list of mtkFeature for the Features: min, max, shape.
make.mtkFeatureList(list(min=-1,max=+1,shape="square"))
```
### make.mtkParameterList

*The make.mtkParameterList function*

**Description**

Creates a list of `mtkParameter` elements from a simple named list.

**Usage**

```r
make.mtkParameterList(x=list())
```

**Arguments**

- `x` a named list.

**Value**

a list of objects from the class `mtkParameter`.

**Author(s)**

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

**Examples**

```r
# Create a list of mtkParameter from a named list for the parameters: min, max, shape.
make.mtkParameterList(list(min=-1,max=+1,shape="hello"))
```

### Morris

*The Morris method*

**Description**

A `mtk` compliant implementation of the `morris` method for experiments design and sensitivity analysis.

**Usage**

- `mtkMorrisDesigner(listParameters = NULL)`
- `mtkNativeDesigner(design="Morris", information=NULL)`
- `mtkMorrisAnalyser(listParameters = NULL)`
- `mtkNativeAnalyser(analyze="Morris", information=NULL)`
Parameters

- **r**: the number of trajectories or a pair \((r_1, r_2)\) if the version due to Campolongo et al. (2007) is used.
- **type**: the type of design (either `oat` or `simplex`).
- **levels**: the number of levels per factor (if `type` = "oat").
- **grid.jump**: the length of the steps within the trajectories (if `type` = "oat").
- **scale.factor**: a numeric value, the homothety factor of the (isometric) simplexes (if `type` = "simplex").
- **scale**: logical. If `TRUE`, the input design of experiments is scaled before computing the elementary effects so that all factors vary within the range \([0, 1]\).
- **shrink**: a scalar or a vector of scalars between 0 and 1, specifying shrinkage to be used on the probabilities before calculating the quantiles.

Details

1. The `mtk` implementation uses the `morris` function of the `sensitivity` package. For further details on the arguments and the behavior, see `help(morris, sensitivity)`.
2. The `mtk` implementation of the Morris method includes the following classes:
   - `mtkMorrisDesigner`: for the Morris design processes.
   - `mtkMorrisAnalyser`: for Morris analysis processes.
   - `mtkMorrisDesignerResult`: to store and manage the design.
   - `mtkMorrisAnalyserResult`: to store and manage the analysis results.
3. Many ways to create a Morris designer are available in `mtk`, but we recommend the following class constructors: `mtkMorrisDesigner` or `mtkNativeDesigner`.
4. Many ways to create a Morris analyser are available in `mtk`, but we recommend the following class constructors: `mtkMorrisAnalyser` or `mtkNativeAnalyser`.
5. The method Morris is usually used both to build the experiment design and to carry out the sensitivity analysis. In such case, we can use the `mtkDefaultAnalyser` instead of naming explicitly the method for sensitivity analysis (see example III in the examples section)

References


See Also

`help(morris, sensitivity)`
Examples

## Sensitivity analysis of the "Ishigami" model with the "Morris" method

# Example I: by using the class constructors: mtkMorrisDesigner() and mtkMorrisAnalyser()

# Generate the factors
data(Ishigami.factors)

# Build the processes and workflow:

# 1) the design process
expl1.designer <- mtkMorrisDesigner(
    listParameters = list(r=20, type="oat",
        levels=4, grid.jump=2)
)

# 2) the simulation process
expl1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process
expl1.analyser <- mtkMorrisAnalyser()

# 4) the workflow
expl1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
    processesVector = c(design=expl1.designer,
        evaluate=expl1.evaluator,
        analyze=expl1.analyser)
)

# Run the workflow and reports the results.
run(expl1)
print(expl1)
plot(expl1)
# plot3d.morris(extractData(expl1, name="analyze"))

## Example II: by using the class constructors: mtkNativeDesigner() and mtkMorrisAnalyser()

# Generate the factors
data(Ishigami.factors)

# Build the processes and workflow:

# 1) the design process
expl1.designer <- mtkNativeDesigner(design = "Morris",
    information = list(r=20, type="oat",
        levels=4, grid.jump=2)
)

# 2) the simulation process
expl1.evaluator <- mtkNativeEvaluator(model="Ishigami")
# 3) the analysis process with the default method
exp1.analyser <- mtkMorrisAnalyser()

# 4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
                        processesVector = c(design=exp1.designer,
                                             evaluate=exp1.evaluator,
                                             analyze=exp1.analyser)
                      )

# Run the workflow and reports the results.
run(exp1)
print(exp1)

## Example III: by using the class constructors: mtkMorrisDesigner() and mtkDefaultAnalyser()

# Generate the factors
data(Ishigami.factors)

# Build the processes and workflow:

# 1) the design process
exp1.designer <- mtkMorrisDesigner(listParameters =
                               list(r=20, type="oat",
                                    levels=4, grid.jump=2))

# 2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process with the default method
exp1.analyser <- mtkDefaultAnalyser()

# 4) the workflow

exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
                        processesVector = c(design=exp1.designer,
                                             evaluate=exp1.evaluator,
                                             analyze=exp1.analyser)
                        )

# Run the workflow and reports the results.
run(exp1)
print(exp1)
mtk.analyserAddons

Description
A function used to extend the "mtk" package with new analysis methods programmed as R functions. The mtk.analyserAddons function takes a R file as input and converts it into a mtk compliant class which can be seamlessly integrated into the mtk package.

Usage
mtk.analyserAddons(where = NULL, library = NULL,
    authors = NULL, name = NULL,
    main = NULL,
    summary = NULL,
    plot = NULL,
    print = NULL)

Arguments
where
    NULL or a file holding the R function to convert.
library
    NULL or the name of the library if the R function to convert is held in a library.
authors
    NULL or information about the authors of the R function.
name
    a string to name the method when used with the "mtk" package.
main
    the R function which implements the method.
summary
    NULL or a subversion of the summary function provided with the method.
plot
    NULL or a reprogrammed version of the plot function provided with the method.
print
    NULL or a reprogrammed version of the print function provided with the method.

Details
The new method must be programmed according to the following syntax:
main <- function(X, Y, ...) where X is a data.frame holding the experiment design, and Y is a data.frame holding the results produced by the model simulation.
The function main returns a named list with two elements: main and information. The element main holds the result of the sensitivity analysis and the element information is optional, may be used to give supplementary information about the analysis process and the produced results.
Furthermore, in order to report the analysis results more precisely, users can redefine the generic functions: summary(object, ...), plot(x, y, ...), print(x, ...).

Value
invisible()

Author(s)
Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr
mtk.analyserAddons

References


Examples

# an example implementation of the method "Regression"
# called here "RegressionTest" is held in the file
# "inst/extdata/regressionSI.R"

rFile <- "regressionSI.R"
rFile <- paste(path.package("mtk", quiet = TRUE),
"/extdata/", rFile, sep = "")

# to convert the method "RegressionTest" to S4 classes
# compliant with the "mtk" package. The generated "mtk" compliant class
# is called "mtkXXXAnalyser.R" where XXX corresponds to the name of the method.

mtk.analyserAddons(where=rFile, authors="H. Monod, INRA",
name="RegressionTest",
main="regressionSI", print="print.regressionSI",
plot="plot.regressionSI")

# To use the method "RegressionTest" with the package "mtk",
# just source the generated new files

source("mtkRegressionTestAnalyser.R")

## Use the method "RegressionTest" to do sensitivity analysis

# 1) Define the factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
 distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
 distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# 2) Create a workflow with the "Ishigami" model and analyze it with the new method
ishiReg <- mtkExperiment(expFactors=ishi.factors,
design="BasicMonteCarlo",designInfo=list(size=20),
model="Ishigami",
analyze="RegressionTest",
)

# 3) Run the workflow and report the results
run(ishiReg)
summary(ishiReg)
The `mtk.designerAddons` function

Description

A function used to extend the mtk package with new design methods programmed as R functions. The `mtk.designerAddons` function takes a R file as input and converts it into a mtk compliant class which can be seamlessly integrated into the mtk package.

Usage

```r
mtk.designerAddons(where = NULL, library = NULL,
authors = NULL, name = NULL,
main = NULL, summary = NULL,
plot = NULL, print = NULL)
```

Arguments

- `where` NULL or the file containing the R functions to convert into native mtk methods.
- `library` NULL or the name of the package if the R function to convert is included in a package.
- `authors` NULL or information about the authors of the R function.
- `name` a string to name the method when used with the mtk package.
- `main` the name of the R function implementing the designer.
- `summary` NULL or a special version of the summary function provided in the file where.
- `plot` NULL or a special version of the plot function provided in the file where.
- `print` NULL or a special version of the print function provided in the file where.

Details

The `main` function must have the following syntax:

```r
main <- function(factors, distribNames, distribParameters, ...)
```

where `factors` is either a number or a list of strings giving the names of the n input factors, `distribNames` is a list of string giving the names of the n probability distributions that describe the factors' uncertainty, and `distribParameters` is a list of n lists specifying the distribution parameters associated with the uncertainty domains.

The R function `main` returns a named list with two elements: the element `main` is a data.frame containing the generated experiment design and the element `information` is an optional list that may be used to provide complementary information about the design process and results.

Furthermore, in order to give more advanced data reporting mechanism with the new method, users can redefine the generic functions:

```r
summary(object, ...), plot(x,y, ...), print(x, ...)
```
mtk.designerAddons

Value

invisible()

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# an example implementation of the method "MC" is held in the file
# "inst/extdata/montecarloDesigner.R"

rFile <- "montecarloDesigner.R"
rFile <- paste(path.package("mtk"), quiet = TRUE),
"/extdata/",rFile,sep = "")

# to convert this special version of the method "MC"
# to 54 classes compliant with the "mtk" package. The generated "mtk" compliant class
# is called "mtkXXXDesigner.R" where XXX corresponds to the name of the method.
mtk.designerAddons(where=rFile, authors="H. Monod,INRA", name="MC",
main="basicMonteCarlo")

# to use the method "MC" with the package "mtk",
# just source the generated new files

source("mtkMCDesigner.R")

## Use the "mtkMCDesigner" with the "mtk" package in a seamless way:

# 1) Define the factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# 2) Specify a new workflow with the new method
ishiReg <- mtkExperiment(expFactors=ishi.factors,design="MC",
model="Ishigami", analyze="Regression",
designInfo=list(size=20))

# 3) Run the workflow and report the results
run(ishiReg)
mtk.evaluatorAddons

Description
A function used to extend the "mtk" package with new models programmed as R functions. The mtk.evaluatorAddons function takes a R file as input and converts it into a mtk compliant class which can be seamlessly integrated into the mtk package.

Usage
mtk.evaluatorAddons(where = NULL, library = NULL,
authors = NULL, name = NULL, main = NULL,
summary = NULL, plot = NULL,
print = NULL)

Arguments
where
NULL or a file holding the R function to convert.
library
NULL or the name of the library if the R function to convert is held in a library.
authors
NULL or information about the authors of the R function.
name
a string to name the model when used with the "mtk" package.
main
the R function which implements the model.
summary
NULL or a special version of the "summary" function provided with the model.
plot
NULL or a special version of the "plot" function provided with the model.
print
NULL or a special version of the "print" function provided with the model.

Details
The new model must be programmed according to the following syntax:
main <- function(X, ...) where X is a data.frame holding the experiment design used to run the model simulation.
The function main returns a named list with two elements: main and information. The element main holds the result of the model simulation and the element information is optional, may be used to give supplementary information about the simulation process and its results.
Furthermore, users can redefine the following generic functions to report the results more precisely:
summary (object, ...), plot(x,y, ...), print(x, ...).

Value
invisible()
Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# an example implementation of the model "WW" is held
# in the file "inst/extdata/wwdm.R"

rFile <- "wwdm.R"

rFile <- paste(path.package("mtk", quiet = TRUE),
"/extdata/", rFile, sep = ")

# to covert the model "WW" to a S4 classes compliant with the "mtk" package.
# The generated "mtk" compliant class is called "mtkXXXEvaluator.R" where XXX corresponds
# to the name of the model.

mtk.evaluatorAddons(where=rFile, authors="H. Monod,INRA", name="WW", main="wwdm.simule")

# to use the model evaluator "WW" with the package "mtk",
# just source the generated new files

generate("mtkWWEvaluator.R")

## Use the "mtkWWEvaluator" with the "mtk" package in a seamless way:

# 1) Define the factors

Eb <- make.mtkFactor(name="Eb", distribName="unif",
nominal=1.85, distribPara=list(min=0.9, max=2.8))

Eimax <- make.mtkFactor(name="Eimax", distribName="unif",
nominal=0.94, distribPara=list(min=0.9, max=0.99))

K <- make.mtkFactor(name="K", distribName="unif", nominal=0.7,
distribPara=list(min=0.6, max=0.8))

Lmax <- make.mtkFactor(name="Lmax", distribName="unif", nominal=7.5,
distribPara=list(min=3, max=12))

A <- make.mtkFactor(name="A", distribName="unif", nominal=0.0065,
distribPara=list(min=0.0035, max=0.01))

B <- make.mtkFactor(name="B", distribName="unif", nominal=0.00205,
distribPara=list(min=0.0011, max=0.0025))

TI <- make.mtkFactor(name="TI", distribName="unif", nominal=900,

distribPara=list(min=700, max=1100))

WW.factors <- mtkExpFactors(list(Eb,Eimax,K,Lmax,A,B,TD))

# 2) Build a workflow for the "WW" model
```r
exp <- mtkExperiment(expFactors=WW.factors,
design="Morris",designInfo=list(type="oat",
r=10, levels=5, grid.jump=3),
model="WW", modelInfo=list(year=3),
analyze="Morris", analyzeInfo=list(type="oat",
r=10, levels=5, grid.jump=3));

## 3) Run the workflow and reports the results

run(exp)
summary(exp)
```

---

### mtkAnalyser

**The constructor of the class** `mtkAnalyser`

#### Description

The constructor

#### Usage

```r
mtkAnalyser(protocol = "R", site = "mtk", service = ",
parameters= NULL, parametersList = NULL, ready = TRUE,
state = FALSE, result = NULL)
```

#### Arguments

- **protocol**
  a string from "http", "system", "R" respectively representing if the process is implemented remotely, locally or as R function.

- **site**
  the site where the process is implemented if remotely or the package where the process is implemented if as a R function.

- **service**
  a string corresponding to the name of the method implemented in the package "mtk" or the service that implements the process if remotely.

- **parameters**
  a vector of `mtkParameter` representing the parameters necessary to run the process.

- **parametersList**
  a named list containing the parameters necessary to run the process. It gives another way to specify the parameters.

- **ready**
  a logical to indicate if the process is ready to run.

- **state**
  a logical to indicate if the process finished running and the results are available.

- **result**
  an object of a class derived from `mtkAnalyserResult` to hold the results produced by the analyser.

#### Value

an object of the `mtkAnalyser` class
The mtkAnalyser class

Author(s)

References

Examples

```r
# Creates an analyser with the method "Morris" implemented
# in the package "mtk"
analyser <- mtkAnalyser(service="Morris",
parametersList=list(nboot=20))
```

Description

The mtkAnalyser class is a sub-class of the class mtkProcess used to manage the sensitivity analysis process. It provides all the slots and methods defined in the class mtkProcess.

Class Hierarchy

Parent classes: mtkProcess

Direct Known Subclasses: mtkNativeAnalyser, mtkMorrisAnalyser, etc.

Constructor

mtkAnalyser signature(protocol="R", site="mtk", service="", parameters=NULL, parametersList=NULL, ready=TRUE, state=FALSE, result=NULL)

Slots

name: (character) a string to name the processing type. Here, it always takes "analyze".
protocol: (character) a string to name the protocol used to run the process: http, system, R, etc.
site: (character) a string to indicate where the service is located.
service: (character) a string to name the method or the service (if remotely) to invoke.
parameters: (vector) a vector of mtkParameter containing the parameters to pass while calling the service.
ready: (logical) a logical to tell if the process is ready to run.
state: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.
result: (ANY) NULL or an object of the class mtkAnalyserResult to hold the results produced by the process

Methods

setName signature(this = "mtkAnalyser", name = "character"): Not used, just inherited from the parent class.
setParameters signature(this = "mtkAnalyser", f = "vector"): Assigns a new vector of parameters to the process.
getParameters signature(this = "mtkAnalyser"): Returns the parameters as a named list.
is.ready signature( = "mtkAnalyser"): Tests if the process is ready to run.
setReady signature(this = "mtkAnalyser", switch = "logical"): Makes the process ready to run.
is.ready signature( = "mtkAnalyser"): Tests if the results produced by the process are available.
setReady signature(this = "mtkAnalyser", switch = "logical"): Marks the process as already executed.
getResult signature(this = "mtkAnalyser"): Returns the results produced by the process as a mtkAnalyserResult.
getData signature(this = "mtkAnalyser"): Returns the results produced by the process as a data.frame.
serializeOn signature(this = "mtkAnalyser"): Returns all data managed by the process as a named list.
run signature(this = "mtkAnalyser", context= "mtkExpWorkflow"): Runs the sensitivity analysis on the model defined in the context.
summary signature(object = "mtkAnalyser"): Provides a summary of the results produced by the process.
print signature(x = "mtkAnalyser"): Prints a report of the results produced by the process.
plot signature(x = "mtkAnalyser"): Builds a plot of the results produced by the process.
report signature(this = "mtkAnalyser"): Reports the results produced by the process.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References

Examples

# Creates an analyser avec the method "Morris"
# implemented in the package "mtk".

analyser <- mtkAnalyser(service="Morris",
parametersList=list(nboot=20))

mtkAnalyserResult  

Description

The constructor

Usage

mtkAnalyserResult(main = data.frame(), information = list())

Arguments

main  
a data.frame to hold the results produced with the analyser.
information  
a named list containing optional information about the managed data and process.

Value

an object of the mtkAnalyserResult class

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# Construct an object of the \code{mtkAnalyserResult} class
# from a data.frame.
data <- data.frame()
result <- mtkAnalyserResult(main=data, information = list(method="Morris", model="Ishigami"))
The mtkAnalyserResult class

Description

A class to manage the results produced by the sensitivity analysis process.

Class Hierarchy

Parent classes: mtkResult

Direct Known Subclasses: mtkMorrisAnalyserResult, mtkPLMMAnalyserResult, etc.

Constructor

{mtkAnalyserResult} signature(main = data.frame(), information = list())

Slots

main: (data.frame) a data.frame to hold the analysis results produced with the analyser.

information: (list) a named list containing optional information about the managed data and process.

Methods

summary signature(object = "mtkAnalyserResult"): Provides a summary of the analysis results produced with the analyser.

print signature(x = "mtkAnalyserResult"): Prints a report of the analysis results produced with the analyser.

plot signature(x = "mtkAnalyserResult"): Plots the analysis results produced with the analyser.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References

### Examples

```r
# Construct an object of the \code{mtkAna\-lyser\-Result} class
# from a data.frame.
data <- data.frame()
result <- mtkAna\-lyser\-Result(main=data, information
= list(method="Morris", model="Ishigami"))
```

### Description

The constructor of the class `mtkBasicMonteCarloDesigne

### Usage

```r
mtkBasicMonteCarloDesigner(mtkParameters = NULL,
listParameters = NULL)
```

### Arguments

- `mtkParameters` a vector of `mtkParameter` representing the parameters necessary to run the process.
- `listParameters` a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

### Value

an object of the `mtkBasicMonteCarloDesigner` class

### Details

See the `BasicMonteCarlo` method with `help(BasicMonteCarlo)`

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

### References

mtkBasisMonteCarloDesigner-class

Examples

# see examples with help(BasicMonteCarlo)

Description

The mtkBasisMonteCarloDesigner class is a sub-class of the class mtkDesigner. It implements the BasicMonteCarlo method for experiments design and provides all the slots and methods defined in the class mtkDesigner.

Class Hierarchy

Parent classes: mtkDesigner

Direct Known Subclasses:

Constructor

mtkBasisMonteCarloDesigner signature(mtkParameters = NULL, listParameters = NULL)

Slots

name: (character) always takes the string "design".
protocol: (character) always takes the string "R".
site: (character) always takes the string "mtk".
service: (character) always takes the string "BasicMonteCarlo".
parameters: (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.
ready: (logical) a logical to tell if the process is ready to run.
state: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.
result: (ANY) a data holder to hold the results produced by the process

Methods

setName signature(this = "mtkBasisMonteCarloDesigner", name = "character"): Method inherited from the parent class.

setParameters signature(this = "mtkBasisMonteCarloDesigner", f = "vector"): Assigns new parameters to the process.
getParameters signature(this = "mtkBasicMonteCarloDesigner"): Returns the parameters as a named list.

is.ready signature( = "mtkBasicMonteCarloDesigner"): Tests if the process is ready to run.

setReady signature(this = "mtkBasicMonteCarloDesigner", switch = "logical"): Makes the process ready to run.

is.ready signature( = "mtkBasicMonteCarloDesigner"): Tests if the results produced by the process are available.

setReady signature(this = "mtkBasicMonteCarloDesigner", switch = "logical"): Marks the process as already executed.

getResult signature(this = "mtkBasicMonteCarloDesigner"): Returns the results produced by the process as a *mtkBasicMonteCarloDesignerResult*.

getData signature(this = "mtkBasicMonteCarloDesigner"): Returns the results produced by the process as a data.frame.

serializeOn signature(this = "mtkBasicMonteCarloDesigner"): Returns all data managed by the process as a named list.

run signature(this = "mtkBasicMonteCarloDesigner", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.

summary signature(object = "mtkBasicMonteCarloDesigner"): Provides a summary of the results produced by the process.

print signature(x = "mtkBasicMonteCarloDesigner"): Prints a report of the results produced by the process.

plot signature(x = "mtkBasicMonteCarloDesigner"): Plots the results produced by the process.

report signature(this = "mtkBasicMonteCarloDesigner"): Reports the results produced by the process.

Details

See the BasicMonteCarlo method with help(BasicMonteCarlo)

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# See examples from the BasicMonteCarlo method: help(basicMonteCarlo)
mtkBashicMonteCarloDesignerResult

The constructor of class mtkBasicMonteCarloDesignerResult

Description
The constructor

Usage
mtkBashicMonteCarloDesignerResult(main, information=NULL)

Arguments
main a data.frame holding the experimental design produced by the designer.
information a named list containing the information about the managed data and the underlying process.

Value
an object of the mtkBasicMonteCarloDesignerResult class

Author(s)
Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References

Examples

# see examples with help(BasicMonteCarlo)
**mtkBasicMonteCarloDesignerResult-class**

*The mtkBasicMonteCarloDesignerResult class*

**Description**

A class to collect the experimental design produced by the designer implementing the method `BasicMonteCarlo`.

**Class Hierarchy**

- **Parent classes**: `mtkDesignerResult`
- **Direct Known Subclasses**: 

**Constructor**

`mtkBasicMonteCarloDesignerResult` signature(main, information=NULL)

**Slots**

- `main`: (data.frame) a data-frame holding the experimental design.
- `information`: (list) a named list containing optional information about the managed data or the underlying process.

**Methods**

- `summary` signature(object = "mtkBasicMonteCarloDesignerResult"): Provides a summary of the experimental design produced by the designer.
- `print` signature(x = "mtkBasicMonteCarloDesignerResult"): Prints a report of the experimental design produced by the designer.
- `plot` signature(x = "mtkBasicMonteCarloDesignerResult"): Plots the experimental design produced by the designer.

**Author(s)**

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

**References**

mtkDefaultAnalyser

The constructor of the class mtkDefaultAnalyser

Description

This class is used when both the experimental design and the sensitivity analysis are fulfilled with the same method.

Usage

mtkDefaultAnalyser()

Value

an object of the mtkDefaultAnalyser class

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# creates a designer and an analyser with the method "Morris"
# to analyze the model "Ishigami":

# Specify the factors to analyze:
x1 <- make.mtkFactor(name="x1", distribName="unif",
distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
distribPara=list(min=-pi, max=pi))
factors <- mtkExpFactors(list(x1,x2,x3))

# Build the processes:
# 1) the experimental design process with the method "Morris".
expl.designer <- mtkNativeDesigner(design="Morris",
mtkDefaultAnalyser-class

Description

The mtkDefaultAnalyser class is a sub-class of the class mtkAnalyser. It provides all the slots and methods defined in the class mtkAnalyser. The mtkDefaultAnalyser class is used when the method used for the sensitivity analysis is the same as the method used for the experiment design.

Class Hierarchy

Parent classes: mtkAnalyser

Direct Known Subclasses:

Constructor

mtkDefaultAnalyser signature()

Slots

name: (character) always takes the string "analyze".
protocol: (character) a string to name the protocol used to run the process: http, system, R, etc.
site: (character) a string to indicate where the service is located.
service: (character) a string to name the service to invoke.
parameters: (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.
ready: (logical) a logical to tell if the process is ready to run.
state: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.
result: (ANY) a data holder to hold the results produced by the process.
**Methods**

- **setName** signature(this = "mtkDefaultAnalyser", name = "character"): Not used, method inherited from the parent class.

- **setParameters** signature(this = "mtkDefaultAnalyser", f = "vector"): Assigns new parameters to the process.

- **getParameters** signature(this = "mtkDefaultAnalyser"): Returns the parameters as a named list.

- **is.ready** signature( = "mtkDefaultAnalyser"): Tests if the process is ready to run.

- **setReady** signature(this = "mtkDefaultAnalyser", switch = "logical"): Makes the process ready to run.

- **is.ready** signature( = "mtkDefaultAnalyser"): Tests if the results produced by the process are available.

- **setReady** signature(this = "mtkDefaultAnalyser", switch = "logical"): Marks the process as already executed.

- **getResult** signature(this = "mtkDefaultAnalyser"): Returns the results produced by the process as a `mtkAnalyserResult`.

- **getDat** signature(this = "mtkDefaultAnalyser"): Returns the results produced by the process as a data.frame.

- **serializeOn** signature(this = "mtkDefaultAnalyser"): Returns all data managed by the process as a named list.

- **run** signature(this = "mtkDefaultAnalyser", context = "mtkExpWorkflow"): Runs the sensitivity analysis defined in the context.

- **summary** signature(object = "mtkDefaultAnalyser"): Provides a summary of the results produced by the process.

- **print** signature(x = "mtkDefaultAnalyser"): Prints a report of the results produced by the process.

- **plot** signature(x = "mtkDefaultAnalyser"): Reports graphically the results produced by the process.

- **report** signature(this = "mtkDefaultAnalyser"): Reports the results produced by the process.

**Author(s)**

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

**References**


**Examples**

```r
# Create a designer and an analyser avec the method "Morris"
# to analyze the model "Ishigami":

# Specify the factors to analyze:
x1 <- make.mtkFactor(name="x1", distribName="unif",
```
distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
distribPara=list(min=-pi, max=pi))
factors <- mtkExpFactors(list(x1,x2,x3))
# Build the processes:
# 1) the experimental design process with the method "Morris".
expl.designer <- mtkNativeDesigner(design = "Morris",
information=list(r=20,type="oat",levels=4,grid.jump=2))

# 2) the model simulation process with the model "Ishigami".
expl.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process with the default method.
# Here, it is the "Morris" method.
expl.analyser <- mtkDefaultAnalyser()

# Build the workflow with the processes defined previously.
expl <- mtkExpWorkflow(expFactors=factors,
processesVector = c(design=expl.designer,
evaluate=expl.evaluator, analyze=expl.analyser))

# Run the workflow and report the results.
run(expl)
print(expl)

mtkdesigner The constructor of the class mtkdesigner

Description
The constructor

Usage
mtkdesigner(protocol = "R", site = "mtk", service = ",
parameters = NULL, parametersList = NULL, ready = TRUE,
state = FALSE, result = NULL)

Arguments
  protocol (character) a string from "http", "system", "R" respectively representing if the
  process is implemented remotely, locally or as R function.
  site (character) a string to indicate where the service is located.
  service (character) a string to name the method or the service (if remotely) to invoke.
  parameters a vector of [mtkParameter] representing the parameters necessary to run the
  process.
parametersList  a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

ready  a logical to indicate if the process is ready to run.

state  a logical to indicate if the process finished running and the results are available.

result  an object of a class derived from [mtkDesignerResult] to hold the results produced by the designer.

Value

an object of the mtkDesigner class

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# Create a designer with the method "Morris"
# implemented in the package "mtk"
designer <- mtkDesigner(service="Morris",
parametersList=list(nboot=20))

mtkDesigner-class  

The mtkDesigner class

Description

The mtkDesigner class is a sub-class of the class mtkProcess used to manage the experiments design task. It provides all the slots and methods defined in the class mtkProcess.

Class Hierarchy

Parent classes: mtkProcess

Direct Known Subclasses: mtkNativeDesigner, mtkMorrisDesigner, etc.

Constructor

mtkDesigner  signature(protocol = "R", site = "mtk", service = ", parameters = NULL, parametersList = NULL, ready = TRUE, state = FALSE, result = NULL)
mtkDesigner-class

Slots

name: (character) always takes the string "design".
protocol: (character) a string to name the protocol used to run the process: http, system, R, etc.
site the site where the process is implemented if remotely or the package where the process is implemented if as a R function.
service a string corresponding to the name of the method implemented in the package "mtk" or the service that implements the process if remotely.
parameters: (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.
ready: (logical) a logical to tell if the process is ready to run.
state: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.
result: (ANY) a data holder from the class mtkDesignerResult to hold the results produced by the process.

Methods

setName signature(this = "mtkDesigner", name = "character"): Not used, method inherited from the parent class.
setParameters signature(this = "mtkDesigner", f = "vector"): Assigns new parameters to the process.
getParameters signature(this = "mtkDesigner"): Returns the parameters as a named list.
is.ready signature( = "mtkDesigner"): Tests if the process is ready to run.
setReady signature(this = "mtkDesigner", switch = "logical"): Makes the process ready to run.
is.ready signature( = "mtkDesigner"): Tests if the results produced by the process are available.
setReady signature(this = "mtkDesigner", switch = "logical"): Marks the process as already executed.
getResult signature(this = "mtkDesigner"): Returns the results produced by the process as mtkDesignerResult.
getData signature(this = "mtkDesigner"): Returns the results as a data.frame.
serializeOn signature(this = "mtkDesigner"): Returns all data managed by the process as a named list.
run signature(this = "mtkDesigner", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.
summary signature(object = "mtkDesigner"): Provides a summary of the results produced by the process.
print signature(x = "mtkDesigner"): Prints a report of the results produced by the process.
plot signature(x = "mtkDesigner"): Reports graphically the results produced by the process.
report signature(this = "mtkDesigner"): Reports the results produced by the process.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr
mtkDesignerResult

References


Examples

```r
# Create a designer with the method "Morris"
# implemented in the package "mtk"
designer <- mtkDesigner(service="Morris",
parametersList=list(nboot=20))
```

```r
tmtdesignerresult
```

The constructor of the class mtkDesignerResult

Description

The constructor

Usage

```r
mtkDesignerResult(main=data.frame(),information=list())
```

Arguments

- `main` a data.frame holding the experimental design produced by the designer.
- `information` a named list containing the information about the experiments design.

Value

an object of the mtkDesignerResult class

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References

mtkDesignerResult-class

Examples

# Construct an object of the \code{mtkDesignerResult} # class from a data-frame.
data <- data.frame()
expDesign <- mtkDesignerResult(main=data,  
   information = list(sampling="Fast"))

The \code{mtkDesignerResult-class}

Description

A class to collect the experimental design produced by an experiments design process.

Class Hierarchy

\textbf{Parent classes} : \code{mtkResult}

\textbf{Direct Known Subclasses} : mtkSobolDesignerResult, mtkMorrisDesignerResult, etc.

Constructor

\code{mtkDesignerResult} signature(main=data.frame(),information=list())

Slots

\textbf{main}: \code{(data.frame)} a data.frame holding the experimental design produced by the process.
\textbf{information}: \code{(list)} a named list containing optional information about the experiments design.

Methods

\textbf{summary} signature(object = "mtkDesignerResult"): Provides a summary of the experimental design produced by the design process.
\textbf{print} signature(x = "mtkDesignerResult"): Prints a report of the experimental design produced by the design process.
\textbf{plot} signature(x = "mtkDesignerResult"): Plots the experimental design produced by the design process.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References

Examples

```r
# Construct an object of the mtkDesignerResult class from a data-frame.
data <- data.frame()
expDesign <- mtkDesignerResult(main=data,
                               information = list(sampling="Fast"))
```

### mtkDomain

*The constructor of the class mtkDomain*

#### Description

The constructor of the class `mtkDomain`.

#### Usage

```r
mtkDomain(distributionName="unknown",domainNominalValue=0,
          distributionParameters=list())
```

#### Arguments

- **distributionName**
  - a string corresponding to the distribution name associated with the domain.
- **domainNominalValue**
  - an object of the `mtkValue` class or information allowing to create an object of
    the `mtkValue` class, used to hold the nominal value of the domain.
- **distributionParameters**
  - a list to hold the parameters of the distribution associated with the domain.

#### Value

an object of the `mtkDomain` class

#### Examples

```r
# creates a new domain with a continue distribution
d <- mtkDomain(distributionName="unif", domainNominalValue=0,
               distributionParameters = list(max=3, min=0))

# creates a new domain with a discrete distribution
d <- mtkDomain(distributionName="discrete", domainNominalValue=3,
               distributionParameters = list(type='categorical',
                                            levels = c(1,2,3,4,5), weights=rep(0.2, 5)))
```
The `mtkDomain` class is a class used to manage the uncertainty domain associated with a factor.

**Description**

The `mtkDomain` class is a class used to manage the uncertainty domain associated with a factor.

**Class Hierarchy**

- **Parent classes:**
- **Direct Known Subclasses:**

**Constructor**

```r
class(mtkDomain) signature(distributionName = "unknown", domainNominalValue = 0, distributionParameters = list())
```

**Slots**

- `distributionName`: (character) a string representing the distribution law.
- `nominalValue`: (mtkValue) the nominal value of the domain.
- `levels`: (mtkLevels) an object of `mtkLevels` class.
- `distributionParameters`: (list) a list of `mtkParameter` objects.

**Methods**

- `initialize` signature(`Object = "mtkDomain"`): The initializer of the class `mtkDomain`.
- `getDistributionName` signature(`this = "mtkDomain"`): Returns the distribution’s name.
- `getNominalValue` signature(`this = "mtkDomain"`): Returns the the nominal value.
- `getNominalValueType` signature(`this = "mtkDomain"`): Returns the value type of the nominal value.
- `getDiscreteDistributionType` signature(`this = "mtkDomain"`): Returns the type of the discrete distribution.
- `getLevels` signature(`this="mtkDomain"`): Fetches the the levels managed by the domain.
- `getWeights` signature(`this="mtkDomain"`): Fetches the the weights managed by the domain.
- `getDistributionParameters` signature(`this = "mtkDomain"`): Fetches the parameters of the distributions associated with the domain.
- `setLevels` signature(`this="mtkDomain", levels = "vector"`): Affects a new level to the domain where levels is a named list like list(type='categorical', levels=c(1,2,3,4,5), weights=c(0.2, 0.2, 0.2, 0.2, 0.2)).
- `setLevels` signature(`this="mtkDomain", levels = "mtkLevels"`): Affects a new level to the domain where levels is an object from the class `mtkLevels`.
setDistributionParameters signature(this = "mtkDomain", aDistParamList="list"): Affects a new list of parameters to the domain. For continue distributions, aDistParamList may be a list of objects of the class mtkParameter or a named list like list(max=5, min=1). For discrete distributions, aDistParamList may be a named list containing an object of the class mtkLevels or a named list like list(type='categorical', levels = c(1,2,3,4,5), weights=rep(0.2, 5)) from which we can build an object of the class mtkLevels.

print signature(x = "mtkDomain"): Prints the data managed by the domain.

show signature(object = "mtkDomain"): Displays the underlying object of the class mtkDomain.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# Create a new domain with a continue distribution
d <- mtkDomain(distributionName="unif", domainNominalValue=0, distributionParameters = list(max=3, min=0))

# Create a new domain with a discrete distribution
d <- mtkDomain(distributionName="discrete", domainNominalValue=3, distributionParameters = list(type='categorical', levels = c(1,2,3,4,5, weights=rep(0.2, 5)))
# Change the levels to list(type='categorical', levels = c('a','b','c','d'), weights=rep(0.25, 4))
setLevels(d, list(type='categorical', levels = c('a','b','c','d'), weights=rep(0.25, 4)))
Arguments

- **protocol**: a string from "http", "system", "R" respectively representing if the process is implemented remotely, locally or as an R function.
- **site**: the site where the process is implemented if remotely or the package where the process is implemented if as an R function.
- **service**: a string corresponding to the name of the method implemented in the package "mtk" or the service that implements the process if remotely.
- **parameters**: a vector of [mtkParameter] representing the parameters necessary to run the process.
- **parametersList**: a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.
- **ready**: a logical to indicate if the process is ready to run.
- **state**: a logical to indicate if the process finished running and the results are available.
- **result**: an object of the class [mtkEvaluatorResult] to hold the results produced by the Evaluator.

Value

an object of the mtkEvaluator class

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# Create an evaluator with the model "Ishigami" implemented in the package "mtk".
evaluator1 <- mtkEvaluator(service="Ishigami")

# Create an evaluator avec the model "WWDM" implemented in the package "mtk"
evaluator2 <- mtkEvaluator(service="WWDM",
parametersList=list(year=3, tout=FALSE))
The mtkEvaluator class is a sub-class of the class mtkProcess used to manage the model simulation. It provides all the slots and methods defined in the class mtkProcess.

Class Hierarchy

Parent classes: mtkProcess

Direct Known Subclasses: mtkNativeEvaluator, mtkWWDMEvaluator, etc.

Constructor

mtkEvaluator signature(protocol = "R", site = "mtk", service = "", parameters = NULL, parametersList = NULL, ready = TRUE, state = FALSE, result = NULL)

Slots

name: (character) always takes the string "evaluate".
protocol: (character) a string to name the protocol used to run the process: http, system, R, etc.
site: (character) a string to indicate where the service is located.
service: (character) a string to name the service to invoke.
parameters: (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.
ready: (logical) a logical to tell if the process is ready to run.
state: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.
result: (ANY) a data holder to hold the results produced by the process

Methods

setName signature(this = "mtkEvaluator", name = "character"): Not used, method inherited from the parent class.
setParameters signature(this = "mtkEvaluator", f = "vector"): Assigns new parameters to the process.
getParameters signature(this = "mtkEvaluator"): Returns the parameters as a named list.
is.ready signature( = "mtkEvaluator"): Tests if the process is ready to run.
setReady signature(this = "mtkEvaluator", switch = "logical"): Makes the process ready to run.
is.ready signature( = "mtkEvaluator"): Tests if the results produced by the process are available.
setReady signature(this = "mtkEvaluator", switch = "logical"): Marks the process as already executed.
**mtkEvaluatorResult**

getResult signature(this = "mtkEvaluator"): Returns the results produced by the process as a [mtkEvaluatorResult].

data signature(this = "mtkEvaluator"): Returns the results produced by the process as a data.frame.

serializeOn signature(this = "mtkEvaluator"): Returns all data managed by the process as a named list.

run signature(this = "mtkEvaluator", context="mtkExpWorkflow"): Runs the model with the experimental design defined in the context.

summary signature(object = "mtkEvaluator"): Provides a summary of the results produced by the process.

print signature(x = "mtkEvaluator"): Prints a report of the results produced by the process.

plot signature(x = "mtkEvaluator"): Plots the results produced by the process.

report signature(this = "mtkEvaluator"): Reports the results produced by the process.

**Author(s)**

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

**References**


**Examples**

```r
# Create an evaluator with the model "Ishigami"
# implemented in the package "mtk".
evaluator1 <- mtkEvaluator(service="Ishigami")

# Create an evaluator with the model "WWDM"
# implemented in the package "mtk"
evaluator2 <- mtkEvaluator(service="WWDM",
parametersList=list(year=3, tout=FALSE))
```

---

**mtkEvaluatorResult**  The constructor of the class mtkEvaluatorResult

**Description**

The constructor

**Usage**

```r
mtkEvaluatorResult(main=data.frame(), information=list())
```
mtkEvaluatorResult-class

Arguments

- **main**: a data.frame holding the data produced by the model simulation.
- **information**: a named list containing the information about the managed data or process.

Value

- an object of the mtkEvaluatorResult class

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

```r
# Construct an object of the mtkEvaluatorResult class
# class from a data-frame.
data <- data.frame()
simulation <- mtkEvaluatorResult(main=data,
information = list(model="Ishigami"))
```

Description

A class to collect the results of the simulation produced with a model.

Class Hierarchy

Parent classes: mtkResult

Direct Known Subclasses: mtkWWDEvaluatorResult, etc.

Constructor

**mtkEvaluatorResult** signature(main=data.frame(),information=list())

Slots

- **main**: (data.frame) a data.frame holding the data produced by the model simulation.
- **information**: (list) a named list containing information about the managed data and process.
Methods

summary signature(object = "mtkEvaluatorResult"): Provides a summary of the data produced with the model simulation.

print signature(x = "mtkEvaluatorResult"): Prints a report of the data produced with the model simulation.

plot signature(x = "mtkEvaluatorResult"): Plots the data produced with the model simulation.

See Also

help(morris, sensitivity) and help(Regression)

Examples

## See examples from help(mtkAnalyserResult)

mtkExperiment

The constructor of the class mtkExperiment

Description

A simple way to build a workflow for interactive use.

Usage

mtkExperiment(expFactors,
        design=NULL, designInfo=NULL,
        model=NULL, modelInfo=NULL,
        analyze=NULL, analyzeInfo=NULL,
        XY=NULL)

Arguments

expFactors (mtkExpFactors) an object of the mtkExpFactors class.

design (NULL or character) the name of the method used to build the experiment design. NULL means that the experiment design is produced off-line and should be imported through the parameter XY$X.

designInfo (list) a named list to specify the parameters used to generate the experiments design.

model (NULL or character) the name of the model to simulate. NULL means that the simulation is produced off-line and should be imported through the parameter XY$Y.
**modelInfo**

(list) a named list to specify the parameters used to manage the model simulation.

**analyze**

(NULL or character) the name of the method used to compute the sensitivity index.

**analyzeInfo**

(list) a named list to specify the parameters used to carry out the analyses.

**XY**

(NULL or list) a named list with two elements X and Y: X allows importing the experiment design produced off-line and Y allows importing the model simulation produced off-line.

**Value**

an object of the `mtkExperiment` class

**Author(s)**

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

**References**


**Examples**

```r
# Compute the sensitivity index with the method "Regression"
# over the model "Ishigami" according to an experiment design
# generated with the method "BasicMonteCarlo"

x1 <- make.mtkFactor(name="x1", distribName="unif",
                     distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
                     distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
                     distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

ishiReg <- mtkExperiment(expFactors=ishi.factors,
                         design="BasicMonteCarlo", designInfo=list(size=20),
                         model="Ishigami",
                         analyze="Regression", analyzeInfo=list(nboot=20))

run(ishiReg)
summary(ishiReg)
```
The class `mtkexperiment` is a sub-class of the class `mtkexpflow`. It provides more facilities and more flexible use for interactive manipulation of the workflow. Different behaviors may be expected by appropriately combining the parameters: design – the method used for the experiment design; model – the model used for the simulation; analyze – the method used for calculating the sensitivity index; XY – argument used to provide with data produced off-line;

For example, 1) if the experiment design is produced off-line, it will be imported with the help of the parameter "XY$X"; 2) if the model simulation is produced off-line, it will be imported through the parameter "XY$Y";

Class Hierarchy

**Parent classes**: `mtkexpflow`  
**Direct Known Subclasses**: 

Constructor

```r
mtkexperiment signature(expFactors, design=NULL, designInfo=NULL, model=NULL, modelInfo=NULL, analyze=NULL, analyzeInfo=NULL, XY=NULL)
```

Slots

- `expFactors`: (mtkExpFactors) an object of the `mtkExpFactors` class.
- `processesVector`: (vector) a vector of objects from the class `mtkProcess` or its sub-classes.

Methods

- `addProcess` signature(this = "mtkExperiment", p = "mtkProcess", name = "character"): Adds a process to the workflow.
- `deleteProcess` signature(this = "mtkExperiment", name = "character"): Deletes a process from the workflow.
- `setProcess` signature(this = "mtkExperiment", p = "mtkProcess", name = "character"): Replaces a process into the workflow.
- `getProcess` signature(this = "mtkExperiment", name = "character"): Gets a process from the workflow.
- `extractData` signature(this = "mtkExperiment", name = "list"): Returns the results produced by the workflow as a data.frame. According to the processes specified with the argument "name", we can fetch the results produced by the process "design", "evaluate" or "analyze". i.e. name="c("design") gives the experimental design produced by the process "design" and name="c("design","evaluate") gives both the experimental design and the model simulation, etc.
mtkExperiment-class

reevaluate signature(this = "mtkExperiment", name = "character"): Re-evaluate the processes of the workflow to know if they should be re-run. This should be done after changing a process of the workflow. According to the order "design", "evaluate", "analyze", only the processes after the one given by the argument "name" will be re-evaluated.

run signature(this = "mtkExperiment", context= "missing"): Runs the ExpWorkflow.

serializeOn signature(this = "mtkExperiment"): Returns all data managed by the workflow as a named list.

summary signature(object = "mtkExperiment"): Provides a summary of the results produced by the workflow.

print signature(x = "mtkExperiment"): Prints a report of the results produced by the workflow.

plot signature(x = "mtkExperiment"): Plots the results produced by the workflow.

report signature(this = "mtkExperiment"): Reports the results produced by the workflow.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# Compute the sensitivity index with the method "Regression"
# over the model "Ishigami" according to an experiment design
# generated with the method "BasicMonteCarlo"

x1 <- make.mtkFactor(name="x1", distribName="unif",
                 distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
                 distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
                 distribPara=list(min=-pi, max=pi))
factors <- mtkExpFactors(list(x1,x2,x3))

exp <- mtkExperiment(
    factors,
    design = 'BasicMonteCarlo',
    designInfo=list(size=20),
    model = 'Ishigami',
    analyze = 'Regression',
    analyzeInfo = list(ntboot=20)
)
run(exp)
summary(exp)
mtkExpFactors

The constructor of the class mtkExpFactors

Description
This class is used to define the input factors for a simulation experiment.

Usage
mtkExpFactors(expFactorsList=list())

Arguments
expFactorsList  a list of mtkFactor objects.

Value
an object of the mtkExpFactors class

Author(s)
Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

Examples

# Create an object of the class mtkExpFactor
x1 <- make.mtkFactor(name="x1", distribName="unif",
                      distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
                      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
                      distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

mtkExpFactors-class  The mtkExpFactors class

Description
The mtkExpFactors class is a class used to manage the factors involved in a sensitivity analysis.

Class Hierarchy
Parent classes :
Direct Known Subclasses :
**Constructor**

```r
mtkExpFactors signature(expFactorsList=list())
```

**Slots**

```
expFactorsList: (list) a list of mtkFactor objects.
```

**Methods**

```r
initialize signature(.Object="mtkExpFactors") : The initializer.
setFactors signature(this="mtkExpFactors", aFactList="list") : Assigns a new list of mtkFactor objects.
getFactors signature(this="mtkExpFactors") : Returns the factors as a list of mtkFactor objects.
getNames signature(this = "mtkExpFactors") : Returns the names of the managed factors.
getFactorNames signature(this = "mtkExpFactors") : Returns the names of the managed factors as the method getNames.
getDistributionNames signature(this="mtkExpFactors") : Gets a list of mtkExpFactors names.
getDistributionParameters signature(this="mtkExpFactors") : Gets the parameters.
getFeatures signature(this = "mtkExpFactors") : Returns the features associated with the managed factors.
getDistributionNominalValues signature(this="mtkExpFactors") : Returns the nominal values associated with the distributions of the managed factors.
getDistributionNominalValueTypes signature(this="mtkExpFactors") : Returns the data type of the nominal value associated with the managed factors.
[ signature( x = "mtkExpFactors", i="ANY") : Extracts or replaces parts of an object of the class mtkExpFactors.
[ signature( x = "mtkExpFactors", i="ANY") : Extracts or replaces parts of an object of class mtkExpFactors.
$ signature(x = "mtkExpFactors") : Extracts or replaces parts of an object of the class.
print signature(x = "mtkExpFactors") : Prints information about the managed factors.
show signature(object = "mtkExpFactors") : Displays the underlying object of the class mtkExpFactors.
```

**Author(s)**

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

**Examples**

```r
# Define the factor
x1 <- make.mtkFactor(name="x1", distribName="unif",
distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
distribPara=list(min=-pi, max=pi))
```
The constructor of the class mtkExpWorkflow

Description

The class mtkExpWorkflow is used to manage the processes involved in a sensitivity analysis. We can construct a workflow in two ways: either from pre-defined factors and processes or from a XML file.

Usage

mtkExpWorkflow(
  expFactors = NULL,
  processesVector = NULL,
  xmlFilePath = NULL
)

Arguments

  expFactors (mtkExpFactors) an object of the mtkExpFactors class.
  processesVector (vector) a vector of objects from the class mtkProcess or its sub-classes.
  xmlFilePath (character) a string holding the name of the XML file and its path.

Value

  an object of the mtkExpWorkflow class

Author(s)

  Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References

Examples

#############
# Example 1: Construct a workflow
# from the factors and the processes
#############

x1 <- make.mtkFactor(name="x1", distribName="unif",
                      distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
                      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
                      distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))

designer <- mtkNativeDesigner("BasicMonteCarlo",
                              information=list(size=20))
model <- mtkNativeEvaluator("Ishigami")
analyser <- mtkNativeAnalyzer("Regression",
                              information=list(nboot=20) )

ishiReg <- mtkExpWorkflow(expFactors=ishi.factors,
                          processesVector=c( design=designer,
                                             evaluate=model,
                                             analyze=analyser)
                      )
run(ishiReg)
summary(ishiReg)

#############
# Example 2: Construct a workflow from a XML file
#############

## Create a workflow from XML file
## Nota: If your XML file is a local file
## for example /var/tmp/X.xml", you should
## create the workflow as follows:
## workflow <- mtkExpWorkflow(
##   xmlFilePath="/var/tmp/X.xml"
## )

xmlFile <- "WWDM_morris.xml"

## If WWDM_morris.xml is a local file, the next line is not necessary.
xmlFilePath <- paste(path.package("mtk", quiet = TRUE),
                     "/extdata/",xmlFile,sep = "")
workflow <- mtkExpWorkflow(xmlFilePath=xmlFilePath)

# Run the workflow and report the results
run(workflow)
summary(workflow)
**mtkExpWorkflow-class**

The *mtkExpWorkflow* class is used to coordinate the processes involved in a sensitivity analysis. It controls the state of the processes and coordinates their chaining.

**Class Hierarchy**

**Parent classes:**
- Direct Known Subclasses:

**Constructor**

```r
class <- function(class) {
  if (class == "mtkExpWorkflow") {
    signature(expFactors=NULL, processesVector=NULL, xmlFilePath=NULL)
  }
}
```

**Slots**

- `expFactors`: (*mtkExpFactors*) an object of the *mtkExpFactors* class.
- `processesVector`: (*vector*) a vector of objects from the class *mtkProcess* or its sub-classes.

**Methods**

- `addProcess` signature(this = "mtkExpWorkflow", p = "mtkProcess", name = "character"): Adds a process to the workflow.
- `deleteProcess` signature(this = "mtkExpWorkflow", name = "character"): Deletes a process from the workflow.
- `setProcess` signature(this = "mtkExpWorkflow", p = "mtkProcess", name = "character"): Replaces a process into the workflow.
- `getProcess` signature(this = "mtkExpWorkflow", name = "character"): Gets a process from the workflow.
- `extractData` signature(this = "mtkExpWorkflow", name = "list"): Returns the results produced by the workflow as a data.frame. According to the processes specified with the argument "name", we can fetch the results produced by the process "design”, “evaluate” or “analyze”. i.e. name=c("design") gives the experimental design produced by the process "design" and name=c("design","evaluate") gives both the experimental design and the model simulation, etc.
- `reevaluate` signature(this = "mtkExpWorkflow", name = "character"): Re-evaluate the processes of the workflow to know if they should be re-run. This should be done after changing a process of the workflow. According to the order "design", evaluate”, “analyze”, only the processes after the one given by the argument "name" will be re-evaluated.
- `run` signature(this = "mtkExpWorkflow", context= "missing"): Runs the workflow.
serializeOn signature(this = "mtkExpWorkflow"): Returns all data managed by the workflow as a named list.

summary signature(object = "mtkExpWorkflow"): Provides a summary of the results produced by the workflow.

print signature(x = "mtkExpWorkflow"): Prints a report of the results produced by the workflow.

plot signature(x = "mtkExpWorkflow"): Plots the results produced by the workflow.

report signature(this = "mtkExpWorkflow"): Reports the results produced by the workflow.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

############################
# Example 1: Construct a workflow
# from the factors and the processes
############################

# Specify the factors
x1 <- make.mtkFactor(name="x1", distribName="unif", distribPara=list(min=-pi, max=pi))

x2 <- make.mtkFactor(name="x2", distribName="unif", distribPara=list(min=-pi, max=pi))

x3 <- make.mtkFactor(name="x3", distribName="unif", distribPara=list(min=-pi, max=pi))

ishi.factors <- mtkExpFactors(list(x1,x2,x3))

# Define the processes
designer <- mtkNativeDesigner("BasicMonteCarlo", information=list(size=20))

cmodel <- mtkNativeEvaluator("Ishigami")

analyser <- mtkNativeAnalyser("Regression", information=list(nboot=20) )

# Build the workflow
ishiReg <- mtkExpWorkflow( expFactors=ishi.factors,
  processesVector=c( design=designer,
    evaluate=model,
    analyze=analyser) )

# Run the workflow and report the results
### Example 2: Construct a workflow from a XML file

#### XML file is held in the directory of the library: “inst/extdata/”

```r
# Specify the XML file's name
xmFile <- "WWDM_morris.xml"
# find where the examples are held.
xmFilePath <- paste(path.package("mtk", quiet = TRUE),
"/extdata/", xmFile, sep = "")

# Create the workflow from the XML
# Nota: If your XML file is local
# file for example /var/tmp/X.xml", you should
# create the workflow as follows:
# workflow <- mtkExpWorkflow(
# xmFilePath = "/var/tmp/X.xml"
# )
workflow <- mtkExpWorkflow(xmFilePath=xmFilePath)

# Run the workflow and report the results
run(workflow)
summary(workflow)
```

---

**mtkFactor**

*The constructor of the class* **mtkFactor**

**Description**

The constructor of the class **mtkFactor**. See also the function *make.mtkFactor*

**Usage**

```r
mtkFactor(name="unkown", id="unkown", unit="", type="numeric",
domain=mtkDomain(), featureList=list())
```

**Arguments**

- **name**: a string to name the factor.
- **id**: a string giving the id of the factor in the code.
- **unit**: a string giving the measurement unit of the factor levels.
- **type**: a string giving the data type of the factor levels.
- **domain**: an object of the class **mtkDomain** giving the uncertainty domain associated with the factor.
featureList  a list giving the uncertainty domain associated with the factor. It may be a list of objects from the class mtkDomain or a named list defining the features.

Value

an object of the mtkFactor class

Author(s)

Juhui WANG, MIA-jouy, INRA

Examples

# Create an object of the class mtkExpFactor
x1 <- make.mtkFactor(name="x1", distribName="unif",
  distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
  distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
  distribPara=list(min=-pi, max=pi))

mtkFactor-class  The mtkFactor class

Description

The class used to manage an input factor and its uncertainty distribution.

Class Hierarchy

Parent classes :

Direct Known Subclasses :

Constructor

mtkFactor  signature(name="unkown", id="unkown", unit="", type="numeric", domain=mtkDomain(),
  featureList=list())

Slots

name: the name of the input factor.
id: the name of the factor in the simulation code, if different from name.
unit: the measurement units of the factor values. This can be used in graphics or reports, for example.
type: the data type of the factor’s values.
domain: the mtkDomain object that describes the factor’s uncertainty.
featureList: the list of features that may be associated with the factor.
Methods

```
initialize signature(.Object = "mtkFactor"): The initializer of the class mtkFactor.
getName signature(this="mtkFactor"): Fetches the name of the factor.
getType signature(this = "mtkFactor"): Returns the data type of the factor's levels.
getDomain signature(this="mtkFactor"): Fetches the domain associated with the factor. It returns an object of the class mtkDomain.
getDistributionName signature(this="mtkFactor"): Fetches the name of the distribution associated with the uncertainty domain.
getDistributionNominalValue signature(this="mtkFactor"): Fetches the nominal value of the distribution associated with the uncertainty domain.
getDistributionNominalValueType signature(this="mtkFactor"): Fetches the data type associated with the uncertainty domain.
getDiscreteDistributionType signature(this="mtkFactor"): Returns the discrete distribution type.
getDiscreteDistributionLevels signature(this="mtkFactor"): Returns the levels managed by a discrete distribution.
getDiscreteDistributionWeights signature(this="mtkFactor"): Returns the weights managed by a discrete distribution.
getDistributionParameters signature(this="mtkFactor"): The getDistributionParameters method.
getFeatures signature(this="mtkFactor"): Returns the features as a named list.
getMtkFeatures signature(this="mtkFactor"): Returns the features as a vector of objects from the class mtkFeature.
setName signature(this = "mtkFactor", name = "character"): Gives a new name to the factor.
setDomain signature(this = "mtkFactor", domain = "mtkDomain"): Associates a new domain with the factor.
setType signature(this = "mtkFactor", type = "character"): Names explicitly the data type managed by the factor.
setFeatures signature(this="mtkFactor",aFList="list"): Gives new features to the factor. aFList may be a vector of objects from the class mtkFeature or a named list from which we can build a list of features.
print signature(x = "mtkFactor"): Prints the data managed by the factor.
show signature(object = "mtkFactor"): Displays the underlying object of the class mtkFactor.
```

Author(s)

Juhui WANG and Hervé Monod, MIA-jouy, INRA, Hervé Richard, BioSP, INRA

Examples

```
# Manage a factor x1 with a mtkFactor object.
x1 <- make.mtkFactor(name="x1", distrName="unif",
```
mtkFastAnalyser

```r
  distribPara=list(min=-pi, max=pi))
  getName(x1)
  getDomain(x1)
  getDistributionName(x1)
  getType(x1)
  setType(x1, "double")
  getType(x1); # 'double'
```

This constructor of the class `mtkFastAnalyser` takes two arguments:

- `mtkParameters`: a vector of `mtkParameter` representing the parameters necessary to run the process.
- `listParameters`: a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

The constructor

The constructor

Usage

```r
  mtkFastAnalyser(mtkParameters = NULL, listParameters = NULL)
```

Arguments

- `mtkParameters`: a vector of `mtkParameter` representing the parameters necessary to run the process.
- `listParameters`: a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

Value

an object of the `mtkFastAnalyser` class

References


See Also

- `help(fast, sensitivity)`
Examples

```r
##  Sensitivity analysis of the "Ishigami" model with the "Fast" method

# Input the factors
data(Ishigami.factors)

# Build the processes and workflow:

# 1) the design process
exp1.designer <- mtkFastDesigner(listParameters = list(n=1000))

# 2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process
exp1.analyser <- mtkFastAnalyser()

# 4) the workflow
exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors, processesVector = c(design=exp1.designer, evaluate=exp1.evaluator, analyze=exp1.analyser))

# Run the workflow and reports the results.
run(exp1)
print(exp1)
```

mtkFastAnalyser-class  The mtkFastAnalyser class

Description

The `mtkFastAnalyser` class is a sub-class of the class `mtkAnalyser`. It implements the sensitivity analysis method 'Fast' and provides all the slots and methods defined in the class `mtkAnalyser`.

Class Hierarchy

Parent classes: `mtkAnalyser`

Direct Known Subclasses:

Constructor

`mtkFastAnalyser` signature(mtkParameters = NULL, listParameters = NULL)
Slots

name: (character): always takes the string "analyze".
protocol: (character): always takes the string "R".
site: (character): always takes the string "mtk".
service: (character): always takes the string "Fast".
parameters: (vector): a vector of [mtkParameter] containing the parameters to pass while calling the service.
ready: (logical): a logical to tell if the process is ready to run.
state: (logical): a logical to tell if the results produced by the process are available and ready to be consumed.
result: (ANY): a data holder to hold the results produced by the process

Methods

setName signature(this = "mtkFastAnalyser", name = "character"): Not used, method inherited from the parent class.
setParameters signature(this = "mtkFastAnalyser", f = "vector"): Assigns new parameters to the process.
getParameters signature(this = "mtkFastAnalyser"): Returns the parameters as a named list.
is.ready signature( = "mtkFastAnalyser"): Tests if the process is ready to run.
setReady signature(this = "mtkFastAnalyser", switch = "logical"): Makes the process ready to run.
is.ready signature( = "mtkFastAnalyser"): Tests if the results produced by the process are available.
setReady signature(this = "mtkFastAnalyser", switch = "logical"): Marks the process as already executed.
getResult signature(this = "mtkFastAnalyser"): Returns the results produced by the process as a [mtkAnalyserResult].
getData signature(this = "mtkFastAnalyser"): Returns the results produced by the process as a data.frame.
serializeOn signature(this = "mtkFastAnalyser"): Returns all data managed by the process as a named list.
run signature(this = "mtkFastAnalyser", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.
summary signature(object = "mtkFastAnalyser"): Provides a summary of the results produced by the process.
print signature(x = "mtkFastAnalyser"): Prints a report of the results produced by the process.
plot signature(x = "mtkFastAnalyser"): Plots the results produced by the process.
report signature(this = "mtkFastAnalyser"): Reports the results produced by the process.
References


See Also

help(fast, sensitivity)

Examples

```r
## Sensitivity analysis of the "Ishigami" model with the "Fast" method

# Input the factors
data(Ishigami.factors)

# Build the processes and workflow:

# 1) the design process
exp1.designer <- mtkFastDesigner(listParameters
                              = list(n=1000))

# 2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process
exp1.analyser <- mtkFastAnalyser()

# 4) the workflow
exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
                        processesVector = c(design=exp1.designer,
                                             evaluate=exp1.evaluator,
                                             analyze=exp1.analyser))

# Run the workflow and reports the results.
run(exp1)
print(exp1)
```

mtkFastAnalyserResult  The constructor of the class mtkFastAnalyserResult

Description

The constructor
mtkFastAnalyserResult-class

Usage

mtkFastAnalyserResult(main, information=NULL)

Arguments

main a data.frame holding the results of the sensitivity analysis produced by the analyser.
information a named list containing the information about the managed data.

Value

an object of the mtkFastAnalyserResult class

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# See examples from the help of the method: help(Fast)

mtkFastAnalyserResult-class

The mtkFastAnalyserResult class

Description

A class to collect the results of the sensitivity analysis produced by the analyser implementing the method Fast.

Class Hierarchy

Parent classes: mtkAnalyserResult
Direct Known Subclasses:

Constructor

mtkFastAnalyserResult signature(main, information=NULL)
Slots

main: (data.frame) a data.frame holding the experimental design.

information: (NULL) a named list containing optional information about the managed data.

Methods

summary signature(object = "mtkFastAnalyserResult"): Provides a summary of the results produced by the analyser.

print signature(x = "mtkFastAnalyserResult"): Prints a report of the results produced by the analyser.

plot signature(x = "mtkFastAnalyserResult"): Plots the results produced by the analyser.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# See examples from the help of the method: help(Fast)

mtkFastDesigner

The constructor of the class mtkFastDesigner

Description

The constructor

Usage

mtkFastDesigner(mtkParameters = NULL, listParameters = NULL)

Arguments

mtkParameters a vector of [mtkParameter] representing the parameters necessary to run the process.

listParameters a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.
**Value**

an object of the `mtkFastDesigner` class

**See Also**

`help(fast, sensitivity)`

**Examples**

```r
## Sensitivity analysis of the "Ishigami" model with the "Fast" method

# Input the factors
data(Ishigami.factors)

# Build the processes and workflow:

# 1) the design process
exp1.designer <- mtkFastDesigner(listParameters = list(n=1000))

# 2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process
exp1.analyser <- mtkFastAnalyser()

# 4) the workflow
exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
                        processesVector = c(design=exp1.designer,
                                             evaluate=exp1.evaluator, analyze=exp1.analyser))

# Run the workflow and reports the results.
run(exp1)
print(exp1)
```

---

**Description**

The `mtkFastDesigner` class is a sub-class of the class `mtkDesigner`. It implements the sampling method Fast and provides all the slots and methods defined in the class `mtkDesigner`. 
mtkFastDesigner-class

Class Hierarchy

Parent classes: mtkDesigner

Direct Known Subclasses:

Constructor

mtkFastDesigner signature(mtkParameters = NULL, listParameters = NULL)

Slots

name: (character) always takes the string "design".
protocol: (character) always takes the string "R".
site: (character) always takes the string "mtk".
service: (character) always takes the string "Fast".
parameters: (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.
ready: (logical) a logical to tell if the process is ready to run.
state: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.
result: (ANY) a data holder to hold the results produced by the process

Methods

setName signature(this = "mtkFastDesigner", name = "character"): Not used, method inherited from the parent class.
setParameters signature(this = "mtkFastDesigner", f = "vector"): Assigns new parameters to the process.
getParameters signature(this = "mtkFastDesigner"): Returns the parameters as a named list.
is.ready signature(= "mtkFastDesigner"): Tests if the process is ready to run.
setReady signature(this = "mtkFastDesigner", switch = "logical"): Makes the process ready to run.
is.ready signature(= "mtkFastDesigner"): Tests if the results produced by the process are available.
setReady signature(this = "mtkFastDesigner", switch = "logical"): Marks the process as already executed.
getResult signature(this = "mtkFastDesigner"): Returns the results produced by the process as a [mtkDesignerResult].
getData signature(this = "mtkFastDesigner"): Returns the results produced by the process as a data.frame.
serializeOn signature(this = "mtkFastDesigner"): Returns all data managed by the process as a named list.
run signature(this = "mtkFastDesigner", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.
summary signature(object = "mtkFastDesigner"): Provides a summary of the results produced by the process.

print signature(x = "mtkFastDesigner"): Prints a report of the results produced by the process.

plot signature(x = "mtkFastDesigner"): Plots the results produced by the process.

report signature(this = "mtkFastDesigner"): Reports the results produced by the process.

References


See Also

help(fast, sensitivity)

Examples

```r
## Sensitivity analysis of the "Ishigami" model with the "Fast" method

# Input the factors
data(Ishigami.factors)

# Build the processes and workflow:

# 1) the design process
expl1.designer <- mtkFastDesigner(listParameters = list(n=1000))

# 2) the simulation process
expl1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process
expl1.analyser <- mtkFastAnalyser()

# 4) the workflow
expl1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
                       processesVector = c(design=expl1.designer,
                       evaluate=expl1.evaluator, analyze=expl1.analyser))

# Run the workflow and reports the results.
run(expl1)
print(expl1)
```
The constructor of the class `mtkFastDesignerResult`

**Description**

The constructor

**Usage**

```r
mtkFastDesignerResult(main, information=NULL)
```

**Arguments**

- `main` a data.frame holding the experimental design produced by the designer.
- `information` a named list containing the information about the managed data.

**Value**

an object of the `mtkFastDesignerResult` class

**Author(s)**

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

**References**


**Examples**

```r
# See examples from the help of the method: help(Fast)
```
The *mtkFastDesignerResult* class

**Description**

A class to collect the experimental design produced by the designer implementing the method `Fast`.

**Class Hierarchy**

- **Parent classes**: `mtkDesignerResult`
- **Direct Known Subclasses**: 

**Constructor**

`mtkFastDesignerResult` signature(main, information=NULL)

**Slots**

- `main` (*data.frame*) a data.frame holding the experimental design.
- `information` (*NULL*) a named list containing optional information about the managed data.

**Methods**

- **`summary`** signature(object = "mtkFastDesignerResult"): Provides a summary of the experimental design produced by the designer.
- **`print`** signature(x = "mtkFastDesignerResult"): Prints a report of the experimental design produced by the designer.
- **`plot`** signature(x = "mtkFastDesignerResult"): Plots the experimental design produced by the designer.

**Author(s)**

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

**References**


**Examples**

```r
# See examples from the help of the method: help(Fast)
```
Description

The constructor of the class `mtkFeature`. See also `make.mtkFeatureList`.

Usage

```r
mtkFeature(name='unknown', type='logical', val=NULL)
```

Arguments

- `name` *(character)* the name of the feature.
- `type` *(character)* the data type managed by the feature such as 'numeric', 'double', 'logical', etc..
- `val` *(ANY)* the value of the feature.

Value

- an object of the `mtkFeature` class

Author(s)

Juhui WANG, MIA-jouy, INRA

Examples

```r
# creates a feature "he"
f <- mtkFeature(name='he', type='character', val = 'pekin')

# We usually use the 'make.mtkFeatureList()' function to define
# a list of 'mtkFeature' instead of the constructor
# of the 'mtkFeature' class
flist <- make.mtkFeatureList(list(min=-1,max=+1,shape="hello"))
```
Description

The mtkFeature class is a class used to manage the features associated with a factor.

Class Hierarchy

Parent classes: mtkValue

Direct Known Subclasses:

Constructor

mtkFeature signature(name='unknown', type='logical', val=NULL)
make.mtkFeatureList signature(x=list())

Slots

name: (character) the name of the feature.
type: (character) the type of value managed by the feature.
val: (ANY) the value of the feature in the right type.

Methods

getName signature( this = "mtkFeature"): Returns the value of the slot "name".
getValue signature( this = "mtkFeature"): Returns the value of the slot "val".
getType signature(this = "mtkFeature"): Returns the value of the slot "type".
setName signature( this = "mtkFeature", name = "character"): Gives a new value to the slot "name".
setType signature( this = "mtkFeature", type = "character"): Gives a new value to the slot "type".
setValue signature( this = "mtkFeature", val = "ANY"): Gives a new value to the slot "val".
show signature( object = "mtkFeature"): Prints a report of the data managed by the underlying object.
print signature(x = "mtkFeature"): Prints the information managed by the underlying object.

Author(s)

Juhui WANG, MIA-jouy, INRA
Examples

# Create an object of the 'mtkFeature' class.
f <- mtkFeature(name="x", type="double", val=0.0)

# We usually use the make.mtkFeatureList function to define a list of mtkFeature
# instead of the constructor of the mtkFeature class
flist <- make.mtkFeatureList(list(min=-1,max=+1,shape="hello"))

mtkIshigamiEvaluator  The constructor of the class mtkIshigamiEvaluator

Description

The constructor

Usage

mtkIshigamiEvaluator()

Value

an object of the mtkIshigamiEvaluator class

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# Carry out a sensitivity analysis with the Ishigami model

## Input the factors
data(Ishigami.factors)

## Specify the experiments designer
designer <- mtkNativeDesigner("BasicMonteCarlo",
information=list(size=20))

## Specify the model simulator
model <- mtkIshigamiEvaluator()

## Specify the sensitivity analyser
analyser <- mtkNativeAnalyser("Regression", information=list(nboot=20))

## Specify the workflow
ishiReg <- new("mtkExpWorkflow", expFactors=Ishigami.factors,
               processesVector=c(
                  design=designer,
                  evaluate=model,
                  analyze=analyser)
               )

## Run and report the results
run(ishiReg)
summary(ishiReg)

mtkIshigamiEvaluator-class

The mtkIshigamiEvaluator class

Description

The mtkIshigamiEvaluator class is a sub-class of the class mtkEvaluator used to manage the simulation of the model Ishigami.

Class Hierarchy

Parent classes: mtkEvaluator

Direct Known Subclasses:

Constructor

mtkIshigamiEvaluator signature()

Slots

name: (character) always takes the string "evaluate".
protocol: (character) a string to name the protocol used to run the process: http, system, R, etc. Here, it takes the character "R".
site: (character) a string to indicate where the service is located. Here, it always takes the string "mtk".
service: (character) a string to name the service to invoke. Here, it always takes the string "Ishigami".
parameters: (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service. The "Ishigami" model does not need parameters.
ready: (logical) a logical to tell if the process is ready to run.
state: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.

result: (ANY) a data holder to hold the results produced by the process

Methods

setName signature(this = "mtkIshigamiEvaluator", name = "character"): non useful, method inherited from the parent class.

setParameters signature(this = "mtkIshigamiEvaluator", f = "vector"): Assigns new parameters to the process.

getParameters signature(this = "mtkIshigamiEvaluator"): Returns the parameters as a named list.

is.ready signature( = "mtkIshigamiEvaluator"): Tests if the process is ready to run.

setReady signature(this = "mtkIshigamiEvaluator", switch = "logical"): Makes the process ready to run.

is.ready signature( = "mtkIshigamiEvaluator"): Tests if the results produced by the process are available.

setReady signature(this = "mtkIshigamiEvaluator", switch = "logical"): Marks the process as already executed.

getResult signature(this = "mtkIshigamiEvaluator"): Returns the results produced by the process as a \[mtkEvaluatorResult\].

data signature(this = "mtkIshigamiEvaluator"): Returns the results produced by the process as a data.frame.

serializeOn signature(this = "mtkIshigamiEvaluator"): Returns all data managed by the process as a named list.

run signature(this = "mtkIshigamiEvaluator", context= "mtkExpWorkflow"): runs the simulation.

summary signature(object = "mtkIshigamiEvaluator"): Provides a summary of the results produced by the process.

print signature(x = "mtkIshigamiEvaluator"): Prints a report of the results produced by the process.

plot signature(x = "mtkIshigamiEvaluator"): Plots the results produced by the process.

report signature(this = "mtkIshigamiEvaluator"): Reports the results produced by the process.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References

### Examples

```r
# Carry out a sensitivity analysis with the Ishigami model

## Input the factors
data(Ishigami.factors)

## Specify the experiments designer
designer <- mtkNativeDesigner("BasicMonteCarlo",
                           information=list(size=20))

## Specify the model simulator
model <- mtkIshigamiEvaluator()

## Specify the sensitivity analyser
analyzer <- mtkNativeAnalyzer("Regression", information=list(nboot=20))

## Specify the workflow
ishiReg <- new("mtkExpWorkflow", expFactors=Ishigami.factors,
               processesVector=c(
                 design=designer,
                 evaluate=model,
                 analyze=analyzer)
               )

## Run and report the results
run(ishiReg)
summary(ishiReg)
```

---

### mtkLevels

*The constructor of the class* `mtkLevels`

### Description

The constructor of the class `mtkLevels`.

### Usage

```r
mtkLevels(type = "categorical", levels=vector(), weights=numeric(0))
```

### Arguments

- **type**: a string to specify the type of the discrete distribution: categorical, qualitative, etc.
- **levels**: a vector of levels for a discrete domain.
- **weights**: a vector of numeric values used to weight the levels.
**mtkLevels-class**

**Value**

an object of the `mtkLevels` class

**Author(s)**

Juhui WANG, MIA-jouy, INRA

**Examples**

```r
# creates an object of the class mtkLevel
l1 <- mtkLevels(type="qualitative", levels = c("x", "y"), weights=c(0.5, 0.5))
l2 <- mtkLevels(levels = c("a", "b", "c"))
l3 <- mtkLevels(levels = c("a", "b", "c"), weights=c(3, 5, 3))
```

**Description**

The `mtkLevels` class is a class used to manage the weighting levels associated with a factor’s domain.

**Class Hierarchy**

**Parent classes :**

**Direct Known Subclasses :**

**Constructor**

```r
mtkLevels signature(type = "categorical", levels=vector(), weights=numeric(0))
```

**Slots**

- **type**: (character) a string to give the type of the discrete distribution such as 'categorical', 'qualitative', etc.
- **levels**: (vector) a vector to specify the levels.
- **weights**: (numeric) a numeric vector used to weight the levels.

**Methods**

- **getType** signature(this = "mtkLevels"): Returns the type of the discrete distribution such as 'categorical', 'qualitative', etc.
- **setType** signature(this = "mtkLevels", type="character"): Assigns a new type to the underlying object.
- **getLevels** signature(this = "mtkLevels"): Returns the vector of the levels.
- **setLevels** signature(this = "mtkLevels", levels = "vector"): Assigns a new vector to the levels.
getWeights signature(this = "mtkLevels"): Returns the vector of the weights.
setWeights signature(this = "mtkLevels", weights = "numeric"): Assigns new vector to the weight.
print signature(x = "mtkLevel"): Prints a summarized report about the underlying object of the class mtkLevels.
summary signature(object = "mtkLevel"): Gives a summary about the underlying object.
show signature(object = "mtkLevel"): Displays informations about the underlying object.

Author(s)
Juhui WANG, MIA-jouy, INRA

Examples
# Create an object of the class 'mtkLevels'

l <- mtkLevels(type='categorical', levels=seq(1:3), weight=rep(0.33, 3))

# Set the levels' name to ('a', 'b', 'c')
setLevels(l, levels=c('a', 'b', 'c'))

mtkMorrisAnalyser The constructor of the class mtkMorrisAnalyser

Description
The constructor

Usage
mtkMorrisAnalyser(mtkParameters = NULL, listParameters = NULL)

Arguments
mtkParameters a vector of [mtkParameter] holding the parameters necessary to run the process.
listParameters a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

Value
an object of the mtkMorrisAnalyser class

References
See Also

help(morris, sensitivity) and help(Morris)

Examples

```r
## Sensitivity analysis of the "Ishigami" model with the "Morris" method

# Generate the factors
data(Ishigami.factors)

# Build the processes and workflow:
# 1) the design process
exp1.designer <- mtkMorrisDesigner(listParameters = list(r=20, type="oat", levels=4, grid.jump=2))

# 2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process
exp1.analyser <- mtkMorrisAnalyser()

# 4) the workflow
exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors, processesVector = c(design=exp1.designer, evaluate=exp1.evaluator, analyze=exp1.analyser))

# Run the workflow and report the results.
run(exp1)
print(exp1)
```

mtkMorrisAnalyser-class

The `mtkMorrisAnalyser` class

Description

The `mtkMorrisAnalyser` class is a sub-class of the class `mtkAnalyser`. It implements the sensitivity analysis method `Morris` and provides all the slots and methods defined in the class `mtkAnalyser`.

Class Hierarchy

Parent classes: `mtkAnalyser`

Direct Known Subclasses:
mtkMorrisAnalyser-class

Constructor

```r
tkMorrisAnalyser signature(mtkParameters = NULL, listParameters = NULL)
```

Slots

- **name**: (character) always takes the string "analyze".
- **protocol**: (character) always takes the string "R".
- **site**: (character) always takes the string "mtk".
- **service**: (character) always takes the string "Morris".
- **parameters**: (vector) a vector of `mtkParameter` containing the parameters to pass while calling the service.
- **ready**: (logical) a logical to tell if the process is ready to run.
- **state**: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.
- **result**: (ANY) a data holder to hold the results produced by the process

Methods

- **setName** signature(this = "mtkMorrisAnalyser", name = "character"): Not used, method inherited from the parent class.
- **setParameters** signature(this = "mtkMorrisAnalyser", f = "vector"): Assigns new parameters to the process.
- **getParameters** signature(this = "mtkMorrisAnalyser"): Returns the parameters as a named list.
- **is.ready** signature( = "mtkMorrisAnalyser"): Tests if the process is ready to run.
- **setReady** signature(this = "mtkMorrisAnalyser", switch = "logical"): Makes the process ready to run.
- **is.ready** signature( = "mtkMorrisAnalyser"): Tests if the results produced by the process are available.
- **setReady** signature(this = "mtkMorrisAnalyser", switch = "logical"): Marks the process as already executed.
- **getResult** signature(this = "mtkMorrisAnalyser"): Returns the results produced by the process as a `mtkMorrisAnalyzerResult`.
- **getData** signature(this = "mtkMorrisAnalyser"): Returns the results produced by the process as a data.frame.
- **serializeOn** signature(this = "mtkMorrisAnalyser"): Returns all data managed by the process as a named list.
- **run** signature(this = "mtkMorrisAnalyser", context= "mtkExpWorkflow"): Runs the process to generate the results.
- **summary** signature(object = "mtkMorrisAnalyser"): Provides a summary of the results produced by the process.
- **print** signature(x = "mtkMorrisAnalyser"): Prints a report of the results produced by the process.
- **plot** signature(x = "mtkMorrisAnalyzer"): Plots the results produced by the process.
- **report** signature(this = "mtkMorrisAnalyser"): Reports the results produced by the process.
References


See Also

help(morris, sensitivity) and help(Morris)

Examples

```r
## Sensitivity analysis of the "Ishigami" model with the "Morris" method

data(Ishigami.factors)

# Generate the factors
data(Ishigami.factors)

# Build the processes and workflow:

# 1) the design process
exp1.designer <- mtkMorrisDesigner( listParameters
   = list(r=20, type="oat", levels=4, grid.jump=2))

# 2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process
exp1.analyser <- mtkMorrisAnalyser()

# 4) the workflow
exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors, processesVector = c(design=exp1.designer, evaluate=exp1.evaluator, analyze=exp1.analyser))

# Run the workflow and report the results.
run(exp1)
print(exp1)
```

mtkMorrisAnalyserResult

The constructor of the class mtkMorrisAnalyserResult

Description

The constructor
**Usage**

```r
mtkMorrisAnalyserResult(main, information=NULL)
```

**Arguments**

- `main`: a data.frame holding the results of the sensitivity analysis produced by the analyser.
- `information`: a named list containing the information about the managed data.

**Value**

an object of the `mtkMorrisAnalyserResult` class

**Author(s)**

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

**References**


**Examples**

```r
# See examples from the help of the method: help(Morris)
```

---

**Description**

A class to collect the results of the sensitivity analysis produced by the analyser implementing the method Morris.

**Class Hierarchy**

Parent classes: `mtkAnalyserResult`

Direct Known Subclasses:

**Constructor**

```r
mtkMorrisAnalyserResult signature(main, information=NULL)
```
Slots

- **main**: (data.frame) a data.frame holding the results produced by the "Morris" analyser.
- **information**: (NULL) a named list containing optional information about the managed data.

Methods

- **summary** signature(object = "mtkMorrisAnalyserResult"): Provides a summary of the results produced by the analyser.
- **print** signature(x = "mtkMorrisAnalyserResult"): Prints a report of the results produced by the analyser.
- **plot** signature(x = "mtkMorrisAnalyserResult"): Plots the results produced by the analyser.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

```r
# See examples from the help of the method: help(Morris)
```

Description

The constructor

Usage

```r
mtkMorrisDesigner(mtkParameters = NULL, listParameters = NULL)
```

Arguments

- **mtkParameters** a vector of [mtkParameter] representing the parameters necessary to run the process.
- **listParameters** a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.
Value

an object of the `mtkMorrisDesigner` class

References


See Also

help(morris, sensitivity) and help(Morris)

Examples

```r
## Sensitivity analysis of the "Ishigami" model with the "Morris" method

# Generate the factors
data(Ishigami.factors)

# Build the processes and workflow:

# 1) the design process
exp1.designer <- mtkMorrisDesigner( listParameters = list(r=20, type="oat", levels=4, grid.jump=2))

# 2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process
exp1.analyser <- mtkMorrisAnalyser()

# 4) the workflow
exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors, 
                       processesVector = c(design=exp1.designer, 
                          evaluate=exp1.evaluator, analyze=exp1.analyser))

# Run the workflow and report the results.
run(exp1)
print(exp1)
```
Description

The mtkMorrisDesigner class is a sub-class of the class mtkDesigner. It implements the method Morris and provides all the slots and methods defined in the class mtkDesigner.

Class Hierarchy

Parent classes: mtkDesigner
Direct Known Subclasses:

Constructor

mtkMorrisDesigner signature(mtkParameters = NULL, listParameters = NULL)

Slots

name: (character) always takes the string "design".
protocol: (character) always takes the string "R".
site: (character) always takes the string "mtk".
service: (character) always takes the string "Morris".
parameters: (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.
ready: (logical) a logical to tell if the process is ready to run.
state: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.
result: (ANY) a data holder to hold the results produced by the process

Methods

setName signature(this = "mtkMorrisDesigner", name = "character"): Not used, method inherited from the parent class.
setParameters signature(this = "mtkMorrisDesigner", f = "vector"): Assigns new parameters to the process.
getParameters signature(this = "mtkMorrisDesigner"): Returns the parameters as a named list.
is.ready signature( = "mtkMorrisDesigner"): Tests if the process is ready to run.
setReady signature(this = "mtkMorrisDesigner", switch = "logical"): Makes the process ready to run.
is.ready signature( = "mtkMorrisDesigner"): Tests if the results produced by the process are available.
setReady signature(this = "mtkMorrisDesigner", switch = "logical"): Marks the process as already executed.

getResult signature(this = "mtkMorrisDesigner"): Returns the results produced by the process as a [mtkMorrisDesignerResult].

getData signature(this = "mtkMorrisDesigner"): Returns the results produced by the process as a data.frame.

serializeOn signature(this = "mtkMorrisDesigner"): Returns all data managed by the process as a named list.

run signature(this = "mtkMorrisDesigner", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.

summary signature(object = "mtkMorrisDesigner"): Provides a summary of the results produced by the process.

print signature(x = "mtkMorrisDesigner"): Prints a report of the results produced by the process.

plot signature(x = "mtkMorrisDesigner"): Plots the results produced by the process.

report signature(this = "mtkMorrisDesigner"): Reports the results produced by the process.

References


See Also

help(morris, sensitivity) and help(Morris)

Examples

```r
## Sensitivity analysis of the "Ishigami" model with the "Morris" method

data(Ishigami.factors)

# Generate the factors

# Build the processes and workflow:

# 1) the design process
exp1.designer <- mtkMorrisDesigner(listParameters = list(r=20, type="oat", levels=4, grid.jump=2))

# 2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process
exp1.analyser <- mtkMorrisAnalyser()

# 4) the workflow
```
mtkMorrisDesignerResult

The constructor of the class mtkMorrisDesignerResult

Description
The constructor

Usage
mtkMorrisDesignerResult(main, information=NULL)

Arguments
main a data.frame holding the experimental design produced by the designer.
information a named list containing the information about the managed data.

Value
an object of the mtkMorrisDesignerResult class

Author(s)

References

Examples

# See examples from the help of the method: help(Morris)
mtkMorrisDesignerResult-class

The mtkMorrisDesignerResult class

Description

A class to collect the experimental design produced by the designer implementing the method Morris.

Class Hierarchy

Parent classes: mtkDesignerResult

Direct Known Subclasses:

Constructor

mtkMorrisDesignerResult signature(main, information=NULL)

Slots

main: (data.frame) a data.frame holding the experimental design produced by the designer.

information: (NULL) a named list containing optional information about the managed data.

Methods

summary signature(object = "mtkMorrisDesignerResult"): Provides a summary of the experimental design produced by the designer.

print signature(x = "mtkMorrisDesignerResult"): Prints a report of the experimental design produced by the designer.

plot signature(x = "mtkMorrisDesignerResult"): Plots the experimental design produced by the designer.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# See examples from the help of the method: help(Morris)
**Description**

The constructor.

**Usage**

```r
mtkNativeAnalyser(analyze=NULL, X=NULL, information=NULL)
```

**Arguments**

- **analyze**: NULL, an R function or a string to specify the analyser to use.
- **X**: NULL or a data.frame to load the results produced off-line.
- **information**: a named list to provide with supplementary information about the analysis produced off-line or the parameters used by the analyser.

**Value**

an object of the `mtkNativeAnalyser` class

**Details**

We can construct an object of the `mtkNativeAnalyser` class in three manners:

- the analyser is provided within the package "mtk" The argument "analyze" takes a string giving the name of the method used to carry out the sensitivity analysis, the argument "information" gives the list of parameters used by the analyser.
- the analyser is available as an R function implemented outside the package "mtk" The argument "analyze" takes an R function implementing the analyser, the argument "information" may be used to give supplementary information about the R function.
- the results of the sensitivity analysis are already available as a data.frame. We use "mtk" only for reporting. The argument "X" takes the data.frame holding the available results, and the argument "information" may be omitted or simply used to give supplementary information about the analysis.

More examples for using this class, see `?class(mtkNativeEvaluator)`.

**Author(s)**

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

**References**

See Also

?class(mtkNativeEvaluator)

Examples

# Create a native analyser with the method "Morris" implemented in the package "mtk"

anlyser <- mtkNativeAnalyser(
  analyze="Morris",
  information=list(nboot=20))

---

**mtkNativeAnalyser-class**

*The mtkNativeAnalyser class*

---

**Description**

The `mtkNativeAnalyser` class is a sub-class of the class `mtkAnalyser` used to manage the sensitivity analysis task implemented locally (i.e. tasks don't need to call services from the Web). It provides all the slots and methods defined in the class `mtkAnalyser`.

**Class Hierarchy**

- **Parent classes**: `mtkAnalyser`
- **Direct Known Subclasses**:

**Constructor**

`mtkNativeAnalyser` signature(analyze=NULL, X=NULL, information=NULL)

**Slots**

- `analyze`: (*ANY*) a string, an R function, or NULL to inform the method to use for the sensitivity analysis.
- `name`: (*character*) always takes the string "analyze".
- `protocol`: (*character*) a string to name the protocol used to run the process: http, system, R, etc. Here, it always takes "R".
- `site`: (*character*) a string to indicate where the service is located.
- `service`: (*character*) a string to name the service to invoke. Here, it may be a R function or a method implemented in the package "mtk".
- `parameters`: (*vector*) a vector of [mtkParameter] containing the parameters to pass while calling the service.
- `ready`: (*logical*) a logical to tell if the process is ready to run.
- `state`: (*logical*) a logical to tell if the results produced by the process are available and ready to be consumed.
- `result`: (*ANY*) a data holder to hold the results produced by the process
Methods

setname signature(this = "mtkNativeAnalyser", name = "character"): Not used here, method inherited from the parent class.

setparameters signature(this = "mtkNativeAnalyser", f = "vector"): Assigns new parameters to the process.

getparameters signature(this = "mtkNativeAnalyser"): Returns the parameters as a named list.

is.ready signature( = "mtkNativeAnalyser"): Tests if the process is ready to run.

setReady signature(this = "mtkNativeAnalyser", switch = "logical"): Makes the process ready to run.

is.ready signature( = "mtkNativeAnalyser"): Tests if the results produced by the process are available.

setReady signature(this = "mtkNativeAnalyser", switch = "logical"): Marks the process as already executed.

getResult signature(this = "mtkNativeAnalyser"): Returns the results produced by the process as a [mtkAnalyserResult].

getdata signature(this = "mtkNativeAnalyser"): Returns the results produced by the process as a data.frame.

serializeOn signature(this = "mtkNativeAnalyser"): Returns all data managed by the process as a named list.

run signature(this = "mtkNativeAnalyser", context= "mtkExpWorkflow"): Runs the Analyser.

summary signature(object = "mtkNativeAnalyser"): Provides a summary of the results produced by the process.

print signature(x = "mtkNativeAnalyser"): Prints a report of the results produced by the process.

plot signature(x = "mtkNativeAnalyser"): Plots the results produced by the process.

report signature(this = "mtkNativeAnalyser"): Reports the results produced by the process.

Details

We can construct an object of the mtkNativeAnalyser class from the following situations:

1. The analyser is provided within the package "mtk";
2. The analyser is provided as an R function implemented outside the package "mtk": If so, the R function must produce a result as a named list with two elements: X and information, where X is a data.frame containing the analysis result and information is a named list containing supplementary information about the analysis process.
3. The results of the model exploration are produced off-line and available as a data.frame. We just want to use the "mtk" package for reporting.

For detail uses, see examples from help(mtkNativeEvaluator).

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr
References


Examples

# Create a native analyser with the method "Morris" implemented in the package "mtk"

```r
analyser <- mtkNativeAnalyser(
analyze="Morris",
information=list(nboot=20))
```

#### mtkNativeDesigner

The constructor of the class mtkNativeDesigner

**Description**

The constructor.

**Usage**

```r
mtkNativeDesigner(design=NULL, X=NULL, information=NULL)
```

**Arguments**

- `design` NULL, an R function or a string to specify the method used to generate the experiments design.
- `X` NULL or a data.frame to load the experimental design produced off-line.
- `information` a named list to provide with supplementary information about the experimental design produced off-line or the parameters used by the designer.

**Value**

an object of the mtkNativeDesigner class

**Details**

We can construct an object of the mtkNativeDesigner class from the following situations:

- the designer is provided within the package "mtk" The argument "design" takes a string giving the method used to generate the experimental design, and the argument "information" gives the list of parameters used by the designer. e.g. designer <- mtkNativeDesigner(design="Morris", information = list(nboot=20)).
- the designer is provided with an R function implemented outside the package "mtk" The argument "design" takes the R function, the argument "information" may be used to give supplementary information about the R function.
• the experimental design is produced off-line and available as a data.frame. The argument "design" is not used, the argument "X" takes the data.frame holding the available experimental design, and the argument "information" may be omitted or simply used to give supplementary information about the method used to generate the experimental design. e.g. Designer <- mtkNativeDesigner(X = mcDesign, information = list(sampling = "Monte-Carlo")).

For details uses, see examples from help(mtkNativeEvaluator).

Author(s)
Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References

See Also
help(mtkNativeEvaluator)

Examples

# Create a native designer with the method "Morris"
# implemented in the package "mtk"

designer <- mtkNativeDesigner(design="Morris", information=list(size=20))

mtkNativeDesigner-class

The mtkNativeDesigner class

Description
The mtkNativeDesigner class is a sub-class of the class mtkDesigner used to manage the sampling task implemented locally (i.e. tasks don’t need to call services from the Web). By object inheriting, it provides all the slots and methods defined in the class mtkDesigner.

Class Hierarchy

Parent classes: mtkDesigner

Direct Known Subclasses:

Constructor

mtkNativeDesigner signature(design=NULL, X=NULL, information=NULL)
Slots

- **design**: (ANY) a string, an R function, or NULL to inform the designer to use.
- **name**: (character) always takes the string "design".
- **protocol**: (character) a string to name the protocol used to run the process: http, system, R, etc. Here, it always takes "R".
- **site**: (character) a string to indicate where the service is located. Here, it gives no sense.
- **service**: (character) a string to name the service to invoke.
- **parameters**: (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.
- **ready**: (logical) a logical to tell if the process is ready to run.
- **state**: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.
- **result**: (ANY) a data holder to hold the results produced by the process

Methods

- **setName** signature(this = "mtkNativeDesigner", name = "character"): Method inherited from the parent class. It gives no sense here.
- **setParameters** signature(this = "mtkNativeDesigner", f = "vector"): Assigns new parameters vector to the process.
- **getParameters** signature(this = "mtkNativeDesigner"): Returns the parameters vector as a named list.
- **is.ready** signature( = "mtkNativeDesigner"): Tests if the process is ready to run.
- **setReady** signature(this = "mtkNativeDesigner", switch = "logical"): Makes the process ready to run.
- **is.ready** signature( = "mtkNativeDesigner"): Tests if the results produced by the process are available.
- **setReady** signature(this = "mtkNativeDesigner", switch = "logical"): Marks the process as already executed.
- **getResult** signature(this = "mtkNativeDesigner"): Returns the results produced by the process as a [mtkDesignerResult].
- **getData** signature(this = "mtkNativeDesigner"): Returns the results produced by the process as a data.frame.
- **serializeOn** signature(this = "mtkNativeDesigner"): Returns all data managed by the process as a named list.
- **run** signature(this = "mtkNativeDesigner", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.
- **summary** signature(object = "mtkNativeDesigner"): Provides a summary of the results produced by the process.
- **print** signature(x = "mtkNativeDesigner"): Prints a report of the results produced by the process.
- **plot** signature(x = "mtkNativeDesigner"): Produces a graphical report of the results produced by the process.
- **report** signature(this = "mtkNativeDesigner"): Reports the results produced by the process.
mtnativeEvaluator

Details

We can construct an object of the mtkNativeDesigner class from the following situations:

1. The designer is provided within the package "mtk";
2. The designer is provided as an R function implemented outside the package "mtk"; If so, the R function must produce a result as a named list with two elements: X and information, where X is a data.frame containing the analysis result and information is a named list containing supplementary information about the analysis process.
3. The experiments design is produced off-line and available as a data.frame. We just want to use the "mtk" package for reporting.

For detail uses, see examples from help(mtkNativeEvaluator).

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# Create a native designer with the method "Morris"
# implemented in the package "mtk"

designer <- mtkNativeDesigner(
    design="Morris",
    information=list(size=20)
)

mtkNativeEvaluator  The constructor of the class mtkNativeEvaluator

Description

The constructor.

Usage

mtkNativeEvaluator(model=NULL, Y=NULL, information=NULL)
Arguments

model
NULL, an R function or a string to specify the model to simulate.

Y
NULL or a data.frame to load the results of model simulation produced off-line.

information
a named list to provide with supplementary information about the simulation produced off-line or the parameters used by the evaluator.

Value
an object of the mtkNativeEvaluator class

Details

We can construct an object of the mtkNativeEvaluator class from the following situations:

• The model is provided within the package "mtk"The argument "model" takes a string giving the model to simulate, and the argument "information" gives the list of parameters used for the model simulation. e.g. model <- mtkNativeEvaluator( model="Ishigami").

• The model is provided with an R function implemented outside the package "mtk"The argument "model" takes the R function, the argument "information" may be used to give supplementary information about the R function.

• The simulation results are produced off-line and available as a data.frameThe argument "model" is not used, the argument "Y" takes the data.frame holding the model simulation, and the argument "information" may be omitted or simply used to give supplementary information about the simulation process. e.g. model <- mtkNativeDesigner( Y = simulatedData, information = list(model = "Ishigami"))

For details uses, see examples from ?class(mtkNativeEvaluator).

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


See Also

?class(mtkNativeEvaluator)
Examples

```r
## 1) Create a model simulation with the model "Ishigami" implemented in the package "mtk"
mtkEvaluator <- mtkNativeEvaluator(model="Ishigami")

## 2) Create a model simulation with a R function implemented outside the package "mtk"

# a) Create a R function to represent the model of population

ME <- function(K, Y0, a, t=5, ...) {
  res <- exp(-a*t)
  res <- Y0+res*(K-Y0)
  res <- K*Y0/res
  out <- as.integer(res)
  return(out)
}

# b) Do the sensitivity analysis for the function "ME"

K <- make.mtkFactor(name="K", nominal=400, distribName="unif",
                     distribPara=list(min=100, max=1000))
Y0 <- make.mtkFactor(name="Y0", nominal=20, distribName="unif",
                     distribPara=list(min=1, max=40))
a <- make.mtkFactor(name="a", nominal=0.1, distribName="unif",
                     distribPara=list(min=0.05, max=0.2))
factors <- mtkExpFactors(list(K,Y0,a))

plan <- mtkNativeDesigner ("BasicMonteCarlo",
                            information=c(size=500))

model <- mtkNativeEvaluator(model=ME, information=c(t=5))

index<- mtkNativeAnalyzer("Regression", information=c(nboot=20) )

expt <- mtkExpWorkflow( expFactors=factors,
                        processesVector=c( design= plan,
                                          evaluate= model,
                                          analyze= index) )
run(expt)
summary(expt)

## 3) Import the results of model simulation produced off-line into
## an object of mtkNativeEvaluator

data <- data.frame()
model <- mtkNativeEvaluator(Y=data,
                            information = list(model="Ishigami"))
```
The mtkNativeEvaluator class

Description

The mtkNativeEvaluator class is a sub-class of the class mtkEvaluator used to manage the simulation task implemented locally (i.e. tasks don’t need to call services from the Web). It provides all the slots and methods defined in the class mtkEvaluator.

Class Hierarchy

Parent classes: mtkEvaluator
Direct Known Subclasses:

Constructor

mtkNativeEvaluator signature(model=NULL, Y=NULL, information=NULL)

Slots

model: (ANY) a string, an R function, or NULL to inform the model to simulate.
name: (character) always takes the string "evaluate".
protocol: (character) a string to name the protocol used to run the process: http, system, R, etc. Here, it always takes "R".
site: (character) a string to indicate where the service is located. Here, it always takes "mtk".
service: (character) a string to name the service to invoke.
parameters: (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.
ready: (logical) a logical to tell if the process is ready to run.
state: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.
result: (ANY) a data holder to hold the results produced by the process

Methods

setName signature(this = "mtkNativeEvaluator", name = "character"): Not used, method inherited from the parent class.
setParameters signature(this = "mtkNativeEvaluator", f = "vector"): Assigns new parameters to the process.
getParameters signature(this = "mtkNativeEvaluator"): Returns the parameters as a named list.
is.ready signature( = "mtkNativeEvaluator"): Tests if the process is ready to run.
setReady signature(this = "mtkNativeEvaluator", switch = "logical"): Makes the process ready to run.
**mtkNativeEvaluator-class**

- **is.ready** signature( = "mtkNativeEvaluator"): Tests if the results produced by the process are available.
- **setReady** signature(this = "mtkNativeEvaluator", switch = "logical"): Marks the process as already executed.
- **getResult** signature(this = "mtkNativeEvaluator"): Returns the results produced by the process as a `mtkEvaluatorResult`.
- **getdata** signature(this = "mtkNativeEvaluator"): Returns the results produced by the process as a data.frame.
- **serializeOn** signature(this = "mtkNativeEvaluator"): Returns all data managed by the process as a named list.
- **run** signature(this = "mtkNativeEvaluator", context= "mtkExpWorkflow"): runs the simulation.
- **summary** signature(object = "mtkNativeEvaluator"): Provides a summary of the results produced by the process.
- **print** signature(x = "mtkNativeEvaluator"): Prints a report of the results produced by the process.
- **plot** signature(x = "mtkNativeEvaluator"): Plots the results produced by the process.
- **report** signature(this = "mtkNativeEvaluator"): Reports the results produced by the process.

**Details**

We can construct an object of the `mtkNativeEvaluator` class from the following situations: 1) 2) 3) the experimental design is produced off-line and available as a data.frame.

We can construct an object of the `mtkNativeEvaluator` class from the following situations:

1. The evaluator is provided within the package "mtk";
2. The evaluator is provided as an R function outside the package "mtk";
3. The simulation is carried out off-line. We just want to use the "mtk" package for reporting.

**Author(s)**

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

**References**


**Examples**

```r
# 1) Create a model simulation with the model "Ishigami" implemented in the package "mtk"
evaluator <- mtkNativeEvaluator(model="Ishigami")
```

```r
# 2) Create a model simulation with a R function implemented outside the package "mtk"
# a) Create a R function to represent the model of population
```
ME <- function(K, Y0, a, t=5, ...){
  res <- exp(-a*t)
  res <- Y0+res*(K-Y0)
  res <- K*Y0/res
  out <- as.integer(res)
  return(out)
}
# b) Do the sensitivity analysis for the function "ME"
K <- make.mtkFactor(name="K", nominal=400, distribName="unif",
  distribPara=list(min=100, max=1000))
Y0 <- make.mtkFactor(name="Y0", nominal=20, distribName="unif",
  distribPara=list(min=1, max=40))
a <- make.mtkFactor(name="a", nominal=0.1, distribName="unif",
  distribPara=list(min=0.05, max=0.2))
factors <- mtkExpFactors(list(K,Y0,a))

plan <- mtkNativeDesigner ("BasicMonteCarlo",
  information=c(size=500))

model <- mtkNativeEvaluator(model=ME, information=c(t=5))

index<- mtkNativeAnalyser("Regression", information=c(nboot=20))

expt <- mtkExpWorkflow( expFactors=factors, 
  processesVector=c( 
    design= plan, 
    evaluate= model, 
    analyze= index) 
)
run(expt)
summary(expt)

## 3) Import the results of model simulation produced off-line into an object of mtkNativeEvaluator
data <- data.frame()
model <- mtkNativeEvaluator(Y=data, 
  information = list(model="Ishigami"))

---

**mtkParameter**  
*The constructor of the class mtkParameter*

**Description**

The constructor of the class mtkParameter. See also make.mtkParameterList
Usage
mtkParameter(name='unknown', type='logical', val=NULL)

Arguments
- name (character) the name of the parameter.
- type (character) the type of the parameter such as 'numeric', 'double', 'logical', etc..
- val (ANY) the value of the parameter.

Value
an object of the mtkParameter class

Author(s)
Juhui WANG, MIA-jouy, INRA

Examples
# Create an object of the 'mtkParameter' class.
p <- mtkParameter(name="x", type="double", val=0.0)

# We usually use the 'make.mtkParameterList()' function to define
# a list of 'mtkParameter' instead of the constructor
# of the 'mtkParameter' class
flist <- make.mtkParameterList(x=list(min=-1,max=+1))
Slots

name: (character) the name of the parameter.
type: (character) the type of the parameter.
val: (ANY) the value of the parameter.

Methods

getName signature(this = "mtkParameter"): Returns the value of the slot "name".
getValue signature(this = "mtkParameter"): Returns the value of the slot "val".
getType signature(this = "mtkParameter"): Returns the value of the slot "type".
setName signature(this = "mtkParameter", name="character"): Gives a new value to the slot "name".
setValue signature(this = "mtkParameter", val="ANY"): Gives a new value to the slot "val".
setType signature(this = "mtkParameter", type="character"): Gives a new value to the slot "type".
show signature(object = "mtkParameter"): Prints a report of the data managed by the underlying object.
print signature(x = "mtkParameter"): Prints the information managed by the underlying object.

Author(s)

Juhui WANG, MIA-jouy, INRA

Examples

# Create an object of the 'mtkParameter' class.

p <- mtkParameter(name="x", type="double", val=0.0)

# We usually use the 'make.mtkParameterList()' function to define a list of # 'mtkParameter' instead of the constructor # of the 'mtkParameter' class

plist <- make.mtkParameterList(list(min=-1,max=+1,shape="hello"))
**Arguments**

xmlPath       a string to specify the XML file to parse.

**Value**

an object of the `mtkParsor` class

**Author(s)**

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

**References**


**Examples**

```r
# Create a parsor with the file "inst/extdata/WWDM.xml".

# Specify the XML file's name
xmlFile <- "WWDM_morris.xml"

# find where the examples are held.  
# This is only necessary for the example since the system does  
# not know where the file "WWDM.xml" is kept.
xmlFilePath <- paste(path.package("mtk", quiet = TRUE),  
                     "/extdata/", xmlFile, sep = "")

## Create a parsor from the xml file
parson <- mtkParsor(xmlFilePath)

# Create an empty workflow.
workflow <- mtkExpWorkflow()

# Parse the XML file and initialize the workflow  
# with the data extracted from the XML file.  
run(parson, workflow)

# Run the workflow and report the results of the sensitivity analysis
run(workflow)
summary(workflow)
```
mtkParsor-class

The mtkParsor class

Description

The mtkParsor class is the main class used to parse the XML files used in the "mtk" package. It provides a generic way to communicate with the plate-form of model simulation.

Class Hierarchy

Parent classes:

Direct Known Subclasses:

Constructor

mtkParsor signature(xmlPath="")

Slots

xmlPath: (character) the XML file’s path and name.

Methods

setXMLFilePath signature(this = "mtkParsor", xmlPath = "character"): Sets the xml File.

run signature(this = "mtkParsor", context = "mtkExpWorkflow"): Parses the XML file and fills the workflow defined in the "context" argument with the data extracted from the XML file.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# Create a parsor with the file "inst/extdata/WWDM.xml".

# Specify the XML file's name
xmlFile <- "WWDM_morris.xml"

# find where the examples are held.
# This is only necessary for the example since the system does
# not know where the file "WWDM.xml" is kept.
xmlFilePath <- paste(path.package("mtk", quiet = TRUE), 
"/extdata/", xmlFile, sep = "")

## Create a parser from the xml file
parson <- mtkParsor(xmlFilePath)

# Create an empty workflow.
workflow <- mtkExpWorkflow()

# Parse the XML file and initialize the workflow
# with the data extracted from the XML file.
run(parson, workflow)

# Run the workflow and report the results of the sensitivity analysis
run(workflow)
summary(workflow)

---

### mtkPLMMAalyser

*The constructor of the class mtkPLMMAalyser*

#### Description

The constructor

#### Usage

```r
mtkPLMMAalyser(mtkParameters = NULL, listParameters = NULL)
```

#### Arguments

- `mtkParameters` a vector of `[mtkParameter]` representing the parameters necessary to run the process.
- `listParameters` a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

#### Value

an object of the `mtkPLMMAalyser` class

#### Author(s)

Rober Faivre, MIA-Toulouse, INRA, Contact: faivre@toulouse.inra.fr, Juhui WANG, MIA-Jouy, Inra,
mtkPLMMAnalyser-class

References


Examples

# see examples with help(PLMM)

---

mtkPLMMAnalyser-class  The mtkPLMMAnalyser class

Description

The mtkPLMMAnalyser class is a sub-class of the class mtkAnalyser. It implements the sensitivity analysis method PLMM and provides all the slots and methods defined in the class mtkAnalyser.

Class Hierarchy

Parent classes: mtkAnalyser

Direct Known Subclasses:

Constructor

mtkPLMMAnalyser  signature(mtkParameters = NULL, listParameters = NULL)

Slots

name: (character) always takes the string "analyze".
protocol: (character) always takes the string "R".
site: (character) always takes the string "mtk".
service: (character) always takes the string "PLMM".
parameters: (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.
ready: (logical) a logical to tell if the process is ready to run.
state: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.
result: (ANY) a data holder to hold the results produced by the process
Methods

**setName** signature(this = "mtkPLMMAnalyser", name = "character"): Not used, method inherited from the parent class.

**setParameters** signature(this = "mtkPLMMAnalyser", f = "vector"): Assigns new parameters to the process.

**getParameter** signature(this = "mtkPLMMAnalyser"): Returns the parameters as a named list.

**is.ready** signature( = "mtkPLMMAnalyser"): Tests if the process is ready to run.

**setReady** signature(this = "mtkPLMMAnalyser", switch = "logical"): Makes the process ready to run.

**is.ready** signature( = "mtkPLMMAnalyser"): Tests if the results produced by the process are available.

**setReady** signature(this = "mtkPLMMAnalyser", switch = "logical"): Marks the process as already executed.

**getResult** signature(this = "mtkPLMMAnalyser"): Returns the results produced by the process as a **mtkPLMMAnalyserResult**.

**getData** signature(this = "mtkPLMMAnalyser"): Returns the results produced by the process as a data.frame.

**serializeOn** signature(this = "mtkPLMMAnalyser"): Returns all data managed by the process as a named list.

**run** signature(this = "mtkPLMMAnalyser", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.

**summary** signature(object = "mtkPLMMAnalyser"): Provides a summary of the results produced by the process.

**print** signature(x = "mtkPLMMAnalyser"): Prints a report of the results produced by the process.

**plot** signature(x = "mtkPLMMAnalyser"): Plots the results produced by the process.

**report** signature(this = "mtkPLMMAnalyser"): Reports the results produced by the process.

Author(s)

Rober Faivre, MIA-Toulouse, INRA, Contact: faivre@toulouse.inra.fr, Juhui WANG, MIA-Jouy, Inra,

References


Examples

```
# see examples with help(PLMM)
```
mtkPLMMAnalyserResult

The constructor of the class `mtkPLMMAnalyserResult`

Description

The constructor

Usage

```r
mtkPLMMAnalyserResult(main, information=NULL)
```

Arguments

- `main` a data.frame holding the results of the sensitivity analysis produced by the PLMM analyser.
- `information` a named list containing the information about the managed data.

Value

an object of the `mtkPLMMAnalyserResult` class

Author(s)

Rober Faivre, MIA-Toulouse, INRA, Contact: faivre@toulouse.inra.fr, Juhui WANG, MIA-Jouy, Inra,

References


Examples

```r
# see examples with help(PLMM)
```
mtkPLMMAnalyserResult-class

Description

A class to collect the results of the sensitivity analysis produced by the analyser implementing the method PLMM.

Class Hierarchy

- Parent classes: mtkAnalyserResult
- Direct Known Subclasses:

Constructor

mtkPLMMAnalyserResult signature(main,information=NULL)

Slots

- main: (data.frame) a data.frame holding the experimental design.
- information: (NULL) a named list containing optional information about the managed data.

Methods

- summary signature(object = "mtkPLMMAnalyserResult"): Provides a summary of the experimental design produced by the analyser.
- print signature(x = "mtkPLMMAnalyserResult"): Prints a report of the experimental design produced by the analyser.
- plot signature(x = "mtkPLMMAnalyserResult"): Plots the experimental design produced by the analyser.

Author(s)

Rober Faivre, MIA-Toulouse, INRA, Contact: faivre@toulouse.inra.fr, Juhui WANG, MIA-Jouy, Inra,

References


Examples

# see examples with help(PLMM)
The constructor of the \texttt{mtkProcess} class

\textbf{Description}

The constructor

\textbf{Usage}

```
mtkProcess(
  name,
  protocol = "R",
  site = "mtk",
  service = "",
  parameters = NULL,
  ready = FALSE,
  state = FALSE,
  result = NULL
)
```

\textbf{Arguments}

- \texttt{name} the processing step associated with this process. It may be "design", "evaluate", or "analyze".
- \texttt{protocol} a string from "http", "system", "R" respectively representing if the process is implemented remotely, locally or as R function.
- \texttt{site} the site where the process is implemented if remotely or the package where the process is implemented if as a R function.
- \texttt{service} the service name or a system call that implements the process.
- \texttt{parameters} a vector of \texttt{mtkParameter} representing the parameters necessary to run the process.
- \texttt{ready} a logical to indicate if the process is ready to run.
- \texttt{state} a logical to indicate if the process finished running and the results are available.
- \texttt{result} an object of a class derived from \texttt{mtkResult} to hold the results produced by the process.

\textbf{Value}

an object of the \texttt{mtkProcess} class

\textbf{Details}

The \texttt{mtkProcess} class is a virtual class to manage the generic properties of processes involved in the "mtk" package.

For details uses, see examples from \texttt{help(mtkNativeDesigner)}, \texttt{help(mtkNativeEvaluator)}, \texttt{help(mtkNativeAnalyser)}.,
Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# see examples with help(mtkNativeDesigner)

mtkProcess-class  The mtkProcess class

Description

The mtkProcess is a class to represent the processes managed within the workflow. It provides a generic mechanism for conceptualizing the common behavior of the processes used in experimental design, model simulation and sensitivity analysis.

Class Hierarchy

Parent classes :

Direct Known Subclasses: mtkDesigner, mtkEvaluator, mtkAnalyser

Constructor

mtkProcess signature(name, protocol = "R", site = "mtk", service = "", parameters = NULL, ready = FALSE, state = FALSE, result = NULL)

Slots

name: (character) a string to name the step of the analysis: "design", "evaluate" or "analyze".
protocol: (character) a string to name the protocol used to run the process: "http", "system", "R", etc.
site: (character) a string to indicate where the service is located: "mtk", URI, etc.
service: (character) a string to name the service to invoke.
parameters: (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.
ready: (logical) a logical to tell if the process is ready to run.
state: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.
result: (ANY) a data holder to keep the results produced by the process
Methods

**setName** signature(this = "mtkProcess", name = "character"): Gives a name to the process.

**getName** signature(this = "mtkProcess"): Returns the name of the process.

**setParameters** signature(this = "mtkProcess", f = "vector"): Assigns new parameters to the process.

**getParameters** signature(this = "mtkProcess"): Returns the parameters as a named list.

**is.ready** signature(this = "mtkProcess", switch = "logical"): Tests if the process is ready to run.

**setReady** signature(this = "mtkProcess", switch = "logical"): Makes the process ready to run.

**is.ready** signature(this = "mtkProcess", switch = "logical"): Tests if the results produced by the process are available.

**setResult** signature(this = "mtkProcess", state = "logical"): Marks the process as already executed.

**getResult** signature(this = "mtkProcess"): Returns the results produced by the process as a named list.

**getResu«t** signature(this = "mtkProcess"): Returns the results produced by the process as a data frame.

**serializeOn** signature(this = "mtkProcess"): Returns all data managed by the process as a named list.

**run** signature(this = "mtkProcess", context= "mtkExpWorkflow"): Runs the process.

**summary** signature(object = "mtkProcess", ...): Displays a summary of the results produced by the process.

**print** signature(x = "mtkProcess"): Prints a report of the results produced by the process.

**plot** signature(x = "mtkProcess", y, ...): Plots the results produced by the process.

**report** signature(this = "mtkProcess"): Reports the results produced by the process.

Details

The **mtkProcess** class is a virtual class to manage the generic properties of processes involved in the "mtk" package.

For details uses, see examples from help(mtkNativeDesigner), help(mtkNativeEvaluator), help(mtkNativeAnalyser).

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# see examples with help(mtkNativeDesigner)
mtkRandLHSDesigner

The constructor of the class mtkRandLHSDesigner

**Description**

The constructor

**Usage**

mtkRandLHSDesigner(mtkParameters = NULL, listParameters = NULL)

**Arguments**

- mtkParameters: a vector of `mtkParameter` representing the parameters necessary to run the process.
- listParameters: a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

**Value**

an object of the `mtkRandLHSDesigner` class

**See Also**

- package?lhs, help(LHS)

**Examples**

# To do, example for LHS method

mtkRandLHSDesigner-class

The mtkRandLHSDesigner class

**Description**

The mtkRandLHSDesigner class is a sub-class of the class mtkDesigner. It implements the method RandLHS and provides all the slots and methods defined in the class mtkDesigner.

**Class Hierarchy**

Parent classes: `mtkDesigner`

Direct Known Subclasses:
Constructor

**mtkRandLHSDesigner** signature(mtkParameters = NULL, listParameters = NULL)

Slots

- **name**: (character) always takes the string "design".
- **protocol**: (character) always takes the string "R".
- **site**: (character) always takes the string "mtk".
- **service**: (character) always takes the string "RandLHS".
- **parameters**: (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.
- **ready**: (logical) a logical to tell if the process is ready to run.
- **state**: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.
- **result**: (ANY) a data holder to hold the results produced by the process

Methods

- **setName** signature(this = "mtkRandLHSDesigner", name = "character"): Not used, method inherited from the parent class.
- **setParameters** signature(this = "mtkRandLHSDesigner", f = "vector"): Assigns new parameters to the process.
- **getParameters** signature(this = "mtkRandLHSDesigner"): Returns the parameters as a named list.
- **is.ready** signature( = "mtkRandLHSDesigner"): Tests if the process is ready to run.
- **setReady** signature(this = "mtkRandLHSDesigner", switch = "logical"): Makes the process ready to run.
- **is.ready** signature( = "mtkRandLHSDesigner"): Tests if the results produced by the process are available.
- **setReady** signature(this = "mtkRandLHSDesigner", switch = "logical"): Marks the process as already executed.
- **getResult** signature(this = "mtk RandLHSDesigner"): Returns the results produced by the process as a [mtkRandLHSDesignerResult].
- **get Data** signature(this = "mtkRandLHSDesigner"): Returns the results produced by the process as a data.frame.
- **serializeOn** signature(this = "mtkRandLHSDesigner"): Returns all data managed by the process as a named list.
- **run** signature(this = "mtkRandLHSDesigner", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.
- **summary** signature(object = "mtkRandLHSDesigner"): Provides a summary of the results produced by the process.
- **print** signature(x = "mtkRandLHSDesigner"): Prints a report of the results produced by the process.
- **plot** signature(x = "mtkRandLHSDesigner"): Plots the results produced by the process.
- **report** signature(this = "mtkRandLHSDesigner"): Reports the results produced by the process.
See Also

package?lsh, help(LHS)

Examples

# To do, example for LHS method
The `mtkRandLHSDesignerResult` class

**Description**

A class to collect the experimental design produced by the designer implementing the method `randLHS`.

**Class Hierarchy**

**Parent classes:** `mtkDesignerResult`

**Direct Known Subclasses:**

**Constructor**

```r
mtkRandLHSDesignerResult signature(main, information=NULL)
```

**Slots**

- `main`: (data.frame) a data.frame holding the experimental design.
- `information`: (NULL) a named list containing optional information about the managed data.

**Methods**

- `summary` signature(object = "mtkRandLHSDesignerResult"): Provides a summary of the experimental design produced by the designer.
- `print` signature(x = "mtkRandLHSDesignerResult"): Prints a report of the experimental design produced by the designer.
- `plot` signature(x = "mtkRandLHSDesignerResult"): Plots the experimental design produced by the designer.

**See Also**

- `package?lsh`, `help(LHS)`

**Examples**

```r
# To do, example for LHS method
```
mtkReadFactors-methods

The mtkReadFactors method

Description

a list of factors

Usage

mtkReadFactors(file, path)

Arguments

file the name of the file to read.
path the path to the file to read.

Value

an object of the class mtkDomain

Author(s)

Hervé Richard, BioSP, INRA, Domaine Saint paul, 84914 Avignon Cedex 9

Examples

# see examples for the \code{\linkS4class(mtkExpFactors)} class.

mtkRegressionAnalyser  The constructor of the class mtkRegressionAnalyser

Description

The constructor

Usage

mtkRegressionAnalyser(
  mtkParameters = NULL,
  listParameters = NULL
)
Arguments

mtkParameters a vector of \texttt{[mtkParameter]} representing the parameters necessary to run the process.

listParameters a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

Value

an object of the \texttt{mtkRegressionAnalyser} class

See Also

\texttt{help(morris, sensitivity)} and \texttt{help(Regression)}

Examples

```r
## Sensitivity analysis of the "Ishigami" model with the "Monte-Carlo" and "Regression" methods

# Generate the factors
data(Ishigami.factors)

# Build the processes and workflow:
# 1) the design process
exp.designer <- mtkBasicMonteCarloDesigner (listParameters=list(size=20))

# 2) the simulation process
exp.evaluator <- mtkIshigamiEvaluator()

# 3) the analysis process
exp.analyser <- mtkRegressionAnalyser(listParameters=list(nboot=20))

# 4) the workflow
exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
                      processesVector = c(design=exp.designer,
                                          evaluate=exp.evaluator,
                                          analyze=exp.analyser))

# Run the workflow and report the results.
run(exp1)
print(exp1)
```
**Description**

The `mtkRegressionAnalyzer` class is a sub-class of the class `mtkAnalyzer`. It implements the sensitivity analysis method `Regression` and provides all the slots and methods defined in the class `mtkAnalyzer`.

**Class Hierarchy**

- **Parent classes**: `mtkAnalyzer`
- **Direct Known Subclasses**:

**Constructor**

```
mtkRegressionAnalyzer signature(mtkParameters = NULL, listParameters = NULL)
```

**Slots**

- `name` (character) always takes the string "analyze".
- `protocol` (character) always takes the string "R".
- `site` (character) always takes the string "mtk".
- `service` (character) always takes the string "Regression".
- `parameters` (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.
- `ready` (logical) a logical to tell if the process is ready to run.
- `state` (logical) a logical to tell if the results produced by the process are available and ready to be consumed.
- `result` (ANY) a data holder to hold the results produced by the process

**Methods**

- `setName` signature(this = "mtkRegressionAnalyzer", name = "character"): Not used, method inherited from the parent class.
- `setParameters` signature(this = "mtkRegressionAnalyzer", f = "vector"): Assigns new parameters to the process.
- `getParameters` signature(this = "mtkRegressionAnalyzer"): Gets the parameters as a named list.
- `is.ready` signature( = "mtkRegressionAnalyzer"): Tests if the process is ready to run.
- `setReady` signature(this = "mtkRegressionAnalyzer", switch = "logical"): Makes the process ready to run.
- `is.ready` signature( = "mtkRegressionAnalyzer"): Tests if the results produced by the process are available.
### mtkRegressionAnalyser-class

**setReady** signature(this = "mtkRegressionAnalyser", switch = "logical"): Marks the process as already executed.

**getResult** signature(this = "mtkRegressionAnalyser"): Returns the results produced by the process as a `mtkRegressionAnalyserResult`.

**getData** signature(this = "mtkRegressionAnalyser"): Returns the results produced by the process as a data.frame.

**serializeOn** signature(this = "mtkRegressionAnalyser"): Returns all data managed by the process as a named list.

**run** signature(this = "mtkRegressionAnalyser", context = "mtkExpWorkflow"): Generates the experimental design by sampling the factors.

**summary** signature(object = "mtkRegressionAnalyser"): Provides a summary of the results produced by the process.

**print** signature(x = "mtkRegressionAnalyser"): Prints a report of the results produced by the process.

**plot** signature(x = "mtkRegressionAnalyser"): Plots the results produced by the process.

**report** signature(this = "mtkRegressionAnalyser"): Reports the results produced by the process.

### See Also

`help(morris, sensitivity)` and `help(Regression)`

### Examples

```r
## Sensitivity analysis of the "Ishigami" model with the "Monte-Carlo" and "Regression" methods

# Generate the factors
data(Ishigami.factors)

# Build the processes and workflow:

# 1) the design process
exp.designer <- mtkBasicMonteCarloDesigner(listParameters=list(size=20))

# 2) the simulation process
exp.evaluator <- mtkIshigamiEvaluator()

# 3) the analysis process
exp.analyser <- mtkRegressionAnalyser(listParameters=list(nboot=20))

# 4) the workflow
exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors, processesVector = c(design=exp.designer, evaluate=exp.evaluator, analyze=exp.analyser))

# Run the workflow and report the results.
run(exp1)
```
The constructor of the class `mtkRegressionAnalyserResult`

**Description**

The constructor

**Usage**

```r
mtkRegressionAnalyserResult(main, information=NULL)
```

**Arguments**

- `main` : a data.frame holding the results of the sensitivity analysis produced by the analyser.
- `information` : a named list containing the information about the managed data.

**Value**

an object of the `mtkRegressionAnalyserResult` class

**See Also**

`help(morris, sensitivity)` and `help(Regression)`

**Examples**

```r
## See examples from help(mtkAnalyserResult)
```
mtkRegressionAnalyserResult-class

Description

A class to collect the results of the sensitivity analysis produced by the analyser implementing the method Regression.

Class Hierarchy

Parent classes: mtkAnalyserResult

Direct Known Subclasses:

Constructor

mtkRegressionAnalyserResult signature(main, information=NULL)

Slots

main: (data.frame) a data.frame holding the experimental design.
information: (NULL) a named list containing optional information about the managed data.

Methods

summary signature(object = "mtkRegressionAnalyserResult"): Provides a summary of the experimental design produced by the analyser.
print signature(x = "mtkRegressionAnalyserResult"): Prints a report of the experimental design produced by the analyser.
plot signature(x = "mtkRegressionAnalyserResult"): Plots the experimental design produced by the analyser.

See Also

help(morris, sensitivity) and help(Regression)

Examples

## See examples from help(mtkAnalyserResult)
The constructor of the class \texttt{mtkResult}

\section*{Description}

The constructor

\section*{Usage}

\begin{verbatim}
mtkResult(information=list())
\end{verbatim}

\section*{Arguments}

\begin{itemize}
\item \texttt{information} a named list containing the information about the managed data.
\end{itemize}

\section*{Value}

an object of the \texttt{mtkResult} class

\section*{Details}

The \texttt{mtkResult} class is a virtual class to manage the generic properties of results produced by the processes involved in the "mtk" package.

For details uses, see examples from \texttt{help(mtkAnalyserResult)}, \texttt{help(mtkDesignerResult)}, \texttt{help(mtkEvaluatorResult)}.

\section*{Author(s)}

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

\section*{References}


\section*{See Also}

\begin{verbatim}
help(mtkAnalyserResult), help(mtkDesignerResult), help(mtkEvaluatorResult)
\end{verbatim}

\section*{Examples}

\begin{verbatim}
## See examples from help(mtkAnalyserResult), help(mtkDesignerResult), help(mtkEvaluatorResult)
\end{verbatim}
The mtkResult class

**Description**
A general and simple class to collect the results produced by diverse processes involved in the "mtk" package.

**Class Hierarchy**
- **Parent classes**: 
- **Direct Known Subclasses**: mtkDesignerResult, mtkEvaluatorResult, etc.

**Constructor**
- `mtkResult` signature(information=list())

**Slots**
- `information`: (list) a named list containing information about the managed data.

**Methods**
- `summary` signature(object = "mtkResult"): Provides a summary report about the managed data.
- `serializeOn` signature(this = "mtkResult"): Returns all managed data as a named list.

**Details**
The `mtkResult` class is a virtual class to manage the generic properties of results produced by the processes involved in the "mtk" package.

For details uses, see examples from help(mtkAnalyserResult), help(mtkDesignerResult), help(mtkEvaluatorResult).

**Author(s)**
Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

**References**

**See Also**
help(mtkAnalyserResult), help(mtkDesignerResult), help(mtkEvaluatorResult)
Examples

```r
## See examples from help(mtkAnalyserResult), help(mtkDesignerResult), help(mtkEvaluatorResult)
```

---

**mtkSobolAnalyser**  
*The constructor of the class mtkSobolAnalyser*

### Description

The constructor

### Usage

```r
mtkSobolAnalyser(mtkParameters = NULL, listParameters = NULL)
```

### Arguments

- `mtkParameters`: a vector of `[mtkParameter]` representing the parameters necessary to run the process.
- `listParameters`: a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

### Value

An object of the `mtkSobolAnalyser` class

### References


### See Also

`help(sobol2002, sensitivity)` and `help(Sobol)`

### Examples

```r
## Sensitivity analysis of the "Ishigami" model with the "Sobol" method
```
mtkSobolAnalyser-class

Description

The mtkSobolAnalyser class is a sub-class of the class mtkAnalyser. It implements the sensitivity analysis method Sobol and provides all the slots and methods defined in the class mtkAnalyser.

Class Hierarchy

Parent classes: mtkAnalyser

Direct Known Subclasses:

Constructor

mtkSobolAnalyser signature(mtkParameters = NULL, listParameters = NULL)

Slots

name: (character) always takes the string "analyze".
protocol: (character) always takes the string "R".
site: (character) always takes the string "mtk".
service: (character) always takes the string "Sobol".
parameters: (vector) a vector of [mtkParameter] containing the parameters to pass while calling the service.
ready: (logical) a logical to tell if the process is ready to run.
state: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.
result: (ANY) a data holder to hold the results produced by the process

Methods

setName signature(this = "mtkSobolAnalyser", name = "character"): Not used, method inherited from the parent class.
setParameters signature(this = "mtkSobolAnalyser", f = "vector"): Assigns new parameters to the process.
getParameters signature(this = "mtkSobolAnalyser"): Returns the parameters as a named list.
is.ready signature( = "mtkSobolAnalyser"): Tests if the process is ready to run.
setReady signature(this = "mtkSobolAnalyser", switch = "logical"): Makes the process ready to run.
is.ready signature( = "mtkSobolAnalyser"): Tests if the results produced by the process are available.
**setReady** signature(this = "mtkSobolAnalyser", switch = "logical"): Marks the process as already executed.

**getResult** signature(this = "mtkSobolAnalyser"): Returns the results produced by the process as a `mtkSobolAnalyserResult`.

**getData** signature(this = "mtkSobolAnalyser"): Returns the results produced by the process as a data.frame.

**serializeOn** signature(this = "mtkSobolAnalyser"): Returns all data managed by the process as a named list.

**run** signature(this = "mtkSobolAnalyser", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.

**summary** signature(object = "mtkSobolAnalyser"): Provides a summary of the results produced by the process.

**print** signature(x = "mtkSobolAnalyser"): Prints a report of the results produced by the process.

**plot** signature(x = "mtkSobolAnalyser"): Plots the results produced by the process.

**report** signature(this = "mtkSobolAnalyser"): Reports the results produced by the process.

---

**References**


**See Also**

help(sobol, sensitivity) and help(Sobol)

**Examples**

```r
## Sensitivity analysis of the "Ishigami" model with the "Sobol" method

mtkSobolAnalyserResult

*The constructor of the class mtkSobolAnalyserResult*

---

**Description**

The constructor

**Usage**

```r
mtkSobolAnalyserResult(main, information=NULL)
```
mtkSobolAnalyserResult-class

Arguments

main a data.frame holding the results of the sensitivity analysis produced by the analyser.
information a named list containing the information about the managed data.

Value

an object of the mtkSobolAnalyserResult class

See Also

help(mtkAnalyserResult) and help(Sobol)

Examples

## See examples from help(mtkAnalyserResult).

---

mtkSobolAnalyserResult-class

_The mtkSobolAnalyserResult class_

---

Description

A class to collect the results of the sensitivity analysis produced by the analyser implementing the method Sobol.

Class Hierarchy

- **Parent classes**: mtkAnalyserResult
- **Direct Known Subclasses**:

Constructor

mtkSobolAnalyserResult signature(main,information=NULL)

Slots

main: *(data.frame)* a data.frame holding the experimental design.
information: *(NULL)* a named list containing optional information about the managed data.
Methods

`summary` signature(object = "mtkSobolAnalyserResult"): Provides a summary of the experimental design produced by the analyser.

`print` signature(x = "mtkSobolAnalyserResult"): Prints a report of the experimental design produced by the analyser.

`plot` signature(x = "mtkSobolAnalyserResult"): Plots the experimental design produced by the analyser.

See Also

help(mtkAnalyserResult) and help(Sobol)

Examples

```r
## See examples from help(mtkAnalyserResult).
```

---

**mtkSobolDesigner**  
*The constructor of the class mtkSobolDesigner*

Description

The constructor

Usage

```r
mtkSobolDesigner(mtkParameters = NULL, listParameters = NULL)
```

Arguments

- `mtkParameters`: a vector of `[mtkParameter]` representing the parameters necessary to run the process.
- `listParameters`: a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

Value

an object of the `mtkSobolDesigner` class

References

See Also

help(sobol2002, sensitivity) and help(Sobol)

Examples

## Sensitivity analysis of the "Ishigami" model with the "Sobol" method

```r

mtkSobolDesigner-class

The mtkSobolDesigner class

Description

This class is a sub-class of the class \texttt{mtkDesigner}. It implements the sampling method 'Sobol' and provides all the slots and methods defined in the class \texttt{mtkDesigner}.

Class Hierarchy

Parent classes: \texttt{mtkDesigner}

Direct Known Subclasses:

Constructor

\texttt{mtkSobolDesigner} signature(\texttt{mtkParameters = NULL}, \texttt{listParameters = NULL})

Slots

name: (\texttt{character}) always takes the string "design".
protocol: (\texttt{character}) always takes the string "R".
site: (\texttt{character}) always takes the string "mtk".
service: (\texttt{character}) always takes the string "Sobol".
parameters: (\texttt{vector}) a vector of \texttt{mtkParameter} containing the parameters to pass while calling the service.
ready: (\texttt{logical}) a logical to tell if the process is ready to run.
state: (\texttt{logical}) a logical to tell if the results produced by the process are available and ready to be consumed.
result: (\texttt{ANY}) a data holder to hold the results produced by the process
```
Methods

setname signature(this = "mtkSobolDesigner", name = "character"): Not used, method inherited from the parent class.

setparameters signature(this = "mtkSobolDesigner", f = "vector"): Assigns new parameters to the process.

getparameters signature(this = "mtkSobolDesigner"): Returns the parameters as a named list.

is.ready signature( = "mtkSobolDesigner"): Tests if the process is ready to run.

setReady signature(this = "mtkSobolDesigner", switch = "logical"): Makes the process ready to run.

is.ready signature( = "mtkSobolDesigner"): Tests if the results produced by the process are available.

setReady signature(this = "mtkSobolDesigner", switch = "logical"): Marks the process as already executed.

getResult signature(this = "mtkSobolDesigner"): Returns the results produced by the process as a [mtkSobolDesignerResult].

getData signature(this = "mtkSobolDesigner"): Returns the results produced by the process as a data.frame.

serializeOn signature(this = "mtkSobolDesigner"): Returns all data managed by the process as a named list.

run signature(this = "mtkSobolDesigner", context= "mtkExpWorkflow"): Generates the experimental design by sampling the factors.

summary signature(object = "mtkSobolDesigner"): Provides a summary of the results produced by the process.

print signature(x = "mtkSobolDesigner"): Prints a report of the results produced by the process.

plot signature(x = "mtkSobolDesigner"): Plots the results produced by the process.

report signature(this = "mtkSobolDesigner"): Reports the results produced by the process.

References


See Also

help(sobol, sensitivity) and help(Sobol)

Examples

## Sensitivity analysis of the "Ishigami" model with the "Sobol" method
mtnSobolDesignerResult

Description

The constructor of the class mtnSobolDesignerResult

Usage

mtnSobolDesignerResult(main, information=NULL)

Arguments

main a data.frame holding the experimental design produced by the designer.
information a named list containing the information about the managed data.

Value

an object of the mtnSobolDesignerResult class

See Also

help(mtnDesignerResult) and help(Sobol)

Examples

## See examples from help(mtnDesignerResult).

mtnSobolDesignerResult-class

Description

A class to collect the experimental design produced by the Designer implementing the method Sobol.

Class Hierarchy

Parent classes: mtnDesignerResult

Direct Known Subclasses:
Constructor

\texttt{mtkSobolDesignerResult} signature(\texttt{main,information=NULL})

Slots

\texttt{main}: \texttt{(data.frame)} a data.frame holding the experimental design.

\texttt{information}: \texttt{(NULL)} a named list containing optional information about the managed data.

Methods

\texttt{summary} signature(object = "mtkSobolDesignerResult"): Provides a summary of the experimental design produced by the designer.

\texttt{print} signature(x = "mtkSobolDesignerResult"): Prints a report of the experimental design produced by the designer.

\texttt{plot} signature(x = "mtkSobolDesignerResult"): Plots the experimental design produced by the designer.

See Also

help(mtkDesignerResult) and help(Sobol)

Examples

## See examples from help(mtkDesignerResult).

---

\textit{mtkSystemEvaluator} \hspace{1cm} The constructor of the class \texttt{mtkSystemEvaluator}

Description

The constructor

Usage

\begin{verbatim}
mtkSystemEvaluator(
  service = "",
  mtkParameters = NULL,
  listParameters = NULL
)
\end{verbatim}
Arguments

service a string specifying the way to invoke the application implementing the model.

mtkParameters a vector of \texttt{mtkParameter} representing the parameters necessary to run the process.

listParameters a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

Value

an object of the \texttt{mtkSystemEvaluator} class

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

\begin{verbatim}
# see examples
\end{verbatim}

\begin{verbatim}
mtkSystemEvaluator-class
\end{verbatim}

\begin{verbatim}
The mtkSystemEvaluator class
\end{verbatim}

Description

The \texttt{mtkSystemEvaluator} class is a sub-class of the class \texttt{mtkEvaluator} used to manage the simulation of the model implemented as a system application.

Class Hierarchy

\textbf{Parent classes} : \texttt{mtkEvaluator}

\textbf{Direct Known Subclasses} :

Constructor

\texttt{mtkSystemEvaluator} signature( service="", mtkParameters=NULL, listParameters = NULL)
**Slots**

- **name**: (character) always takes the string "evaluate".
- **protocol**: (character) always takes the string "system".
- **site**: (character) not used here.
- **service**: (character) a string to invoke the system command implementing the model.
- **parameters**: (vector) a vector of [mtkParameter] containing the parameters to pass while invoking the system command.
- **ready**: (logical) a logical to tell if the process is ready to run.
- **state**: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.
- **result**: (ANY) a data holder to hold the results produced by the process.

**Methods**

- **setName** signature(this = "mtkSystemEvaluator", name = "character"): Not used, method inherited from the parent class.
- **setParameters** signature(this = "mtkSystemEvaluator", f = "vector"): Assigns new parameters to the process.
- **getParameters** signature(this = "mtkSystemEvaluator"): Returns the parameters as a named list.
- **is.ready** signature( = "mtkSystemEvaluator"): Tests if the process is ready to run.
- **setReady** signature(this = "mtkSystemEvaluator", switch = "logical"): Makes the process ready to run.
- **is.ready** signature( = "mtkSystemEvaluator"): Tests if the results produced by the process are available.
- **setReady** signature(this = "mtkSystemEvaluator", switch = "logical"): Marks the process as already executed.
- **getResult** signature(this = "mtkSystemEvaluator"): Returns the results produced by the process as a [mtkEvaluatorResult].
- **getData** signature(this = "mtkSystemEvaluator"): Returns the results produced by the process as a data.frame.
- **serializeOn** signature(this = "mtkSystemEvaluator"): Returns all data managed by the process as a named list.
- **run** signature(this = "mtkSystemEvaluator", context= "mtkExpWorkflow"): runs the simulation.
- **summary** signature(object = "mtkSystemEvaluator"): Provides a summary of the results produced by the process.
- **print** signature(x = "mtkSystemEvaluator"): Prints a report of the results produced by the process.
- **plot** signature(x = "mtkSystemEvaluator"): Plots the results produced by the process.
- **report** signature(this = "mtkSystemEvaluator"): Reports the results produced by the process.

**Author(s)**

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr
References


Examples

# see examples

mtkSystemEvaluatorResult

The constructor of the class mtkSystemEvaluatorResult

Description

The constructor

Usage

mtkSystemEvaluatorResult(main, information=NULL)

Arguments

main a data.frame holding the results produced by the evaluator.
information a named list containing the information about the managed data.

Value

an object of the mtkSystemEvaluatorResult class

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

Examples

# See examples
The `mtkSystemEvaluatorResult` class

Description

A class to collect the results produced by the evaluator implemented as a system application.

Class Hierarchy

- **Parent classes**: `mtkEvaluatorResult`
- **Direct Known Subclasses**: 

Constructor

`mtkSystemEvaluatorResult` signature(main, information=NULL)

Slots

- **main**: (data.frame) a data.frame holding the results produced by the model simulation.
- **information**: (NULL) a named list containing optional information about the managed data.

Methods

- **summary** signature(object = "mtkSystemEvaluatorResult"): Provides a summary of the results produced by the evaluator.
- **print** signature(x = "mtkSystemEvaluatorResult"): Prints a report of the results produced by the evaluator.
- **plot** signature(x = "mtkSystemEvaluatorResult"): Plots the results produced by the evaluator.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

Examples

# See examples
mtkValue-class

The constructor of the class mtkValue

Description

The constructor

Usage

mtkValue(name='unknown', type='', val=NULL)

Arguments

name the name of the variable.

the type of the variable, i.e. double, integer, character, logical, null, etc.

val the value of the variable. It may be a single or a vector of values.

Value

an object of the mtkValue class

Author(s)

Juhui WANG, MIA-jouy, INRA

Examples

# Create an object of 'mtkValue'

triple <- mtkValue('a', 'double', c(2.5, 3.0))

mtkValue-class

The mtkValue class

Description

The mtkValue class is a virtual class used to manage a triple (name, type, value).

Class Hierarchy

Parent classes:

Direct Known Subclasses: mtkParameter, codemtkFeature

Constructor

mtkValue signature(name='unknown', type='', val=NULL)
Slots

name: (character) the name of the variable.

val: (ANY) the value of the variable in the right type. It may be a single value or a vector of values.

Methods

getName signature( this = "mtkValue"): Returns the value of the slot "name".

getValue signature( this = "mtkValue"): Returns the value of the slot "val".

getType signature(this = "mtkValue"): Returns the value of the slot "type".

setName signature(this = "mtkValue", name = "character"): Gives a new value to the slot "name".

setValue signature(this = "mtkValue", type = "ANY"): Gives a new value to the slot "val".

setType signature(this = "mtkValue", type = "character"): Gives a new value to the slot "type".

show signature( object = "mtkValue"): Prints a report of the data managed by the underlying object.

print signature(x = "mtkValue"): Prints the information managed by the underlying object.

Author(s)

Juhui WANG, MIA-jouy, INRA

Examples

# Create a new object of 'mtkValue'
d <- mtkValue("a", "double", c(0,1))
getType(d) # gives "double"
getName(d) # gives "a"
getValue(d) # gives (0, 1)

setType(d, 'character')
getName(d) # gives "a"
getValue(d) # gives (0, 1)

setValue(d, "3.14")
getValue(d) # gives "3.14"
Arguments

*mtkParameters* a vector of `[mtkParameter]` representing the parameters necessary to run the process.

*listParameters* a named list containing the parameters to pass while calling the process. This gives another way to specify the parameters.

Value

an object of the `mtkWWDMEvaluator` class

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


See Also

`help(WWDM)`

Examples

```r
# Carry out a sensitivity analysis with the WWDM model

## Input the factors
data(WWDM.Factors)

## Specify the experiments designer
designer <- mtkMorrisDesigner(
  listParameters = list(type="oat", levels=5, grid.jump=3, r=10)
)

## Specify the model simulator
model <- mtkWWDMEvaluator(
  listParameters = list(year=3, tout=FALSE)
)

## Specify the sensitivity analyser
analyser <- mtkMorrisAnalyser()
```
## mtkWWDMEvaluator-class

**The mtkWWDMEvaluator class**

**Description**

The `mtkWWDMEvaluator` class is a sub-class of the class `mtkEvaluator` used to manage the simulation of the model WWDM.

**Class Hierarchy**

**Parent classes**: `mtkEvaluator`

**Direct Known Subclasses**: 

**Constructor**

```r
mtkWWDMEvaluator signature(mtkParameters = NULL, listParameters = NULL)
```

**Slots**

- `name`: (character) always takes the string "evaluate".
- `protocol`: (character) a string to name the protocol used to run the process: http, system, R, etc. Here, it always takes the character "R".
- `site`: (character) a string to indicate where the service is located. Here, it always takes the string "mtk".
- `service`: (character) a string to name the service to invoke. Here, it always takes the string "WWDM".
- `parameters`: (vector) a vector of `mtkParameter` containing the parameters to pass while calling the service. The WWDM model does not need parameters.
- `ready`: (logical) a logical to tell if the process is ready to run.
- `state`: (logical) a logical to tell if the results produced by the process are available and ready to be consumed.
- `result`: (ANY) a data holder to hold the results produced by the process.

```r
## Specify the workflow
exp <- new("mtkExpWorkflow", expFactors=WWDM.factors,
    processesVector=c(
        design=designer,
        evaluate=model,
        analyze=analyser
    )
)
## Run and report the results
run(exp)
summary(exp)
```
Methods

set

setName signature(this = "mtkWWDMEvaluator", name = "character"): Not used, method inherited from the parent class.

setParameters signature(this = "mtkWWDMEvaluator", f = "vector"): Assigns new parameters to the process.

getParameters signature(this = "mtkWWDMEvaluator"): Returns the parameters as a named list.

is.ready signature( = "mtkWWDMEvaluator"): Tests if the process is ready to run.

setReady signature(this = "mtkWWDMEvaluator", switch = "logical"): Makes the process ready to run.

is.ready signature( = "mtkWWDMEvaluator"): Tests if the results produced by the process are available.

setReady signature(this = "mtkWWDMEvaluator", switch = "logical"): Marks the process as already executed.

getResult signature(this = "mtkWWDMEvaluator"): Returns the results produced by the process as a [mtkWWDMEvaluatorResult].

getData signature(this = "mtkWWDMEvaluator"): Returns the results produced by the process as a data.frame.

serializeOn signature(this = "mtkWWDMEvaluator"): Returns all data managed by the process as a named list.

run signature(this = "mtkWWDMEvaluator", context= "mtkExpWorkflow"): runs the simulation.

summary signature(object = "mtkWWDMEvaluator"): Provides a summary of the results produced by the process.

print signature(x = "mtkWWDMEvaluator"): Prints a report of the results produced by the process.

plot signature(x = "mtkWWDMEvaluator"): Plots the results produced by the process.

report signature(this = "mtkWWDMEvaluator"): Reports the results produced by the process.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


See Also

help(WWDM)
Examples

```r
# Carry out a sensitivity analysis with the WWDM model

## Input the factors
data(WWDM.factors)

## Specify the experiments designer
designer <- mtkMorrisDesigner(
  listParameters = list(type="oat", levels=5, grid.jump=3, r=10)
)

## Specify the model simulator
model <- mtkWWDEvaluator(
  listParameters = list(year=3)
)

## Specify the sensitivity analyser
analyser <- mtkMorrisAnalyser()

## Specify the workflow
exp <- new("mtkExpWorkflow", expFactors=WWDM.factors,
  processesVector=c(
    design=designer,
    evaluate=model,
    analyze=analyser
  )
)

## Run and report the results
run(exp)
summary(exp)
```

### mtkWWDEvaluatorResult

The constructor of the class `mtkWWDEvaluatorResult`

#### Description

The constructor

#### Usage

`mtkWWDEvaluatorResult(main, information=NULL)`

#### Arguments

- **main**: a data.frame holding the results produced by the evaluator.
- **information**: a named list containing the information about the managed data.
Value

an object of the `mtkWWDMEvaluatorResult` class

See Also

`help(mtkEvaluatorResult)` and `help(WWDM)`

Examples

```r
## See examples from `help(mtkEvaluatorResult)`. 
```
Examples

## See examples from help(mtkEvaluatorResult).

---

**PLMM**  
*The PLMM method for sensitivity analysis*

**Description**

A mtk compliant implementation of the PLMM method for sensitivity analysis using polynomial linear metamodelling.

**Usage**

- `mtkPLMMAnalyser(listParameters = NULL)`
- `mtkNativeAnalyser(analyze="PLMM", information=NULL)`

**Parameters**

degree.pol: the maximum degree of polynomials (the sum of the degrees of cross products of polynomials is lower or equal to degree.pol). See details.

rawX: orthogonal polynomials (default value FALSE) or raw polynomials (TRUE). See poly, polym.

numY: the column number of the dependent variable (default is the first column of the dataframe of outputs).

listX: the column numbers of the dependent variables (default is all the dependent variables).

**Parameters for auxiliary functions**

all: all the specific summaries and plots are displayed if TRUE (default is FALSE). Else, see the which option.

which: when all=FALSE, the name of the specific summary or plot. Options are "best" (default), "full", "best.adjustedR2", "full.adjustedR2". See details.

lang: language of the summary and plot ("en" (default) for english, "fr" for french).

digits: number of digits in the summary (default = options(digits).

colors: colors used in plot (default = c("red", "orange","blue")).

legend.loc: location of the legend in plot (default no legend( NULL), options are "topleft", "topright", ... See help(legend)).
Details

1. The PLMM metamodelling approach consists in estimating 3 models and comparing the percentage of variance (coefficient of determination) explained by these 3 models. The 3 models are \( \text{poly}(A,B,C) \), \( \text{poly}(A) \), \( \text{poly}(B,C) \) where \( \text{poly} \) computes orthogonal polynomials. \( \text{poly}(A,B,C) \) gives the total variance explained by the full metamodel, \( \text{poly}(A) \) gives the variance that can be explained by factor A only (in the sense of polynomials of A) and \( \text{poly}(B,C) \) gives the variance not explained by factor A. Total sensitivity index of factor A is computed as \( \max \{ R^2(\text{poly}(A)), 1 - R^2(\text{poly}(A,B,C)) - R^2(\text{poly}(B,C)) \} \) where \( R^2(M) \) is the coefficient of determination of model M, and first order sensitivity index as \( \min \{ R^2(\text{poly}(A)), 1 - R^2(\text{poly}(A,B,C)) - R^2(\text{poly}(B,C)) \} \). The PLMM function computes a best model in the sense of stepwise model selection starting with the constant model with direction fixed to both (see \text{stepAIC} for more details). Total sensitivity and first order indices are computed in the same. Additional results are given when using adjusted \( R^2 \) for both best and full models. Names of the results (needed in which option) are: best, full, best.adjustedR2, full.adjustedR2.

2. Computational aspects: PLMM does not use the \( \text{poly} \) function (as \( \text{poly} \) needs time to orthogonalize when the number of factors and the degree of the polynomials are high). The cross products are computed as cross products of one dimensional orthogonal polynomials \( \text{poly}(A) \ast \text{poly}(B) \ast \text{poly}(C) \). So we have to take care with the selected components of the best model (obtained with a stepwise model selection). Care should be taken for interpreting them because the dependent variables are orthogonalized. This not the case when the \text{rawX} option is set to TRUE. To prevent from computational side effects, the input factors are first scaled.

3. The \texttt{mtk} implementation of the PLMM method includes the following classes:
   - \texttt{mtkPLMMAnalyser}: for PLMM analysis processes.
   - \texttt{mtkPLMMAnalyserResult}: to store and manage the analysis results.

4. The \texttt{mtk} implementation of the PLMM method includes the following generic functions:
   - \texttt{summary}: to display summary of analysis results. See parameters for auxiliary functions.
   - \texttt{plot}: to plot analysis results. See parameters for auxiliary functions.

5. Many ways to create a PLMM analyser are available in \texttt{mtk}, but we recommend the following class constructors: \texttt{mtkPLMMAnalyser} or \texttt{mtkNativeAnalyser}.

References


See Also

\texttt{help(poly, stepAIC)}

Examples

```r
## Sensitivity analysis of the "Ishigami" model with the "PLMM" method
```
# Generates the factors
data(Ishigami.factors)

# Builds the processes and workflow:

# 1) the experimental design process with the method "BasicMonteCarlo".
expl.designer <- mtkNativeDesigner("BasicMonteCarlo", information=list(size=100))

# 2) the simulation process
expl.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process
expl.analyser <- mtkNativeAnalyser("PLMM", information = list(degree.pol=3, numY=1))

# 4) the workflow
expl <- mtkExpWorkflow(expFactors=Ishigami.factors, 
processesVector = c(design=expl.designer, 
evaluate=expl.evaluator, 
analyze=expl.analyser))

# Runs the workflow and reports the results.
run(expl)
summary(getProcess(expl, name="analyze"), lang="fr")
summary(getProcess(expl, name="analyze"), lang="fr", 
which="full", all=FALSE, digit=4)
eextractData(expl, name="analyze")$best$call
plot(getProcess(expl, name="analysis"), lang="fr", legend.loc="topleft")
plot(getProcess(expl, name="analysis"), which="full", 
all=FALSE, legend.loc="topright")

## Example II: comparing metamodes of the WWDM model

# Generates the factors
data(WWDM.factors)

# 1) to create a sampler with the Monte-Carlo method
sampler <- mtkNativeDesigner("BasicMonteCarlo", information = list(size=100) )

# 2) to create a simulator with the WWDM model
model <- mtkNativeEvaluator("WWDM", information = list(year=3))

# 3) to create a partial workflow (design and evaluation)
experience1 <- mtkExpWorkflow(expFactors=WWDM.factors, 
processesVector=c(design=sampler, evaluate=model) )
run(experience1)

# 4) to create an "analysor" with the Regression method
The `plot` method

**Description**

Plots graphically the results produced by the process.

**Usage**

```r
plot(x, y, ...)```

**Arguments**

- `x` : the underlying object of class `mtkProcess`
- `y` : `see par` for details about the graphical parameter arguments
- `...` : `see par` for details about the graphical parameter arguments
Value

invisible()

Details

1. The behavior of the plot depends on the sub-class where the method is implemented.
2. See the documentation of the particular sub-class for details of what is produced. Use `methods("plot")` to get all the methods for the plot generic.
3. See `par` for details about the graphical parameter arguments.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

See Also

`help(plot)`

Examples

# Create a designer and an analyser avec the method "Morris"
# to analyze the model "Ishigami":

# Specify the factors to analyze:
x1 <- make.mtkFactor(name="x1", distribName="unif",
distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
distribPara=list(min=-pi, max=pi))
factors <- mtkExpFactors(list(x1,x2,x3))

# Build the processes:
# 1) the experimental design process with the method "Morris".
exp1.designer <- mtkNativeDesigner(design="Morris",
   information=list(r=20,type="oat",levels=4,grid.jump=2))

# 2) the model simulation process with the model "Ishigami".
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# # 3) the analysis process with the default method.
# Here, it is the Morris method.
exp1.analyser <- mtkDefaultAnalyser()

# Build the workflow with the processes defined previously.
exp1 <- mtkExpWorkflow(expFactors=factors,
   processesVector = c(design=exp1.designer,
   evaluate=exp1.evaluator, analyze=exp1.analyser))
# Run the workflow and plot the results.
run(exp1)
plot(exp1)
# Extract a process and report its results

p <- getProcess(exp1, "analyze")
plot(p)

---

## Description

Prints a report of the results produced by the process.

## Usage

`print(x, ...)`

## Arguments

- `x` the underlying object of class `mtkProcess`.
- `...` see the documentation of the function: `base::print()`.

## Value

`invisible()`

## Details

1. The behavior of the `print` depends on the sub-class where the method is implemented.
2. See the documentation of the particular sub-class for details of what is produced.
3. Use `methods("print")` to get all the methods for the `print` generic.

## Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

## See Also

`help(print)`
Examples

# Create a designer and an analyser avec the method "Morris"
# to analyze the model "Ishigami":

# Specify the factors to analyze:
x1 <- make.mtkFactor(name="x1", distribName="unif",
                     distribPara=list(min=pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
                     distribPara=list(min=pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
                     distribPara=list(min=pi, max=pi))
factors <- mtkExpFactors(list(x1,x2,x3))

# Build the processes:
# 1) the experimental design process with the method "Morris".
exp1.designer <- mtkNativeDesigner(design="Morris",
                                   information=list(r=20,type="oat",levels=4,grid.jump=2))

# 2) the model simulation process with the model "Ishigami".
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process with the default method.
# Here, it is the Morris method.
ex1.analyser <- mtkDefaultAnalyser()

# Build the workflow with the processes defined previously.
ex1 <- mtkExpWorkflow(expFactors=factors,
                       processesVector = c(design=exp1.designer,
                                          evaluate=exp1.evaluator, analyze=exp1.analyser))
# Run the workflow and plot the results.
run(ex1)
print(ex1)

# Extract a process and report its results
p <- getProcess(ex1, "analyze")
print(p)

Quantiles

The Quantiles function

Description

Calculates the quantiles of a univariate distribution.

Usage

Quantiles(pvalues, distribName, distribParameters, shrink=0.95)
Arguments

pvalues a vector of probability values.
distribName a string giving the name of a probability distribution.
distribParameters a list of parameters of the distribution.
shrink a scalar eqn<=1 to determine how to shrink the pvalues(used when the quantiles are infinite for pvalues equal to 0 or 1).

Value

the q-values

Author(s)

Hervé Monod, MIA-Jouy, Inra, Domaine de Vilvert, 78352 Jouy en Josas, France

Examples

Quantiles(seq(0,1,length=11),"unif",list(min=8,max=16))
Quantiles(seq(0,1,length=11),"unif",list(min=8,max=16),shrink=0.5)
Quantiles(seq(0,1,length=11),"norm",list(mean=0, sd=1),shrink=0.5)

Description

A mtk compliant implementation of the method for drawing Random Latin Hypercube Design.

Usage

- mtkRandLHSDesigner(listParameters = NULL)
- mtkNativeDesigner(design="RandLHS", information=NULL)

Parameters used to manage the method

size: The number of partitions (simulations or design points).
preserveDraw: logical (default FALSE). Ensures that two subsequent draws with the same n, but one with k and one with m variables (k<m), will have the same first k columns if the seed is the same.

Details

1. The mtk implementation uses the randomLHS function of the package lhs. For further details on the arguments and the behavior, see help(randomLHS, lhs).
2. The implementation of the RandLHS method includes the class mtkRandLHSDesigner to manage the sampling task and the class mtkRandLHSDesignerResult to manage the results produced by the sampling process.
References


See Also

help(randomLHS, lhs)

Examples

# uses the RandLHS method
## Random Latin Hypercube draws for the "Ishigami" model

# Example I: by using the class constructors: mtkRandLHSDesigner()

# Generate the factors
data(Ishigami.factors)

# Build the processes and workflow:

# 1) the design process
expl.designer <- mtkRandLHSDesigner( listParameters = list(size=10) )

# 2) the workflow

expl <- mtkExpWorkflow(expFactors=Ishigami.factors,
                       processesVector = c(design=expl.designer) )

# Run the workflow and reports the results.
run(expl)
print(expl)
plot(expl)

---

**reevaluate-methods**

**The reevaluate method**

**Description**

Re-evaluates the processes of the workflow to know if they should be re-run. This must be done after changing a process in the workflow. The argument "name" gives the process from which the workflow should be reevaluated. i.e. if name="design", we tell the workflow that all the processes after the process "design" should be reevaluated. If name="evaluate", we tell the workflow that only the processes after the process "evaluate" should be re-evaluated, etc.

**Usage**

reevaluate(this, name)
Arguments

- `this`: the underlying object of class `mtkExpWorkflow`.
- `name`: a string from "design", "evaluate", or "analyze" to specify the process from which we re-evaluate the workflow.

Value

`invisible()`

Details

This function is only useful for the kernel programming.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

```
# see examples.
```

Description

`mtk` compliant implementation of the `src` method for computing the sensitivity index based on standardized (rank) regression coefficients.

Usage

```
• mtkRegressionAnalyser(listParameters = NULL)
• mtkNativeAnalyser(analyze="Regression", information=NULL)
```

Parameters used to manage the method

- `rank`: logical. If TRUE, the analysis is done on the ranks (default is FALSE). See the help on function `src` in the package `sensitivity`.
- `nboot`: the number of bootstrap replicates (default 100). See the help on function `src` in the package `sensitivity`.
- `conf`: the confidence level for bootstrap confidence intervals (default 0.95). See the help on function `src` in the package `sensitivity`. 
Details

1. The mtk implementation uses the src function of the package sensitivity. For further details on the arguments and the behavior, see help(src, sensitivity).

2. The implementation of the "Regression" method includes the class mtkRegressionAnalyzer to manage the analysis task and the class mtkRegressionAnalyzerResult to manage the results produced by the analysis process.

References


See Also

help(src, sensitivity)

Examples

# Uses the method "Regression" to analyze the model "Ishigami":
# Generate the factors
data(Ishigami.factors)

# Builds experiment design with the Monte-Carlo method
designer <- mtkBasicMonteCarloDesigner( listParameters=list(size=20) )

# Builds a simulator for the model "Ishigami" with the defined factors
model <- mtkNativeEvaluator("Ishigami")

# Builds an analyser with the method "Regression" implemented in the package "mtk"
analyser <- mtkNativeAnalyzer("Regression", information=list(nboot=20))

# Builds a workflow to manage the processes scheduling.
ishiReg <- mtkExpWorkflow( expFactors=Ishigami.factors, processesVector=c(design=designer, evaluate=model, analyze=analyser) )

# Runs the workflow et reports the results
run(ishiReg)
summary(ishiReg)
plot(ishiReg)

Description

Returns a detail report of the results produced by the process.
Usage

```r
report(this)
```

Arguments

- `this`: the underlying object of class `mtkProcess`

Value

The form of the value returned by `report` depends on the sub-class where the method is implemented.

See the documentation of the particular sub-class for details of what is produced.

By default, it prints the report on the display device and return `invisible()`.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

```r
# Create a designer and an analyser with the method "Morris"
# to analyze the model "Ishigami":

# Specify the factors to analyze:
x1 <- make.mtkFactor(name="x1", distribName="unif",
                     distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
                     distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
                     distribPara=list(min=-pi, max=pi))
factors <- mtkExpFactors(list(x1,x2,x3))

# Build the processes:
# 1) the experimental design process with the method "Morris".
expl.designer <- mtkNativeDesigner(design="Morris",
                                   information=list(r=20,type="oat",levels=4,grid.jump=2))

# 2) the model simulation process with the model "Ishigami".
expl.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process with the default method.
# Here, it is the Morris method.
```
run-methods

```r
exp1.analyser <- mtkDefaultAnalyser()

# Build the workflow with the processes defined previously.
exp1 <- mtkExpWorkflow(expFactors=factors,
                        processesVector = c(design=exp1.designer,
                                        evaluate=exp1.evaluator, analyze=exp1.analyser))
# Run the workflow and plot the results.
run(exp1)
report(exp1)

# Extract a process and report its results
p <- getProcess(exp1, "analyze")
report(p)
```

### Description

Runs a task defined in a process or workflow. Examples classes in which this function is implemented are the following: `[mtkParser]`, `[mtkExpWorkflow]`, `[mtkProcess]` and their sub-classes. Examples of "run" are:

- `run(this, context)"this" is an object of class [mtkNativeDesigner]`, and "context" is an object of class `[mtkExpWorkflow]`.
- `run(this, context)"this" is an object of class [mtkParser]`, and "context" is an object of class `[mtkExpWorkflow]`.

### Usage

```r
run(this, context)
```

### Arguments

- `this` an object corresponding to the task to launch. It may be an object of the following classes: `[mtkParser]`, `[mtkExpWorkflow]`, `[mtkProcess]` or their sub-classes.

- `context` missing or an object specifying the context which manages the task. It may be an object of the following classes: `[mtkExpWorkflow]` or its sub-classes.

### Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr
serializeOn-methods

The serializeOn method

Description

Returns all data and informations managed by an object as a named list.

Usage

serializeOn(this)
setDistributionParameters-methods

Arguments

this the underlying object

Value

a named list

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

Examples

# Function not used yet in the current release.

Description

Sets the parameters of the distribution associated with a factor’s domain.

Usage

setDistributionParameters(this, aDistParamList)

Arguments

this the underlying object of the class mtkDomain.

aDistParamList a list of objects of class mtkParameter or a named list from which we can build
a list of objects of class mtkParameter.

Value

invisible()

Author(s)

Juhui WANG, MIA-jouy, INRA
Examples

# 1) Build an object of the "mtkDomain" class
d <- mtkDomain(distributionName="unif", domainNominalValue=0)

## Define the parameters
p <- make.mtkParameterList(list(min=-pi, max=pi))

## Assign the parameters to the mtkDomain's object
setDistributionParameters(d, p)

# 2) Build an object of the "mtkDomain" class
# Assign the parameters to the mtkDomain's object
setDistributionParameters(d, list(min=-pi, max=pi))

# 3) Build an object of the "mtkDomain" class with a discrete distribution
# Assign the parameters to the mtkDomain's object
setDistributionParameters(d, list(type='categorical', levels=seq(1:3), weights=rep(0.33,3)))

setDomain-methods

Description

Associates a new domain with the factor.

Usage

setDomain(this, domain)

Arguments

this an object of the class mtkFactor.
domain an object of the class mtkDomain.

Value

invisible()

Author(s)

Juhui WANG, MIA-jouy, INRA
Examples

# Define a factor
x1 <- make.mtkFactor(name="x1")

# Define a domain
d <- mtkDomain(distributionName="unif",
               domainNominalValue=0, distributionParameters = list(max=3, min=0))

# Use the setDomain to change the domain of the factor
setDomain(x1,d)

setFactors-methods

The setFactors method

Description

Assigns a list of objects of the class mtkFactor to the underlying object.

Usage

setFactors(this, aFactList)

Arguments

this the underlying object of the class mtkExpFactors.
aFactList a list of objects of the class mtkFactor.

Value

invisible()

Author(s)

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

Examples

# Build an object of the "mtkExpFactors" class
ishi.factors <- mtkExpFactors()

# Define the factors
x1 <- make.mtkFactor(name="x1", distribName="unif",
                      distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif",
                      distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif",
                      distribPara=list(min=-pi, max=pi))
# Assign the factors to the mtkExpFactors' object

```r
setFactors(ishi.factors, list(x1,x2,x3))
```

---

**Description**

Sets the features to an object of the `mtkFactor` class.

**Usage**

```r
setFeatures(this, aFList)
```

**Arguments**

- `this` an object of the class `mtkFactor`
- `aFList` a list of `mtkFeature` objects.

**Value**

`invisible`

**Author(s)**

Hervé Richard, BioSP, Inra, Herve.Richard@avignon.inra.fr, Hervé Monod and Juhui WANG, MIA-jouy, INRA

**Examples**

```r
# Build an object of the "mtkFactor" class
x1 <- make.mtkFactor(name="x1", type="double", nominal=0, distribName="unif",
                      distribPara=list(min=-pi, max=pi))
# Define the list of features
f <- make.mtkFeatureList(list(f=4.5,c=+6,shape="parabolic"))

# Assign the features to the factor
setFeatures(x1,f)
```
setLevels-methods

The `setLevels` method

Description
Sets new levels to a discrete distribution.

Usage
```
setLevels(this, levels)
```

Arguments
```
this
  an object of the class `mtkDomain` or `mtkLevels`.
levels
  an object of the class `mtkLevels` or a list from which we can create an object of
  the class `mtkLevels`.
```

Value
invisible

Author(s)
Juhui WANG, MIA-jouy, INRA

Examples
```
# Create a `mtkLevels` for a discrete distribution

l <- mtkLevels(type='categorical', levels = c(1,2,3,4,5), weights=rep(0.2, 5))
# Change the levels' name to ('a', 'b', 'c', 'd', 'e')
setLevels(l, c('a','b','c','d','e'))

# Create a new domain with a discrete distribution

d <- mtkDomain(distributionName="discrete", domainNominalValue=3, distributionParameters = list(type='categorical',
levels = c(1,2,3,4,5), weights=rep(0.2, 5)))

# Create a new `mtkLevels` for a discrete distribution and assign it to the domain

l <- mtkLevels(type='categorical', levels = c('a','b','c','d','e'), weights=rep(0.2, 5))
setLevels(d, l)

# Change the domain's levels to type='categorical', levels = c(5,4,3,2,1), weights=rep(0.2, 5)

setLevels(d, levels=list(type='categorical', levels = c(5,4,3,2,1), weights=rep(0.2, 5)))
```
The setName method

Description

Gives a new name to the underlying object

Usage

```
setName(this, name)
```

Arguments

- `this` : the underlying object
- `name` : a string indicating the new name.

Value

`invisible()`

Details

Used by many classes. The behavior depends on the underlying class.

Author(s)

Juhui WANG, MIA-jouy, INRA

Examples

```
# Define a factor
x1 <- make.mtkFactor(name="x1", type="double", distribName="unif",
                     distribPara=list(min=-pi, max=pi))

# Change the numeric value of the factor to "numeric" type.
setName(x1, name="mit")

# Create a new object of mtkValue
d <- mtkValue("a", "double", 0)

# Change the name of the object to "x" type.
setName(d, "x")
```
setParameters-methods

The setParameters method

Description
Assigns a vector of parameters to the process.

Usage
setParameters(this,f)

Arguments
this the underlying object of class mtkProcess
f a vector of mtkParameter.

Value
invisible()

Author(s)
Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References

Examples
# Create a process for experiments design
designer <- mtkNativeDesigner/design ="Morris")

# Create a list of mtkParameter for the parameters: min, max, shape.
p <- make.mtkParameterList(list(size=20))

# Assign the parameters to the process
setParameters(designer, p)
**setProcess-methods**

The *setProcess* method

---

**Description**

Places or replaces a process into the workflow.

**Usage**

```r
setProcess(this, p, name)
```

**Arguments**

- `this`: the underlying object of the class `mtkExpWorkflow`.
- `p`: an object of the class `mtkProcess`.
- `name`: a string from "design", "evaluate", or "analyze" to specify the process to place or replace.

**Value**

`invisible()`

**Details**

This method is especially useful when we need to compare different methods or models.

**Author(s)**

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

**References**


**Examples**

```r
# Analyze the "Ishigami" model with the "Regression" method

x1 <- make.mtkFactor(name="x1", distribName="unif", distribPara=list(min=-pi, max=pi))
x2 <- make.mtkFactor(name="x2", distribName="unif", distribPara=list(min=-pi, max=pi))
x3 <- make.mtkFactor(name="x3", distribName="unif", distribPara=list(min=-pi, max=pi))
ishi.factors <- mtkExpFactors(list(x1,x2,x3))
```
designer <- mtkNativeDesigner("BasicMonteCarlo",
    information=list(size=20))
model <- mtkNativeEvaluator("Ishigami")
analyser <- mtkNativeAnalyser("Regression", information=list(nboot=20))

ishiReg <- mtkExpWorkflow( expFactors=ishi.factors,
    processesVector=c( design=designer,
                       evaluate=model,
                       analyze=analyser)
)
run(ishiReg)
save(ishiReg)

# Re-analyzes the model "Ishigami" with the method "Morris"

# 1) Build a designer with the method "Morris" and put it into the workflow
morris.designer <- mtkNativeDesigner( design="Morris",
    information=list(r=20,type="oat",levels=4,grid.jump=2)
)
setProcess(ishiReg, morris.designer, "design")

# 2) Build an analysis process with the default method and put it
# into the workflow
default.analyser <- mtkDefaultAnalyser()
setProcess(ishiReg, default.analyser, "analyze")

# 3) Run the new workflow
run(ishiReg)
save(ishiReg)

---

**setReady-methods**

*The setReady method*

**Description**

Makes the process ready to run.

**Usage**

```r
setReady(this, switch)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>this</td>
<td>the underlying object of the class <code>mtkProcess</code></td>
</tr>
<tr>
<td>switch</td>
<td>a logical (TRUE or FALSE).</td>
</tr>
</tbody>
</table>
**Value**

invisible()

**Details**

This function is only useful for the programmers who need to program the mtk’s internal functions.

**Author(s)**

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

**References**


**Examples**

```r
# This function is only useful for the programmers
# who need to program the mtk's internal functions.
```

---

**setState-methods**

**The setState method**

**Description**

Marks the state of the process as TRUE when the results produced by the process are available.

**Usage**

```r
setState(this, state)
```

**Arguments**

- `this` the underlying object of the `mtkProcess` class
- `state` a logical (TRUE or FALSE).

**Value**

invisible()

**Details**

This function is only useful for the programmers who need to program the mtk’s internal functions.
Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References


Examples

# This function is only useful for the programmers
# who need to program the mtk's internal functions.

```

```

Description

Gives a new type to the underlying object.

Usage

```r
setType(this, type)
```

Arguments

- `this`  
  the underlying object
- `type`  
  a string indicating the new type for the data. It may be "numeric", "integer", "double", etc.

Value

invisble()

Details

Used by many classes. The behavior depends on the underlying class.

Author(s)

Juhui WANG, MIA-jouy, INRA
Examples

```r
# Define a factor
x1 <- make.mtkFactor(name="x1", type="double", distribName="unif",
                      distribPara=list(min=-pi, max=pi))

# Change the numeric value of the factor to "numeric" type.
set-type(x1, type="numeric")

# Create a new object of mtkValue
d <- mtkValue("a", "double", 0)

# Change the numeric value of the object to "numeric" type.
set-type(d, "numeric")
```

---

**set-Value methods**

*The set-Value method*

Description

Gives a new value to the underlying object

Usage

```
setValue(this, val)
```

Arguments

- **this**: the underlying object of the corresponding class.
- **val**: a new value.

Value

```
invisible()
```

Details

Used by many classes. The behavior depends on the underlying class.

Author(s)

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

References

**Description**

Gives new weights to the discrete distribution associated with the factor’s domain.

**Usage**

```r
setWeights(this, weights)
```

**Arguments**

- `this` is the underlying object of the class to proceed (`mtkLevels`).
- `weights` is a vector of numeric value.

**Value**

`invisible`

**Author(s)**

Juhui WANG, MIA-jouy, INRA

**Examples**

```r
# Create a mtkValue

d <- mtkValue("a", "double", 0)
getValue(d) # gives 0.0

setValue(d, 3.14)
getValue(d) # gives 3.14

# Create a mtkLevels object

l <- mtkLevels(type='categorical', levels=c(1,2,3,4))
setWeights(l, weights=rep(0.25,4))
```
**setXMLFilePath-methods**

*The setXMLFilePath function*

**Description**

Specifies the XML file to parse.

**Usage**

```r
setXMLFilePath(this, xmlPath)
```

**Arguments**

- `this` the underlying object of class `mtkParser`
- `xmlPath` a string indicating the XML file to parse.

**Value**

`invisible()`

**Author(s)**

Juhui WANG, MIA-Jouy, Inra, Juhui.Wang@jouy.inra.fr

**References**


**Examples**

```r
# Specify the XML file's name
xmlFile <- "WWDM_morris.xml"

## Find where the example XML file is held in the 'mtk' package.
## (This line is not useful for real life example!)
xmlFile <- paste0(path.package("mtk", quiet = TRUE), 
"/extdata/", xmlFile, sep = "")

# Create a XML parser.
parser <- mtkParser(xmlFile)

# Create an empty workflow.
workflow <- mtkExpWorkflow()

# Parse the XML file and initialize the workflow
```
The Sobol method

**Description**

A mtk compliant implementation of the Sobol' method for design of experiments and sensitivity analysis.

**Usage**

- `mtkSobolDesigner(listParameters = NULL)`
- `mtkNativeDesigner(design="Sobol", information=NULL)`
- `mtkSobolAnalyser(listParameters = NULL)`
- `mtkNativeAnalyser(analyze="Sobol", information=NULL)`

**Parameters**

- N: the size of the basic samples; the final sample size will be $N^* (k+2)$ where $k$ is the number of the factors to analyze.
- nboot: the number of bootstrap replicates (default 0). See the help on function sobol2002 in the package sensitivity.
conf: the confidence level for bootstrap confidence intervals (default 0.95). See the help on function sobol2002 in the package sensitivity.
sampling: character string specifying the type of sampling method: "MC" (default) for Monte Carlo sampling, "LHS" for Latin Hypercube sampling.
shrink: a scalar or a vector of scalars between 0 and 1 (default 1), specifying shrinkage to be used on the probabilities before calculating the quantiles.

Details

1. The mtk implementation uses the sobol2002 function of the sensitivity package. For further details on the arguments and the behavior, see help(sobol2002, sensitivity).
2. The mtk implementation of the Sobol' method includes the following classes:
   - mtkSobolDesigner: for the Sobol design processes.
   - mtkSobolAnalyser: for Sobol analysis processes.
   - mtkSobolDesignerResult: to store and manage the design.
   - mtkSobolAnalyserResult: to store and manage the analysis results.
3. Many ways to create a Sobol designer are available in mtk, but we recommend the following class constructors: mtkSobolDesigner or mtkNativeDesigner.
4. Many ways to create a Sobol analyser are available in mtk, but we recommend the following class constructors: mtkSobolAnalyser or mtkNativeAnalyser.
5. The Sobol' method is usually used both to build the experiment design and to carry out the sensitivity analysis. In such case, we can use the mtkDefaultAnalyser instead of naming explicitly the method for sensitivity analysis (see example III in the examples section)

References


See Also

help(sobol2002, sensitivity), Quantiles

Examples

```r
## Sensitivity analysis of the "Ishigami" model with the "Sobol" method

# Example I: by using the class constructors: mtkSobolDesigner() and mtkSobolAnalyser()

# Generate the factors
data(Ishigami.factors)

# Build the processes and workflow:

# 1) the design process
expl1.designer <- mtkSobolDesigner( listParameters = list(N=100))

# 2) the simulation process
```
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process
exp1.analyser <- mtkSobolAnalyser()

# 4) the workflow
exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
                      processesVector = c(design=exp1.designer,
                                       evaluate=exp1.evaluator,
                                       analyze=exp1.analyser))

# Run the workflow and reports the results.
run(exp1)
print(exp1)
plot(exp1)

## Example II: by using the class constructors: mtkNativeDesigner() and mtkSobolAnalyser()

# Generate the factors
data(Ishigami.factors)

# Build the processes and workflow:

# 1) the design process
exp1.designer <- mtkNativeDesigner(design = "Sobol", information = list(N=10))

# 2) the simulation process
exp1.evaluator <- mtkNativeEvaluator(model="Ishigami")

# 3) the analysis process with the default method
exp1.analyser <- mtkSobolAnalyser()

# 4) the workflow
exp1 <- mtkExpWorkflow(expFactors=Ishigami.factors,
                      processesVector = c(design=exp1.designer,
                                       evaluate=exp1.evaluator,
                                       analyze=exp1.analyser))

# Run the workflow and reports the results.
run(exp1)
print(exp1)
plot(exp1)

## Example III: by using the class constructors: mtkSobolDesigner() and mtkDefaultAnalyser()

# Generate the factors
data(Ishigami.factors)

# Build the processes and workflow:
The summary method

Description

Returns a summary report of the results produced by the process.

Usage

summary(object, ...)

Arguments

object the underlying object of class mtkProcess.
...

Value

The form of the value returned by summary depends on the sub-class where the method is implemented.

By default, it prints the report on the display device.

Details

1. The behavior of the print depends on the sub-class where the method is implemented.
2. See the documentation of the particular sub-class for details of what is produced.
3. Use methods("summary") to get all the methods for the summary generic.
Author(s)


References


Examples

```r
# Carry out a sensitivity analysis with the Ishigami model

## Input the factors
data(Ishigami.factors)

## Specify the experiments designer
designer <- mtkNativeDesigner("BasicMonteCarlo",
  information=list(size=20))

## Specify the model simulator
model <- mtkIshigamiEvaluator()

## Specify the sensitivity analyser
analyser <- mtkNativeAnalyser("Regression", information=list(nboot=20))

## Specify the workflow
ishiReg <- new("mtkExpWorkflow", expFactors=Ishigami.factors,
  processesVector=c(
    design=designer,
    evaluate=model,
    analyze=analyser)
)

## Run and report a summary of the results produced by the workflow
run(ishiReg)
summary(ishiReg)
```

---

**WWDM**

The **WWDM** (Winter Wheat Dry Matter Model) is a very simple dynamic crop model with a daily time step. It has been developed at INRA (France) by David Makowski, Marie-Hélène Jeuffroy and Martine Guérif.

The behavior of the model is influenced by seven factors:
Eb: Radiation use efficiency
Eimax: Maximal ratio of intercepted to incident radiation
\( K \): Coefficient of extinction
Lmax: Maximal value of the Leaf Area Index (LAI)
A: Coefficient of LAI increase
B: Coefficient of LAI decrease
Tf: Temperature threshold

Details

1. The implementation of the WWDM model includes the object `WWDM_factors` on the input factors, the class `mtkWWDMEvaluator` to run the simulations, and the data frame `wwdm_climates` containing the climate data.

2. In mtk, there are a few ways to build an evaluator of the WWDM model, but we usually recommend the following class constructors: `mtkWWDMEvaluator`, `mtkNativeEvaluator`.

Usage

- `mtkWWDMEvaluator(listParameters=NULL)`
- `mtkNativeEvaluator(model="WWDM",information=NULL)`
- `mtkEvaluator(protocol = "R", site = "mtk", service = "WWDM", parametersList=NULL)`

Parameters used to manage the simulation

- `year` Either NULL or a number between 1 and 14 to specify the number of years to simulate. A database with 14 yearly sequences of meteorological data are included in the environment (data frame `wwdm_climates`).

References


See Also

`help(WWDM_factors)`
Examples

```r
## Evaluation of the "WWDM" model

# Example I: by using the class constructors: mtkWWDMEvaluator()

# Generate the factors
data(WWDM.factors)

# Build the workflow:
# 1) specify the design process
designer <- mtkNativeDesigner("BasicMonteCarlo", information = list(size=50) )

# 2) specify the evaluation process;
model <- mtkWWDMEvaluator(listParameters = list(year=3) )

# 3) specify the workflow with the processes defined previously
exp <- mtkExpWorkflow(expFactors=WWDM.factors, processesVector=c( design=designer, evaluate=model ) )
# Run the workflow and report the results.
run(exp)
summary(exp)

# Personnalize the data reporting
designData <- extractData(exp,name="design")
simulationData <- extractData(exp,name="evaluate")
plot(designData$Eb, simulationData$Biomass, xlab="Eb",ylab="Biomass")

## Example II: by using the class constructor: mtkNativeEvaluator()

# Generate the input factors
data(WWDM.factors)

# Build the workflow:
# 1) specify the design process
designer <- mtkNativeDesigner("BasicMonteCarlo", information = list(size=20) )

# 2) specify the evaluation process;
model <- mtkNativeEvaluator(model="WWDM", information=list(year=3) )

# 3) specify the workflow with the processes defined previously
exp <- mtkExpWorkflow(expFactors=WWDM.factors, processesVector=c( design=designer, evaluate=model ) )
# Run the workflow and report the results.
run(exp)
summary(exp)
plot(exp)
```
**Description**

This dataset gives climatic data needed by the **WWDM** crop model model.

- **ANNE** numeric, year of weather data: from 1 to 14.
- **RG** Global Radiation
- **Tmin** Minimal temperature
- **Tmax** Maximal temperature

**References**


**See Also**

help(**WWDM**)

**Examples**

    data(wwdm.climates)
    summary(wwdm.climates)
    wwdm.climates[1:20,]
    par(mfrow=c(3,1))
    for(i in 1:3) ts.plot(wwdm.climates[ wwdm.climates[,1]==1,1+i],
                          ylab=names(wwdm.climates[1+i]))

**Description**

This dataset gives the input factors and their uncertainty domains involved in the WWDM model.

**Eb**  Radiation use efficiency  
**Eimax**  Maximal ratio of intercepted to incident radiation  
**K**  Coefficient of extinction  
**Lmax**  Maximal value of the Leaf Area Index (LAI)  
**A**  Coefficient of LAI increase  
**B**  Coefficient of LAI decrease  
**TI**  Temperature threshold

**Usage**

```r
data(WWDM.factors)
```

**Format**

an object of the class `mtkExpFactors`.

**References**


**See Also**

`help(WWDM)`

**Examples**

```r
# The code used to generate the WWDM.factors is as follows:
Eb <- make.mtkFactor(name="Eb", distribName="unif", 
nominal=1.85, distribPara=list(min=0.9, max=2.8), unit="g/MJ")
Eimax <- make.mtkFactor(name="Eimax", distribName="unif", 
nominal=0.94, distribPara=list(min=0.9, max=0.99))
K <- make.mtkFactor(name="K", distribName="unif", 
nominal=0.7, distribPara=list(min=0.6, max=0.8))
```
Lmax <- make.mtkFactor(name="Lmax", distribName="unif", 
nominal=7.5, distribPara=list(min=3, max=12), unit="m\u00b2/m\u00b2")
A <- make.mtkFactor(name="A", distribName="unif", 
nominal=0.0065, distribPara=list(min=0.0035, max=0.01))
B <- make.mtkFactor(name="B", distribName="unif", 
nominal=0.00205, distribPara=list(min=0.0011, max=0.0025))
TI <- make.mtkFactor(name="TI", distribName="unif", 
nominal=900, distribPara=list(min=700, max=1100), unit="\u00b0C")

WWDM.factors <- mtkExpFactors(list(Eb,Emax,K,Lmax,A,B,CI))

# To import the WWDM.factors, just use the following line
data(WWDM.factors)
Index

* Topic datasets
  Ishigami.factors, 45
* Topic dataset
  wwdm.climates, 216
  WWDm.factors, 217
 [,mtkExpFactors-method
   (mtkExpFactors-class), 87
 [[,mtkExpFactors-method
   (mtkExpFactors-class), 87
 $,mtkExpFactors-method
   (mtkExpFactors-class), 87

 addProcess, 85, 91
 addProcess (addProcess-methods), 8
 addProcess, mtkExperiment, mtkProcess, character-method
   (mtkExperiment-class), 85
 addProcess, mtkExpWorkflow, mtkProcess, character-method
   (mtkExpWorkflow-class), 91
 addProcess-methods, 8

 ANY, 9, 60, 64, 69, 73, 80, 98, 103, 107, 108, 111, 116, 121, 126, 130, 134, 137, 138, 142, 147, 150, 155, 159, 162, 166, 171, 175, 177

 BasicMonteCarlo, 10

 character, 59, 64, 69, 71, 73, 77, 80, 83, 84, 89, 98, 103, 107, 108, 110, 113, 116, 121, 126, 130, 134, 137, 138, 140, 142, 147, 150, 155, 162, 166, 171, 175, 177

 data.frame, 62, 67, 75, 82, 101, 106, 119, 124, 145, 152, 158, 164, 169, 173, 180
 deleteProcess, 85, 91
 deleteProcess (deleteProcess-methods), 11
 deleteProcess, mtkExperiment, character-method
   (mtkExperiment-class), 85
 deleteProcess, mtkExpWorkflow, character-method
   (mtkExpWorkflow-class), 91
 deleteProcess-methods, 11

 extractData, 85, 91
 extractData (extractData-methods), 13
 extractData, mtkExperiment, character-method
   (mtkExperiment-class), 85
 extractData, mtkExpWorkflow, character-method
   (mtkExpWorkflow-class), 91
 extractData-methods, 13

 Fast, 14

 getdata, 60, 81, 98, 103, 111, 116, 122, 127, 130, 135, 143, 148, 150, 156, 163, 167, 171, 178
 getdata (getdata-methods), 17
 getdata, mtkAnalyser-method
   (mtkAnalyser-class), 59
 getdata, mtkBasicMonteCarloDesigner-method
   (mtkBsicMonteCarloDesigner-class), 64
 getdata, mtkDefaultAnalyser-method
   (mtkDefaultAnalyser-class), 69
 getdata, mtkDesigner-method
   (mtkDesigner-class), 72
 getdata, mtkEvaluator-method
   (mtkEvaluator-class), 80
 getdata, mtkFastAnalyser-method
   (mtkFastAnalyser-class), 97
 getdata, mtkFastDesigner-method
   (mtkFastDesigner-class), 102
 getdata, mtkIshigamiEvaluator-method
   (mtkIshigamiEvaluator-class), 110
 getdata, mtkMorrisAnalyser-method
   (mtkMorrisAnalyser-class), 115
 getdata, mtkMorrisDesigner-method
   (mtkMorrisDesigner-class), 121
getDiscreteDistributionWeights-methods, 20
getDistributionName, 77, 95
gdistributionName
(getDistributionName-methods), 20
gDistributionName, mtkDomain-method
(mtkDomain-class), 77
gDistributionName, mtkFactor-method
(mtkFactor-class), 94
gDistributionName-methods, 20
gDistributionNames
(getDistributionNames-methods), 21
gDistributionNames, mtkExpFactors-method
(mtkExpFactors-class), 87
gDistributionNominalValues-methods, 22
gDistributionNominalValue
(getDistributionNominalValue-methods), 22
gDistributionNominalValue, mtkFactor-method
(mtkFactor-class), 94
gDistributionNominalValue-methods, 23
gDistributionNominalValue-methods, 23
gDistributionNominalValueType, 95
gDistributionNominalValueType
(getDistributionNominalValueType-methods), 95
gDistributionNominalValueType, mtkFactor-method
(mtkFactor-class), 94
gDistributionNominalValueTypes
(getDistributionNominalValueTypes-methods), 24
gDistributionNominalValueTypes-methods, 24
gDistributionParameters, 77, 95
gDistributionParameters
(getDistributionParameters-methods), 77, 95
getDistributionParameters
  (getDistributionParameters-methods), 25
getDistributionParameters, mtkDomain-method
  (mtkDomain-class), 77
getDistributionParameters, mtkExpFactors-method
  (mtkExpFactors-class), 87
getDistributionParameters, mtkFactor-method
  (mtkFactor-class), 94
getDomain, 95
getDomain (getDomain-methods), 25
getDomain, mtkFactor-method
  (mtkFactor-class), 94
getDomain-methods, 26
getFactorFeatures
  (getFactorFeatures-methods), 27
getFactorFeatures, mtkExpFactors-method
  (mtkExpFactors-class), 87
getFactorFeatures-methods, 27
getFactorNames
  (getFactorNames-methods), 28
getFactorNames, mtkExpFactors-method
  (mtkExpFactors-class), 87
getFactorNames-methods, 28
getFactors (getFactors-methods), 29
getFactors, mtkExpFactors-method
  (mtkExpFactors-class), 87
getFactors-methods, 29
getFeatures, 95
getFeatures (getFeatures-methods), 30
getFeatures, mtkFactor-method
  (mtkFactor-class), 94
getFeatures-methods, 30
getLevels, 77, 113
getLevels (getLevels-methods), 30
getLevels, mtkDomain-method
  (mtkDomain-class), 77
getLevels, mtkLevels-method
  (mtkLevels-class), 113
getLevels-methods, 30
getMTKFeatures, 95
getMTKFeatures
  (getMTKFeatures-methods), 31
getMTKFeatures, mtkFactor-method
  (mtkFactor-class), 94
getMTKFeatures-methods, 31
getName, 95, 108, 138, 148, 175
(mtkFastDesigner-class), 102
getParameters, mtkIshigamiEvaluator-method
(mtkIshigamiEvaluator-class), 110
getParameters, mtkMorrisAnalyser-method
(mtkMorrisAnalyser-class), 115
getParameters, mtkMorrisDesigner-method
(mtkMorrisDesigner-class), 121
getParameters, mtkNativeAnalyser-method
(mtkNativeAnalyser-class), 126
getParameters, mtkNativeDesigner-method
(mtkNativeDesigner-class), 129
getParameters, mtkNativeEvaluator-method
(mtkNativeEvaluator-class), 134
getParameters, mtkPLMMAlyser-method
(mtkPLMMAlyser-class), 142
getParameters, mtkProcess-method
(mtkProcess-class), 147
getParameters, mtkRandLHSDesigner-method
(mtkRandLHSDesigner-class), 149
getParameters, mtkRegressionAnalyser-method
(mtkRegressionAnalyser-class), 155
getParameters, mtkSobolAnalyser-method
(mtkSobolAnalyser-class), 162
getParameters, mtkSobolDesigner-method
(mtkSobolDesigner-class), 166
getParameters, mtkSystemEvaluator-method
(mtkSystemEvaluator-class), 170
getParameters, mtkWWDEvaluator-method
(mtkWWDEvaluator-class), 177
getParameters-methods, 35
getProcess, 85, 91
getProcess (getProcess-methods), 36
getProcess, mtkExperiment, character-method
(mtkExperiment-class), 85
getProcess, mtkExpWorkflow, character-method
(mtkExpWorkflow-class), 91
getProcess-methods, 36
getResult, 60, 81, 98, 103, 111, 116, 122,
127, 130, 135, 143, 148, 150, 156,
163, 167, 171, 178
getResult (getResult-methods), 37
getResult, mtkAnalyser-method
(mtkAnalyser-class), 59
getResult, mtkBasicMonteCarloDesigner-method
(mtkBasicMonteCarloDesigner-class), 64
getResult, mtkDefaultAnalyser-method
(mtkDefaultAnalyser-class), 69
getResult, mtkDesigner-method
(mtkDesigner-class), 72
getResult, mtkEvaluator-method
(mtkEvaluator-class), 80
getResult, mtkFastAnalyser-method
(mtkFastAnalyser-class), 97
getResult, mtkFastDesigner-method
(mtkFastDesigner-class), 102
getResult, mtkIshigamiEvaluator-method
(mtkIshigamiEvaluator-class), 110
getResult, mtkMorrisAnalyser-method
(mtkMorrisAnalyser-class), 115
getResult, mtkMorrisDesigner-method
(mtkMorrisDesigner-class), 121
getResult, mtkNativeAnalyser-method
(mtkNativeAnalyser-class), 126
getResult, mtkNativeDesigner-method
(mtkNativeDesigner-class), 129
getResult, mtkNativeEvaluator-method
(mtkNativeEvaluator-class), 134
getResult, mtkPLMMAlyser-method
(mtkPLMMAlyser-class), 142
getResult, mtkProcess-method
(mtkProcess-class), 147
getResult, mtkRandLHSDesigner-method
(mtkRandLHSDesigner-class), 149
getResult, mtkRegressionAnalyser-method
(mtkRegressionAnalyser-class), 155
getResult, mtkSobolAnalyser-method
(mtkSobolAnalyser-class), 162
getResult, mtkSobolDesigner-method
(mtkSobolDesigner-class), 166
getResult, mtkSystemEvaluator-method
(mtkSystemEvaluator-class), 170
getResult, mtkWWDEvaluator-method
(mtkWWDEvaluator-class), 177
getResult-methods, 37
getType, 95, 108, 113, 138, 175
getType (getType-methods), 38
getType, mtkFactor-method
(mtkFactor-class), 94
getType, mtkFeature-method
(mtkFeature-class), 108
getType, mtkLevels-method
INDEX

is.finished, mtkMorrisAnalyser-method
  (mtkMorrisAnalyser-class), 115
is.finished, mtkMorrisDesigner-method
  (mtkMorrisDesigner-class), 121
is.finished, mtkNativeAnalyser-method
  (mtkNativeAnalyser-class), 126
is.finished, mtkNativeDesigner-method
  (mtkNativeDesigner-class), 129
is.finished, mtkNativeEvaluator-method
  (mtkNativeEvaluator-class), 134
is.finished, mtkPLMMAEvans-method
  (mtkPLMMAEvans-class), 142
is.finished, mtkProcess-method
  (mtkProcess-class), 147
is.finished, mtkRandLHSDesigner-method
  (mtkRandLHSDesigner-class), 149
is.finished, mtkRegressionAnalyser-method
  (mtkRegressionAnalyser-class), 155
is.finished, mtkSobolAnalyser-method
  (mtkSobolAnalyser-class), 162
is.finished, mtkSobolDesigner-method
  (mtkSobolDesigner-class), 166
is.finished, mtkSystemEvaluator-method
  (mtkSystemEvaluator-class), 170
is.finished, mtkWWMDEvaluator-method
  (mtkWWMDEvaluator-class), 177
is.finished-methods, 41
is.ready, 60, 80, 98, 103, 111, 116, 121, 127,
  130, 134, 135, 143, 148, 150, 155,
  162, 167, 171, 178
is.ready, mtkAnalyser-method
  (mtkAnalyser-class), 59
is.ready, mtkBasicMonteCarloDesigner-method
  (mtkBMCDesigner-class), 64
is.ready, mtkDefaultAnalyser-method
  (mtkDefaultAnalyser-class), 69
is.ready, mtkDefiner-method
  (mtkDefiner-class), 72
is.ready, mtkEvaluator-method
  (mtkEvaluator-class), 80
is.ready, mtkFastAnalyser-method
  (mtkFAcator-class), 97
is.ready, mtkFastDesigner-method
  (mtkFDesigner-class), 102
is.ready, mtkIshigamiEvaluator-method
  (mtkIshigamiEvaluator-class), 110
initialize, 95
initialize, mtkDomain-method
  (mtkDomain-class), 77
initialize, mtkExpFactors-method
  (mtkExpFactors-class), 87
initialize, mtkFactor-method
  (mtkFactor-class), 94
is.finished, mtkAnalyser-method
  (mtkAnalyser-class), 59
is.finished, mtkBasicMonteCarloDesigner-method
  (mtkBMCDesigner-class), 64
is.finished, mtkDefaultAnalyser-method
  (mtkDefaultAnalyser-class), 69
is.finished, mtkDefiner-method
  (mtkDefiner-class), 72
is.finished, mtkEvaluator-method
  (mtkEvaluator-class), 80
is.finished, mtkFastAnalyser-method
  (mtkFAcator-class), 97
is.finished, mtkFastDesigner-method
  (mtkFDesigner-class), 102
is.finished, mtkIshigamiEvaluator-method
  (mtkIshigamiEvaluator-class), 110

getValnewt(mtkParameter-method
  (mtkParameter-class), 137
getValnewt, 108, 138, 175
getValnewt, mtkParamer-method
  (mtkValue-class), 174
getValnewt, mtkFeature-method
  (mtkFeature-class), 108
getValnewt, mtkValue-method
  (mtkValue-class), 174
getValnewt-methods, 38
getValnewt, mtkDelthu-method
  (mtkDelthu-class), 108
getValnewt, mtkValue-method
  (mtkValue-class), 174
getValnewt-methods, 39
getWeights, 77, 114
getWeights, mtkDomain-method
  (mtkDomain-class), 77
getWeights, mtkLevels-method
  (mtkLevels-class), 113
getWeights-methods, 40
initialize, 95
initialize, mtkDomain-method
  (mtkDomain-class), 77
initialize, mtkExpFactors-method
  (mtkExpFactors-class), 87
initialize, mtkFactor-method
  (mtkFactor-class), 94
is.finished-methods, 41
is.finished, mtkAnalyser-method
  (mtkAnalyser-class), 59
is.finished, mtkBasicMonteCarloDesigner-method
  (mtkBMCDesigner-class), 64
is.finished, mtkDefaultAnalyser-method
  (mtkDefaultAnalyser-class), 69
is.finished, mtkDefiner-method
  (mtkDefiner-class), 72
is.finished, mtkEvaluator-method
  (mtkEvaluator-class), 80
is.finished, mtkFastAnalyser-method
  (mtkFAcator-class), 97
is.finished, mtkFastDesigner-method
  (mtkFDesigner-class), 102
is.finished, mtkIshigamiEvaluator-method
  (mtkIshigamiEvaluator-class), 110

isNreadyLmtkplmmana"ysertMmethod
isNreadyLmtkmorrisana"ysertMmethod
isNreadyLmtkprocessMmethod
isNreadyLmtknativeeva"uatorMmethod
isNreadyLmtknativedesignerMmethod
isNreadyLmtknativeana"ysertMmethod
isNreadyLmtkrandlhsdesignerMmethod
isNfactors
isNreadyMmethods
isNreadyLmtkwwdmeva"uatorMmethod
isNreadyLmtksystemeva"uatorMmethod
isNreadyLmtksobo"designerMmethod
makeNmtkfeaturelist
makeNmtkfactor
143, 45
Ishigami.factors, 43, 45

list, 62, 67, 75, 77, 82–84, 88, 160
is.ready, mtkMorrisAnalyser-method
(mtMorrisAnalyser-class), 110
is.ready, mtkMorrisDesigner-method
(mtMorrisDesigner-class), 115
is.ready, mtkMorrisNativeAnalyser-method
(mtMorrisNativeAnalyser-class), 126
is.ready, mtkMorrisNativeDesigner-method
(mtMorrisNativeDesigner-class), 129
is.ready, mtkMorrisNativeEvaluator-method
(mtMorrisNativeEvaluator-class), 134
is.ready, mtkPLMMAnyser-method
(mtPLMMAnyser-class), 142
is.ready, mtkProcess-method
(mtProcess-class), 147
is.ready, mtkRandLHSDesigner-method
(mtRandLHSDesigner-class), 149
is.ready, mtkRegressionAnalyser-method
(mtRegressionAnalyser-class), 155
is.ready, mtkSobolAnalyser-method
(mtSobolAnalyser-class), 162
is.ready, mtkSobolDesigner-method
(mtSobolDesigner-class), 166
is.ready, mtkSystemAnalyser-method
(mtSystemAnalyser-class), 170
is.ready, mtkSystemEvaluator-method
(mtSystemEvaluator-class), 177
is.ready-methods, 42
Ishigami, 43, 45
Ishigami.factors, 43, 45

mtkAnalyserResult, 58, 60, 61, 61, 70, 98, 100, 118, 127, 145, 158, 164
mtkAnalyserResult-class, 62
mtkBasisMonteCarloDesigner, 10, 63, 63
mtkBasisMonteCarloDesigner-class, 64
mtkBasisMonteCarloDesignerResult, 10, 65, 66, 66, 67
mtkBasisMonteCarloDesignerResult-class, 67
mtkDefaultAnalyser, 15, 49, 68, 68, 69, 210
mtkDefaultAnalyser-class, 69
mtkDesigner, 37, 64, 71, 72, 102, 103, 121, 129, 147, 149, 166
mtkDesigner-class, 72
mtkDesignerResult, 37, 67, 72–74, 74, 75, 103, 106, 124, 130, 152, 160, 168
mtkDesignerResult-class, 75
mtkDomain, 21, 25, 31, 33, 34, 40, 76, 76, 78, 93–95, 195, 196, 199
mtkDomain-class, 77
mtkEvaluator, 78, 79, 110, 134, 147, 170, 177
mtkEvaluator-class, 80
mtkEvaluatorResult, 79, 81, 81, 82, 111, 135, 160, 171, 173, 180
mtkEvaluatorResult-class, 82
mtkExperiment, 83, 84, 85
mtkExperiment-class, 85
mtkExpFactors, 21, 23, 25, 27, 29, 32, 45, 83, 85, 87, 88, 89, 91, 197, 217
mtkExpFactors-class, 87
mtkExpWorkflow, 8, 12, 13, 36, 85, 89, 89, 91, 190, 193, 202
mtkExpWorkflow-class, 91
mtkFactor, 18–22, 24–26, 29–31, 47, 87, 88, 93, 94, 95, 196–198
mtkFactor-class, 94
mtkFastAnalyser, 15, 96, 96, 97
mtkFastAnalyser-class, 97
mtkFastAnalyserResult, 15, 99, 100
mtkFastAnalyserResult-class, 100
mtkFastDesigner, 15, 101, 102, 103
mtkFastDesigner-class, 102
mtkFastDesignerResult, 15, 105, 105, 106
mtkFastDesignerResult-class, 106
mtkFeature, 31, 47, 95, 107, 107, 108, 174, 198
mtkFeature-class, 108
mtkIshigamiEvaluator, 43, 109, 109, 110
mtkIshigamiEvaluator-class, 110
mtkLevels, 31, 40, 77, 112, 112, 113, 114, 199, 207
mtkLevels-class, 113
mtkLevels, 113
mtkMorrisAnalyser, 49, 59, 114, 114, 116
mtkMorrisAnalyser-class, 115
mtkMorrisAnalyserResult, 49, 62, 116, 117, 118
mtkMorrisAnalyserResult-class, 118
mtkMorrisDesigner, 49, 72, 119, 120, 121
mtkMorrisDesigner-class, 121
mtkMorrisDesignerResult, 49, 122, 123, 123, 124
mtkMorrisDesignerResult-class, 124
mtkNativeAnalyser, 15, 49, 59, 125, 125, 126, 127, 182, 210
mtkNativeAnalyser-class, 126
mtkNativeDesigner, 10, 15, 49, 72, 128, 128, 129, 131, 193, 210
mtkNativeDesigner-class, 129
mtkNativeEvaluator, 43, 80, 131, 132, 134, 135, 214
mtkNativeEvaluator-class, 134
mtkParameter, 48, 58, 59, 63, 64, 69, 71, 73, 77, 79, 80, 96, 98, 101, 103, 110, 114, 116, 119, 121, 126, 130, 134, 136, 136, 137, 141, 142, 146, 147, 149, 150, 154, 155, 161, 162, 165, 166, 170, 171, 174, 176, 177, 195, 201
mtkParameter-class, 137
mtk Parsor, 138, 139, 140, 193, 208
mtk Parsor-class, 140
mtkPLMManalyser, 141, 141, 142, 182
mtkPLMManalyser-class, 142
mtkPLMManalyserResult, 62, 143, 144, 144, 145, 182
mtkPLMManalyserResult-class, 145
mtkProcess, 8, 17, 35–37, 41, 42, 59, 72, 80, 85, 89, 91, 146, 146, 148, 184, 186, 192, 193, 201–204, 212
mtkProcess-class, 147
mtkRandLHSDesigner, 149, 149, 150, 188
mtkRandLHSDesigner-class, 149
mtkRandLHSDesignerResult, 150, 151, 151, 152, 188
mtkRandLHSDesignerResult-class, 152
mtkReadFactors
(mtkReadFactors-methods), 153
mtkReadFactors-methods, 153
mtkRegressionAnalyser, 153, 154, 155, 191
mtkRegressionAnalyser-class, 155
mtkRegressionAnalyserResult, 156, 157, 157, 158, 191
mtkRegressionAnalyserResult-class, 158
mtkResult, 37, 62, 75, 82, 146, 159, 159, 160
mtkResult-class, 160
mtkSoboAnalyser, 161, 161, 162, 210
mtkSoboAnalyser-class, 162
mtkSoboAnalyserResult, 163, 163, 164, 210
mtkSoboAnalyserResult-class, 164
mtkSoboDesigner, 165, 165, 166, 210
mtkSoboDesigner-class, 166
mtkSoboDesignerResult, 167, 168, 168, 169, 210
mtkSoboDesignerResult-class, 168
mtkSystemEvaluator, 169, 170
mtkSystemEvaluator-class, 170
mtkSystemEvaluatorResult, 172, 172, 173
mtkSystemEvaluatorResult-class, 173
mtkValue, 39, 76, 77, 108, 137, 174, 174
mtkValue-class, 174
mtkWWDMEvaluator, 80, 175, 176, 177, 214
mtkWWDMEvaluator-class, 177
mtkWWDMEvaluatorResult, 82, 178, 179, 180
mtkWWDMEvaluatorResult-class, 180
NULL, 101, 106, 119, 124, 145, 152, 158, 164, 169, 173, 180
numeric, 14, 113

PLMM, 181
pllm (PLMM), 181
plot (plot.mtkProcess-method), 184
plot, mtkAnalyser-method
(mtkAnalyser-class), 59
plot, mtkAnalyserResult-method
(mtkAnalyserResult-class), 62
plot, mtkBasicMonteCarloDesigner-method
(mtkBasicMonteCarloDesigner-class),
print (mtkBasisMonteCarloDesigner-class), 64
print (mtkBasisMonteCarloDesignerResult-method)
print (mtkBasisMonteCarloDesignerResult-class), 67
print (mtkDefaultAnalyser-class), 69
print (mtkDesigner-class), 72
print (mtkDesignerResult-method)
print (mtkDesignerResult-class), 75
print (mtkDomain-class), 77
print (mtkEvaluator-class), 80
print (mtkEvaluatorResult-method)
print (mtkEvaluatorResult-class), 82
print (mtkExperiment-class), 85
print (mtkExpFactors-class), 87
print (mtkExpWorkFlow-method)
print (mtkExpWorkFlow-class), 91
print (mtkFactor-class), 94
print (mtkFastAnalyser-class), 97
print (mtkFastAnalyserResult-method)
print (mtkFastAnalyserResult-class), 100
print (mtkFastDesigner-class), 102
print (mtkFastDesignerResult-method)
print (mtkFastDesignerResult-class), 106
print (mtkFeature-class), 108
print (mtkIshigamiEvaluator-class), 110
print (mtkLevels-class), 113
print (mtkMorrisAnalyser-class), 115
print (mtkMorrisAnalyserResult-method)
print (mtkMorrisAnalyserResult-class), 118
print (mtkMorrisDesigner-class), 121
print (mtkMorrisDesignerResult-method)
print (mtkMorrisDesignerResult-class), 124
print (mtkNativeAnalyser-class), 126
print (mtkNativeDesigner-class), 129
print (mtkNativeEvaluator-class), 134
print (mtkParameter-class), 137
print (mtkPLMMAnalyser-class), 142
print (mtkPLMMAnalyserResult-method)
print (mtkPLMMAnalyserResult-class), 145
print (mtkProcess-class), 186
print (mtkRandLHSDesigner-class), 149
print (mtkRandLHSDesignerResult-method)
print (mtkRandLHSDesignerResult-class), 152
print (mtkRegressionAnalyser-class), 155
print (mtkRegressionAnalyserResult-method)
print (mtkRegressionAnalyserResult-class), 158
print (mtkSobolAnalyser-class), 162
print (mtkSobolAnalyserResult-method)
print (mtkSobolAnalyserResult-class), 164
print (mtkSobolDesigner-class), 166
print (mtkSobolDesignerResult-method)
print (mtkSobolDesignerResult-class), 168
print (mtkSystemEvaluator-class), 170
print (mtkSystemEvaluatorResult-method)
print (mtkSystemEvaluatorResult-class), 173
print (mtkValue-class), 174
print (mtkWWDMEvaluator-method)
print (mtkWWDMEvaluator-class), 177
INDEX

print, mtkWWDEvaluatorResult-method
  (mtkWWDEvaluatorResult-class), 180

Quantiles, 187

RandLHS, 188
reevaluate, 86, 91
reevaluate (reevaluate-methods), 189
reevaluate.mtkExperiment, character-method
  (mtkExperiment-class), 85
reevaluate.mtkExpWorkflow, character-method
  (mtkExpWorkflow-class), 91
reevaluate-methods, 189
Regression, 190
report, 60, 81, 86, 98, 104, 111, 116, 122,
  127, 130, 135, 143, 148, 150, 156,
  163, 167, 171, 178
report (report-methods), 191
report, mtkAnalyser-method
  (mtkAnalyser-class), 59
report, mtkBasicMonteCarloDesigner-method
  (mtkBasicMonteCarloDesigner-class), 64
report, mtkDefaultAnalyser-method
  (mtkDefaultAnalyser-class), 69
report, mtkDesigner-method
  (mtkDesigner-class), 72
report, mtkEvaluator-method
  (mtkEvaluator-class), 80
report, mtkExperiment-method
  (mtkExperiment-class), 85
report, mtkExpWorkflow-method
  (mtkExpWorkflow-class), 91
report, mtkFastAnalyser-method
  (mtkFastAnalyser-class), 97
report, mtkFastDesigner-method
  (mtkFastDesigner-class), 102
report, mtkIshigamiEvaluator-method
  (mtkIshigamiEvaluator-class), 110
report, mtkMorrisAnalyser-method
  (mtkMorrisAnalyser-class), 115
report, mtkMorrisDesigner-method
  (mtkMorrisDesigner-class), 121
report, mtkNativeAnalyser-method
  (mtkNativeAnalyser-class), 126
report, mtkNativeDesigner-method
  (mtkNativeDesigner-class), 129
report, mtkNativeEvaluator-method
  (mtkNativeEvaluator-class), 134
report, mtkPLMMAlyser-method
  (mtkPLMMAlyser-class), 142
report, mtkProcess-method
  (mtkProcess-class), 147
report, mtkRandLHSDesigner-method
  (mtkRandLHSDesigner-class), 149
report, mtkRegressionAnalyser-method
  (mtkRegressionAnalyser-class), 155
report, mtkSobolAnalyser-method
  (mtkSobolAnalyser-class), 162
report, mtkSobolDesigner-method
  (mtkSobolDesigner-class), 166
report, mtkSystemEvaluator-method
  (mtkSystemEvaluator-class), 170
report, mtkWWDEvaluator-method
  (mtkWWDEvaluator-class), 177
report-methods, 191
run, 60, 81, 86, 98, 103, 111, 116, 122,
  127, 130, 135, 140, 143, 148, 150,
  156, 163, 167, 171, 178
run (run-methods), 193
run, mtkAnalyser, mtkExpWorkflow-method
  (mtkAnalyser-class), 59
run, mtkBasicMonteCarloDesigner, mtkExpWorkflow-method
  (mtkBasicMonteCarloDesigner-class), 64
run, mtkDefaultAnalyser, mtkExpWorkflow-method
  (mtkDefaultAnalyser-class), 69
run, mtkDesigner, mtkExpWorkflow-method
  (mtkDesigner-class), 72
run, mtkEvaluator, mtkExpWorkflow-method
  (mtkEvaluator-class), 80
run, mtkExperiment, missing-method
  (mtkExperiment-class), 85
run, mtkExpWorkflow, missing-method
  (mtkExpWorkflow-class), 91
run, mtkFastAnalyser, mtkExpWorkflow-method
  (mtkFastAnalyser-class), 97
run, mtkFastDesigner, mtkExpWorkflow-method
  (mtkFastDesigner-class), 102
run, mtkIshigamiEvaluator, mtkExpWorkflow-method
  (mtkIshigamiEvaluator-class), 110
run, mtkMorrisAnalyser, mtkExpWorkflow-method
  (mtkMorrisAnalyser-class), 115
run, mtkMorrisDesigner, mtkExpWorkflow-method
  (mtkMorrisDesigner-class), 121
run, mtkNativeAnalyser, mtkExpWorkflow-method
  (mtkNativeAnalyser-class), 126
run, mtkNativeDesigner, mtkExpWorkflow-method
  (mtkNativeDesigner-class), 129
run, mtkNativeEvaluator, mtkExpWorkflow-method
  (mtkNativeEvaluator-class), 134
run, mtkPLMMAlyser, mtkExpWorkflow-method
  (mtkPLMMAlyser-class), 142
run, mtkProcess, mtkExpWorkflow-method
  (mtkProcess-class), 147
run, mtkRandLHSDesigner, mtkExpWorkflow-method
  (mtkRandLHSDesigner-class), 149
run, mtkRegressionAnalyser, mtkExpWorkflow-method
  (mtkRegressionAnalyser-class), 155
run, mtkSobolAnalyser, mtkExpWorkflow-method
  (mtkSobolAnalyser-class), 162
run, mtkSobolDesigner, mtkExpWorkflow-method
  (mtkSobolDesigner-class), 166
run, mtkSystemEvaluator, mtkExpWorkflow-method
  (mtkSystemEvaluator-class), 170
run, mtkWWDEvaluator, mtkExpWorkflow-method
  (mtkWWDEvaluator-class), 177
report-methods, 191
run, 60, 81, 86, 98, 103, 111, 116, 122,
  127, 130, 135, 140, 143, 148, 150,
  156, 163, 167, 171, 178
run (run-methods), 193
run, mtkAnalyser, mtkExpWorkflow-method
  (mtkAnalyser-class), 59
run, mtkBasicMonteCarloDesigner, mtkExpWorkflow-method
  (mtkBasicMonteCarloDesigner-class), 64
run, mtkDefaultAnalyser, mtkExpWorkflow-method
  (mtkDefaultAnalyser-class), 69
run, mtkDesigner, mtkExpWorkflow-method
  (mtkDesigner-class), 72
run, mtkEvaluator, mtkExpWorkflow-method
  (mtkEvaluator-class), 80
run, mtkExperiment, missing-method
  (mtkExperiment-class), 85
run, mtkExpWorkflow, missing-method
  (mtkExpWorkflow-class), 91
run, mtkFastAnalyser, mtkExpWorkflow-method
  (mtkFastAnalyser-class), 97
run, mtkFastDesigner, mtkExpWorkflow-method
  (mtkFastDesigner-class), 102
run, mtkIshigamiEvaluator, mtkExpWorkflow-method
  (mtkIshigamiEvaluator-class), 110
run, mtkMorrisAnalyser, mtkExpWorkflow-method
  (mtkMorrisAnalyser-class), 115
run, mtkMorrisDesigner, mtkExpWorkflow-method
  (mtkMorrisDesigner-class), 121
run, mtkNativeAnalyser, mtkExpWorkflow-method
  (mtkNativeAnalyser-class), 126
run, mtkNativeDesigner, mtkExpWorkflow-method
  (mtkNativeDesigner-class), 129
run, mtkNativeEvaluator, mtkExpWorkflow-method
  (mtkNativeEvaluator-class), 134
run, mtkPLMMAlyser, mtkExpWorkflow-method
  (mtkPLMMAlyser-class), 142
run, mtkProcess, mtkExpWorkflow-method
  (mtkProcess-class), 147
run, mtkRandLHSDesigner, mtkExpWorkflow-method
  (mtkRandLHSDesigner-class), 149
run, mtkRegressionAnalyser, mtkExpWorkflow-method
  (mtkRegressionAnalyser-class), 155
run, mtkSobolAnalyser, mtkExpWorkflow-method
  (mtkSobolAnalyser-class), 162
run, mtkSobolDesigner, mtkExpWorkflow-method
  (mtkSobolDesigner-class), 166
run, mtkSystemEvaluator, mtkExpWorkflow-method
  (mtkSystemEvaluator-class), 170
run, mtkWWDEvaluator, mtkExpWorkflow-method
  (mtkWWDEvaluator-class), 177
report-methods, 191
run, 60, 81, 86, 98, 103, 111, 116, 122,
  127, 130, 135, 140, 143, 148, 150,
  156, 163, 167, 171, 178
run (run-methods), 193
run, mtkAnalyser, mtkExpWorkflow-method
  (mtkAnalyser-class), 59
run, mtkBasicMonteCarloDesigner, mtkExpWorkflow-method
  (mtkBasicMonteCarloDesigner-class), 64
run, mtkDefaultAnalyser, mtkExpWorkflow-method
  (mtkDefaultAnalyser-class), 69
run, mtkDesigner, mtkExpWorkflow-method
  (mtkDesigner-class), 72
run, mtkEvaluator, mtkExpWorkflow-method
  (mtkEvaluator-class), 80
run, mtkExperiment, missing-method
  (mtkExperiment-class), 85
run, mtkExpWorkflow, missing-method
  (mtkExpWorkflow-class), 91
run, mtkFastAnalyser, mtkExpWorkflow-method
  (mtkFastAnalyser-class), 97
run, mtkFastDesigner, mtkExpWorkflow-method
  (mtkFastDesigner-class), 102
run, mtkIshigamiEvaluator, mtkExpWorkflow-method
  (mtkIshigamiEvaluator-class), 110
run, mtkMorrisAnalyser, mtkExpWorkflow-method
  (mtkMorrisAnalyser-class), 115
run, mtkMorrisDesigner, mtkExpWorkflow-method
  (mtkMorrisDesigner-class), 121
run, mtkNativeAnalyser, mtkExpWorkflow-method
  (mtkNativeAnalyser-class), 126
run, mtkNativeDesigner, mtkExpWorkflow-method
  (mtkNativeDesigner-class), 129
run,mktMorrisDesign, mtkExpWorkflow-method
  (mtkMorrisDesign-class), 121
run,mktNativeAnalyser, mtkExpWorkflow-method
  (mtkNativeAnalyser-class), 126
run,mktNativeDesigner, mtkExpWorkflow-method
  (mtkNativeDesigner-class), 129
run,mktNativeEvaluator, mtkExpWorkflow-method
  (mtkNativeEvaluator-class), 134
run,mktParser, mtkExpWorkflow-method
  (mtkParser-class), 140
run,mktPLMMAnalyser, mtkExpWorkflow-method
  (mtkPLMMAnalyser-class), 142
run,mktProcess, mtkExpWorkflow-method
  (mtkProcess-class), 147
run,mktRandLHSDesigner, mtkExpWorkflow-method
  (mtkRandLHSDesigner-class), 149
run,mktRegressionAnalyser, mtkExpWorkflow-method
  (mtkRegressionAnalyser-class), 155
run,mktSobolAnalyser, mtkExpWorkflow-method
  (mtkSobolAnalyser-class), 162
run,mktSobolDesigner, mtkExpWorkflow-method
  (mtkSobolDesigner-class), 166
run,mktSystemEvaluator, mtkExpWorkflow-method
  (mtkSystemEvaluator-class), 170
run,mktWWDMEvaluator, mtkExpWorkflow-method
  (mtkWWDMEvaluator-class), 177
run-methods, 193

serializeOn, mtkFastAnalyser-method
  (mtkFastAnalyser-class), 97
serializeOn,mktFastDesigner-method
  (mtkFastDesigner-class), 102
serializeOn,mktIshigamiEvaluator-method
  (mtkIshigamiEvaluator-class), 110
serializeOn,mktMorrisAnalyser-method
  (mkmMorrisAnalyser-class), 115
serializeOn,mktMorrisDesigner-method
  (mkmMorrisDesigner-class), 121
serializeOn,mktNativeAnalyser-method
  (mkmNativeAnalyser-class), 126
serializeOn,mktNativeDesigner-method
  (mkmNativeDesigner-class), 129
serializeOn,mktNativeEvaluator-method
  (mkmNativeEvaluator-class), 134
serializeOn,mktParser-method
  (mkmParser-class), 140
serializeOn,mktPLMMAnalyser-method
  (mkmPLMMAnalyser-class), 142
serializeOn,mktProcess-method
  (mkmProcess-class), 147
serializeOn,mktRandLHSDesigner-method
  (mkmRandLHSDesigner-class), 149
serializeOn,mktRegressionAnalyser-method
  (mkmRegressionAnalyser-class), 155
serializeOn,mktResult-method
  (mkmResult-class), 160
serializeOn,mktSobolAnalyser-method
  (mkmSobolAnalyser-class), 162
serializeOn,mktSobolDesigner-method
  (mkmSobolDesigner-class), 166
serializeOn,mktSystemEvaluator-method
  (mkmSystemEvaluator-class), 170
serializeOn,mktWWDMEvaluator-method
  (mkmWWDMEvaluator-class), 177
serializeOn-methods, 194

distributionParameters, 78
setDistributionParameters, 78
setDistributionParameters-methods, 195
setDomain, 95
setDomain-methods, 196

setDomain-methods, 196
setFactors (setFactors-methods), 197
setFactors, mtkExpFactors, list-method
  (mtkExpFactors-class), 87
setFactors-methods, 197
setFeatures, 95
setFeatures (setFeatures-methods), 198
setFeatures, mtkFactor, list-method
  (mtkFactor-class), 94
setFeatures-methods, 198
setLevels, 77, 113
setLevels (setLevels-methods), 199
setLevels, mtkDomain, list-method
  (mtkDomain-class), 77
setLevels, mtkDomain, mtkLevels-method
  (mtkLevels-class), 77
setLevels, mtkLevels, vector-method
  (mtkLevels-class), 113
setLevels-methods, 199
setName (setName-methods), 200
setName, mtkAnalyser, character-method
  (mtkAnalyser-class), 59
setName, mtkBasicMonteCarloDesigner, character-method
  (mtkBAnyBasicMonteCarloDesigner-class), 64
setName, mtkDefaultAnalyser, character-method
  (mtkDefaultAnalyser-class), 69
setName, mtkDesigner, character-method
  (mtkDesigner-class), 72
setName, mtkEvaluator, character-method
  (mtkEvaluator-class), 80
setName, mtkFactor, character-method
  (mtkFactor-class), 94
setName, mtkFastAnalyser, character-method
  (mtkFastAnalyser-class), 97
setName, mtkFastDesigner, character-method
  (mtkFastDesigner-class), 102
setName, mtkFeature, character-method
  (mtkFeature-class), 108
setName, mtkIshigamiEvaluator, character-method
  (mtkIshigamiEvaluator-class), 110
setName, mtkMorrisAnalyser, character-method
  (mtkMorrisAnalyser-class), 115
setName, mtkMorrisDesigner, character-method
  (mtkMorrisDesigner-class), 121
setName, mtkNativeAnalyser, character-method
  (mtkNativeAnalyser-class), 126
setName, mtkNativeDesigner, character-method
  (mtkNativeDesigner-class), 126
setName, mtkNativeEvaluator, character-method
  (mtkNativeEvaluator-class), 134
setName, mtkParameter, character-method
  (mtkParameter-class), 137
setName, mtkPLMMAlyser, character-method
  (mtkPLMMAlyser-class), 142
setName, mtkProcess, character-method
  (mtkProcess-class), 147
setName, mtkRandLHSDesigner, character-method
  (mtkRandLHSDesigner-class), 149
setName, mtkRegressionAnalyser, character-method
  (mtkRegressionAnalyser-class), 155
setName, mtkSobolAnalyser, character-method
  (mtkSobolAnalyser-class), 162
setName, mtkSobolDesigner, character-method
  (mtkSobolDesigner-class), 166
setName, mtkSystemEvaluator, character-method
  (mtkSystemEvaluator-class), 170
setName, mtkValue, character-method
  (mtkValue-class), 174
setName, mtkWMDMEvaluator, character-method
  (mtkWMDMEvaluator-class), 177
setName-methods, 200
setParameters, 60, 80, 95, 98, 103, 108, 111, 116, 121, 127, 130, 134, 143, 148, 150, 155, 162, 167, 171, 175, 178
setParameters (setParameters-methods), 201
setParameters, mtkAnalyser, vector-method
  (mtkAnalyser-class), 59
setParameters, mtkBasicMonteCarloDesigner, vector-method
  (mtkBasicMonteCarloDesigner-class), 64
setParameters, mtkDefaultAnalyser, vector-method
  (mtkDefaultAnalyser-class), 69
setParameters, mtkDesigner, vector-method
  (mtkDesigner-class), 72
setParameters, mtkEvaluator, vector-method
  (mtkEvaluator-class), 80
setParameters, mtkFastAnalyser, vector-method
  (mtkFastAnalyser-class), 97
setParameters, mtkFastDesigner, vector-method
  (mtkFastDesigner-class), 102
setParameters, mtkFeature, vector-method
  (mtkFeature-class), 108
setParameters, mtkIshigamiEvaluator, vector-method
  (mtkIshigamiEvaluator-class), 110
setParameters, mtkMorrisAnalyser, vector-method
  (mtkMorrisAnalyser-class), 115
setParameters, mtkMorrisDesigner, vector-method
  (mtkMorrisDesigner-class), 121
INDEX

(setParameters, mtkFastDesigner, vector-method)

(setParameters, mtkIshigamiEvaluator, vector-method)

(setParameters, mtkMorrisAnalyser, vector-method)

(setParameters, mtkMorrisDesigner, vector-method)

(setParameters, mtkNativeAnalyzer, vector-method)

(setParameters, mtkNativeEvaluator, vector-method)

(setParameters, mtkPLMMAlyser, vector-method)

(setParameters, mtkProcess, vector-method)

(setParameters, mtkRandLHSDesigner, vector-method)

(setParameters, mtkRegressionAnalyser, vector-method)

(setParameters, mtkSobolAnalyzer, vector-method)

(setParameters, mtkSobolDesigner, vector-method)

(setParameters, mtkSystemEvaluator, vector-method)

(setParameters, mtkWWDEvaluator, vector-method)

(setParameters-methods)

(setProcess)

(setProcess-setParameters-methods)

(setProcess, mtkExperiment, mtkProcess, characteristics)

(setProcess, mtkExpWorkflow, mtkProcess, characteristics)

(setProcess-methods)

(setReady)

(setReady-setParameters-methods)

(setReady, mtkAnalyzer, logical-method)

(setReady, mtkBasicMonteCarloDesigner, logical-method)

(setReady-me, mtkDefaultAnalyzer, logical-method)
setState, mtkDesigner, logical-method (mtkDesigner-class), 72
setState, mtkEvaluator, logical-method (mtkEvaluator-class), 80
setState, mtkFastAnalyzer, logical-method (mtkFastAnalyzer-class), 97
setState, mtkFastDesigner, logical-method (mtkFastDesigner-class), 102
setState, mtkIshigamiEvaluator, logical-method (mtkiishigamiEvaluator-class), 110
setState, mtkMorrisAnalyzer, logical-method (mtkMorrisAnalyzer-class), 115
setState, mtkMorrisDesigner, logical-method (mtkMorrisDesigner-class), 121
setState, mtkNativeAnalyzer, logical-method (mtkNativeAnalyzer-class), 126
setState, mtkNativeDesigner, logical-method (mtkNativeDesigner-class), 129
setState, mtkNativeEvaluator, logical-method (mtkNativeEvaluator-class), 134
setState, mtkPLMMAlyzer, logical-method (mtkPLMMAlyzer-class), 142
setState, mtkProcess, logical-method (mtkProcess-class), 147
setState, mtkRandLHSDesigner, logical-method (mtkRandLHSDesigner-class), 149
setState, mtkRegressionAnalyzer, logical-method (mtkRegressionAnalyzer-class), 155
setState, mtkSobolAnalyzer, logical-method (mtkSobolAnalyzer-class), 162
setState, mtkSobolDesigner, logical-method (mtkSobolDesigner-class), 166
setState, mtkSystemEvaluator, logical-method (mtkSystemEvaluator-class), 170
setState, mtkWDMAnalyzer, logical-method (mtkWDMAnalyzer-class), 177
setState-methods, 204
setType, 95, 108, 113, 138, 175
setType(setType-methods), 205
setType, mtkFactor, character-method (mtkFactor-class), 94
setType, mtkFeature, character-method (mtkFeature-class), 108
setType, mtkLevels, character-method (mtkLevels-class), 113

(setDefaultAnalyzer-class), 69
(setState, mtkDesigner, logical-method (mtkDesigner-class), 72
setState, mtkEvaluator, logical-method (mtkEvaluator-class), 80
setState, mtkFastAnalyzer, logical-method (mtkFastAnalyzer-class), 97
setState, mtkFastDesigner, logical-method (mtkFastDesigner-class), 102
setState, mtkIshigamiEvaluator, logical-method (mtkiishigamiEvaluator-class), 110
setState, mtkMorrisAnalyzer, logical-method (mtkMorrisAnalyzer-class), 115
setState, mtkMorrisDesigner, logical-method (mtkMorrisDesigner-class), 121
setState, mtkNativeAnalyzer, logical-method (mtkNativeAnalyzer-class), 126
setState, mtkNativeDesigner, logical-method (mtkNativeDesigner-class), 129
setState, mtkNativeEvaluator, logical-method (mtkNativeEvaluator-class), 134
setState, mtkPLMMAlyzer, logical-method (mtkPLMMAlyzer-class), 142
setState, mtkProcess, logical-method (mtkProcess-class), 147
setState, mtkRandLHSDesigner, logical-method (mtkRandLHSDesigner-class), 149
setState, mtkRegressionAnalyzer, logical-method (mtkRegressionAnalyzer-class), 155
setState, mtkSobolAnalyzer, logical-method (mtkSobolAnalyzer-class), 162
setState, mtkSobolDesigner, logical-method (mtkSobolDesigner-class), 166
setState, mtkSystemEvaluator, logical-method (mtkSystemEvaluator-class), 170
setState, mtkWDMAnalyzer, logical-method (mtkWDMAnalyzer-class), 177
setState-methods, 204
setType, 95, 108, 113, 138, 175
setType(setType-methods), 205
setType, mtkFactor, character-method (mtkFactor-class), 94
setType, mtkFeature, character-method (mtkFeature-class), 108
setType, mtkLevels, character-method (mtkLevels-class), 113

(setType, mtkParameter, character-method (mtkParameter-class), 137
setType, mtkValue, character-method (mtkValue-class), 174
setType-methods, 205
setValue, 108, 138, 175
setValue(setValue-methods), 206
setValue, mtkFeature, ANY-method (mtkFeature-class), 108
setValue, mtkParameter, ANY-method (mtkParameter-class), 137
setValue, mtkValue, ANY-method (mtkValue-class), 174
setValue, mtkValue-method (mtkValue-class), 174
setValue-methods, 206
setWeights, 114
setWeights(setWeights-methods), 207
setWeights, mtkLevels, numeric-method (mtkLevels-class), 113
setWeights-methods, 207
setXMLFilePath, 140
setXMLFilePath
(setXMLFilePath-methods), 208
setXMLFilePath, mtkParlor, character-method (mtkParlor-class), 140
setXMLFilePath-methods, 208
show, 78, 88, 95, 108, 114, 138, 175
show, mtkDomain, method (mtkDomain-class), 77
show, mtkExpFactors, method (mtkExpFactors-class), 87
show, mtkFactor, method (mtkFactor-class), 94
show, mtkFeature, method (mtkFeature-class), 108
show, mtkLevels, method (mtkLevels-class), 113
show, mtkParameter, method (mtkParameter-class), 137
show, mtkValue, method (mtkValue-class), 174
Sobol, 209
INDEX

summary (summary, mtkProcess-method), 212
summary, mtkAnalyser-method (mtkAnalyser-class), 59
summary, mtkAnalyserResult-method (mtkAnalyserResult-class), 62
summary, mtkBasicMonteCarloDesigner-method (mtkBasicMonteCarloDesigner-class), 64
summary, mtkBasicMonteCarloDesignerResult-method (mtkBasicMonteCarloDesignerResult-class), 67
summary, mtkDefaultAnalyser-method (mtkDefaultAnalyser-class), 69
summary, mtkDesigner-method (mtkDesigner-class), 72
summary, mtkDesignerResult-method (mtkDesignerResult-class), 75
summary, mtkEvaluator-method (mtkEvaluator-class), 80
summary, mtkEvaluatorResult-method (mtkEvaluatorResult-class), 82
summary, mtkExperiment-method (mtkExperiment-class), 85
summary, mtkExpWorkflow-method (mtkExpWorkflow-class), 91
summary, mtkFastAnalyser-method (mtkFastAnalyser-class), 97
summary, mtkFastAnalyserResult-method (mtkFastAnalyserResult-class), 100
summary, mtkFastDesigner-method (mtkFastDesigner-class), 102
summary, mtkFastDesignerResult-method (mtkFastDesignerResult-class), 106
summary, mtkIshigamiEvaluator-method (mtkIshigamiEvaluator-class), 110
summary, mtkLevels-method (mtkLevels-class), 113
summary, mtkMorrisAnalyser-method (mtkMorrisAnalyser-class), 115
summary, mtkMorrisAnalyserResult-method (mtkMorrisAnalyserResult-class), 118
summary, mtkMorrisDesigner-method (mtkMorrisDesigner-class), 121
summary, mtkMorrisDesignerResult-method (mtkMorrisDesignerResult-class), 124
summary, mtkNativeAnalyser-method (mtkNativeAnalyser-class), 126
summary, mtkNativeDesigner-method (mtkNativeDesigner-class), 129
summary, mtkNativeEvaluator-method (mtkNativeEvaluator-class), 134
summary, mtkPLMMAssay-method (mtkPLMMAssay-class), 142
summary, mtkPLMMAssayResult-method (mtkPLMMAssayResult-class), 145
summary, mtkProcess-method, 212
summary, mtkRandLHSDesigner-method (mtkRandLHSDesigner-class), 149
summary, mtkRandLHSDesignerResult-method (mtkRandLHSDesignerResult-class), 152
summary, mtkRegressionAnalyser-method (mtkRegressionAnalyser-class), 155
summary, mtkRegressionAnalyserResult-method (mtkRegressionAnalyserResult-class), 158
summary, mtkResult-method (mtkResult-class), 160
summary, mtkSobolAnalyser-method (mtkSobolAnalyser-class), 162
summary, mtkSobolAnalyserResult-method (mtkSobolAnalyserResult-class), 164
summary, mtkSobolDesigner-method (mtkSobolDesigner-class), 166
summary, mtkSobolDesignerResult-method (mtkSobolDesignerResult-class), 168
summary, mtkSystemEvaluator-method (mtkSystemEvaluator-class), 170
summary, mtkSystemEvaluatorResult-method (mtkSystemEvaluatorResult-class), 173
summary, mtkWWDMEvaluator-method (mtkWWDMEvaluator-class), 177
summary, mtkWWDMEvaluatorResult-method (mtkWWDMEvaluatorResult-class), 180
vector, 59, 64, 69, 73, 80, 85, 89, 91, 98, 103,
INDEX

110, 113, 116, 121, 126, 130, 134, 142, 147, 150, 155, 162, 166, 171, 177

WWDM, 213, 216, 217
WWDM.climates (wwdm.climates), 216
wwdm.climates, 216
WWDM.factors, 214, 217