mtk for R users

HM

3 septembre 2010

Contents

Contents

1 Introduction

Document de travail sur les fonctions à créer dans mtk pour en faciliter l'utilisation sous R. Le document est présenté sous la forme d'une vignette pour un utilisateur R, mais avec en plus des sous-sections détaillant le contenu des fonctions à développer, sous forme de doc R au format Roxygen.

L'idée est que, après validation si besoin est, il n'y a plus qu'à, d'une part recopier la doc en Roxygen dans les sources R et à adapter le code, d'autre part virer les parties sur Roxygen de ce présent fichier pour obtenir un document utilisateur.

Les propositions ci-dessous s'appuient en grande partie sur les constructeurs déjà prévus pour le parsing du schéma XML et donc aussi sur la structure des classes S4 de mtk. Prendre en compte le point de vue utilisateur sous R consiste essentiellement à prévoir des façons simplifiées de déclarer certains arguments, par exemple un scalaire plutôt qu'un objet de classe mtkValue, ou bien l'ordre, les noms et les valeurs par défaut de certains arguments. Il y a néanmoins une proposition de nouvelle fonction pour créer un ExpWorkFlow, version simplifiée de la fonction mtkExpWorkFlow.

This notice is a brief presentation of mtk for R users. We consider that the user starts from scratch to set a sensitivity analysis. Several objects related to the project must be declared. Their name will start systematically by "wwdm or "ishi" in this vignette.

This is not a comprehensive documentation, see the mtk reference manual or on-line help for more details.

2 Compact syntax for a very simple case

To begin with, we show the syntax in a case when the sensitivity experiment is particularly simple and many default arguments apply. The example is a Morris sensitivity analysis on the Ishigami model implemented in the sensitivity R library. All three factors have the same uncertainty domain.

Three basic `mtk` functions are used:

- `mtkDomain` to declare an uncertainty domain;
- `mtkExpFactors` to declare a list of input factors;
- `mtkExperiment` to specify all the information on the computer experiment to be performed.

```
> library("mtk")
> ishiFactors <- mtkExpFactors(factorNames=c("x1","x2","x3"),
+                              domain=mtkDomain(nominalValue=0,
+                                               distribName="unif",
+                                               distribParameters=list(min=-pi,max=pi)))
> simpleExpe <- mtkExperiment(factors=ishiFactors,
+                 simulator="wwdm",
+                 method="Morris",
+                 methodParameters=list(type="oat",nlevels=5,jump=3,rep=10))
> run(simpleExpe)
```

This simple syntax cannot cope with the large diversity of experiments that can be performed through the `mtk` library. So in the following sections, we present the main functions in more detail.

3 Input factors

The first step is to declare the list of input factors and their uncertainty domains. This is done by creating an object of class `mtkExpFactors`. There is a great flexibility in how this can be done.

3.1 WWDM example

Remarque: Les commandes suivantes sont en gros cohérentes avec la syntaxe actuelle de ***mtkFactor*** *et* ***mtkDomain****, sauf pour l'ordre des arguments. La syntaxe est très détaillée pour le 1er facteur (Eb), puis plus dépouillée pour les facteurs suivants et reposant sur un ordre implicite des arguments dans les fonctions* ***mtkFactor*** *et* ***mtkDomain****. Je ne reprends que les arguments incontournables, les autres sont évoqués plus bas dans les propositions plus détaillées sous format Roxygen.*

The wwdm model is a crop model used in agronomy to predict crop growth. It has many parameters and we assume that these parameters have distinct uncertainty domains. In that case, it may be convenient to first declare an empty object of class `mtkExpFactors`, then to fill it progressively with the input factors of interest. We show a detailed syntax for the first factor and more compact ones for the other factors.

```
> wwdmFactors <- mtkExpFactors()
> wwdmFactors$Eb <- mtkFactor(name="Eb",
+                         domain=mtkDomain(nominalValue=1.85,
+                                          distribName="unif",
+                                          distribParameters=list(min=0.9,max=2.8)))
> myFactors$Eimax <- mtkFactor("Eimax",
+                              mtkDomain(nominalValue=0.94,
+                                        distribName="unif",
+                                        distribParameters=list(min=0.94,max=2.8)))
> myFactors$K <- mtkFactor("K", mtkDomain(0.7, "unif", list(min=0.9,max=2.8)))
> myFactors$Lmax <- mtkFactor("Lmax", mtkDomain(7.5, "unif", list(min=3,max=12)))
> myFactors$A <- mtkFactor("A", mtkDomain(0.0065, "unif", list(min=0.0035,max=0.01)))
> myFactors$B <- mtkFactor("B", mtkDomain(0.00205, "unif", list(min=0.0011,max=0.0025)))
> myFactors$TI <- mtkFactor("TI", mtkDomain(900, "unif", list(min=700,max=1100)))
```

Remarque: il faudrait prévoir une version courte pour simplifier la déclaration sous R, par exemple pour Eb:

A more direct declaration is possible:

```
> wwdmFactors$Eb <- mtkFactor(nominalValue=1.85,
+                             distribName="unif",
+                             distribParameters=list(min=0.9,max=2.8))
> ## or, omitting argument names:
> wwdmFactors$Eb <- mtkFactor("Eb", 1.85, "unif", list(min=0.9,max=2.8))
```

Ici j'ai inclus deux suggestions indépendantes:

- *on passe directement les arguments de construction du domaine à la fonction `mtkFactor`;*
- *on évite (autant que possible) à l'utilisateur de spécifier le nom du facteur dans les arguments, puisqu'il est spécifié dans l'objet à affecter.*

3.2 Ishigami example

We come back to the first example on the Ishigami model. The factors may be declared in several ways.

First factor by factor as for the wwdm model, either explicitly:

```
> ishiFactors <- mtkExpFactors()
> ishiFactors$x1 <- mtkFactor(name="x1",
+                   domain=mtkDomain(0,"unif",list(min=-pi,max=pi)))
> ishiFactors$x2 <- mtkFactor(name="x2",
+                   domain=mtkDomain(0,"unif",list(min=-pi,max=pi)))
> ishiFactors$x3 <- mtkFactor(name="x3",
+                   domain=mtkDomain(0,"unif",list(min=-pi,max=pi)))
```

or, say, by storing information in a mtkDomain object and using a loop:

```
> ishiFactors <- mtkExpFactors()
> ishiDomain <- mtkDomain(nominalValue=0, distribName="unif",
+                   distribParameters=list(min=-pi,max=pi))
> for(i in c("x1","x2","x3")){
+     ishiFactors[[i]] <- mtkFactor(name=i,domain=ishiDomain)
+ }
```

But the first section showed that when all factors have the same uncertainty domain, it is possible to use a shortcut:

```
> ishiDomain <- mtkDomain(0,"unif",list(min=-pi,max=pi))
> ishiFactors <- mtkExpFactors(factorNames=c("x1","x2","x3"), domain=ishiDomain)
```

3.3 Conséquences pour le codage des constructeurs

Les docs des principaux constructeurs pourraient avoir la forme suivante:

3.3.1 mtkExpFactors constructor

```
#' Create or initialise an object of class mtkExpFactors
#' @name mtkExpFactors
#' @param factorList a list of \code{\linkS4class{mtkFactor}} objects
#' @param factorNames a vector of factor names
#' @param domain an object of class \code{\linkS4class{mtkDomain}}
#' @return an object of class \code{\linkS4class{mtkExpFactors}}
#' @note There are two ways to use the function: either by declaring a
#'  (possibly empty) list of \code{\linkS4class{mtkFactor}} objects, or by
#'  declaring factor names and an uncertainty domain. The latter case is
#'  appropriate when all input factors have the same uncertainty domain.
#' @example
#'    ## First way
#'    myFactors <- mtkExpFactors() #an empty object of class mtkExpFactor
#'    myFactors$x1 <- mtkFactor("Eb",mtkDomain(0,"unif",list(min=-pi,max=pi)))
#'    ## Second way
#'    myFactors <- mtkExpFactors(factorNames=c("x1","x2","x3"),domain=mtkDomain(0,"unif",list(min=
#' @seealso \code{\link{mtkFactor}}
#' @export mtkExpFactors
```

3.3.2 mtkFactor constructor

```
#' Create an object of class mtkFactor
#' @name mtkFactor
#' @param name the factor name
#' @param domain an object of class \code{\linkS4class{mtkDomain}}
#' @param nominalValue see the \code{\link{mtkDomain}} function
#' @param distribName  see the \code{\link{mtkDomain}} function
#' @param distribParameters  see the \code{\link{mtkDomain}} function
#' @param id the factor name in the simulation code
#' @param valueType the numeric or alphanumeric type of the factor
#'        levels in the simulation code, for example \code{"integer"} or
'         \code{"float"} or  \code{"double"}, etc.
#' @param units a character string giving the measurement units of the factor levels
#' @param features a named list of additional features of the factor
#' @note
#'   There are two ways to specify the uncertainty domain of the
#'   factor. Either through the \code{domain} argument or more directly by using the
#'   \code{nominalValue}, \code{distribName}, \code{distribParameters},
#'   \code{valueType} arguments as defined in the \code{\link{mtkDomain}}
#'   function.
#'   The \code{id} and \code{valueType} arguments may be required to run the simulation
#'   code appropriately.
#'   The \code{units} argument is used for display in graphics or
#'   reports.
#'   The \code{features} argument is used to specify factor
#'   characteristics that are specific to a given exploration
#'   method.
#' @return an object of class \code{\linkS4class{mtkFactor}}
#' @example
```

```
#'   myFactorX1 <- mtkFactor(name="x1",
#'                domain=mtkDomain(0, "unif", list(min=-pi,max=pi)),
#'                valueType="xs:float",
#'                units="kg")
#'   ## OR
#'   myFactorX1 <- mtkFactor(name="x1", nominalValue=0, distribName="unif",
#'                      distribParameters=list(min=-pi,max=pi),
#'                      valueType="xs:float",
#'                      units="kg")
#' @seealso \code{\link{mtkExpFactors}} \code{\link{mtkDomain}}
#' @export mtkFactor
```

Remarques: `id` doit avoir `name` comme valeur par défaut.

3.3.3 mtkDomain constructor

```
#' Create an object of class mtkDomain to specify the domain of
#' uncertainty of input factors
#' @name mtkDomain
#' @param nominalValue nominal value of the input factor (or NA), given
#'        as a string, a scalar, or an object of class \code{\linkS4class{mtkValue}}
#' @param distribName the name of a probability distribution
#' @param distribParameters a named list of parameter values
#'   adapted to the specified distribution or an object of class \code{\linkS4class{mtkParameter}}
#' @note
#'   The names of the distribution and of its parameters must be given
#'   as defined in the R functions related to probability distributions.
#' @example
#'   mtkDomain(nominalValue=0, distribName="unif", distribParameters=list(min=-pi,max=pi), )
#' @export mtkDomain
```

4 Processes

4.1 Specification of a computer experiment

The information on a whole computer experiment is stored in an object of class `mtkExpWorkFlow`. There are two basic ways to construct such an object. The first one makes use of the `mtkExperiment` function. In a simple case, it looks like that:

```
> wwdmWF <- mtkExperiment(expFactors=wwdmExpFactors,
+                 simulator="wwdm",
+                 method="Morris",
+                 methodParameters=list(type="oat",nlevels=5,jump=3,rep=10))
> run(wwdmWF)
```

The second one is a bit less user-friendly but more generic. It makes use of the `mtkExpWorkFlow` function. In the same simple case, it looks like that:

```
> wwdmWF <- mtkExpWorkFlow(expFactors=wwdmExpFactors,
+                 processesVector=c(
+                 mtkSimulator("wwdm","sensitivity"),
+                 mtkSampler("Morris", list(type="oat",nlevels=5,jump=3,rep=10)),
+                 mtkAnalysor("default"))
+                 )
> run(wwdmWF)
```

4.2 Conséquences pour le codage

4.2.1 Fonction mtkExperiment

```
#' Create an object of class \code{mtkExpWorkFlow} that contains all
#' the information to run a sensitivity analysis or model exploration
#' @name mtkExperiment
#' @param factors an object of class \code{\linkS4class{mtkExpFactors}}
#' @param simulator an object of class \code{\linkS4class{mtkSimulator}}
#'        or the name of a R function available in the present workspace
#' @param method the name of a global method implemented in \code{mtk}
#' @param sampler an object of class \code{\linkS4class{mtkSampler}}
#' @param analysor an object of class \code{\linkS4class{mtkAnalysor}}
#' @note The \code{sampler} and \code{analysor} arguments are not
#'       needed in most cases and when they are used, the \code{method} arguments
#'       are not taken into account. This allows to apply sampling and analysis
#'       methods which are not implemented within the same method in \code{mtk}.
#' @seealso \code{\link{mtkExpWorkFlow}}
#' @export mtkSimulator
```

Remarque: les arguments `sampler` *et* `analysor` *peuvent être envisagés dans un second temps seulement.*

4.2.2 Fonctions mtkExpWorkFlow, mtkSimulator, mtkSampler et mtkAnalysor

Pas forcément besoin de grands changements. Voir quand même l'ordre des arguments, leurs noms, leurs valeurs par défaut. Proposition pour l'une de ces fonctions.

4.2.3 Simulation code

```
#' Create an object of class \code{mtkSimulator}
#' @name mtkSimulator
#' @param service the command or R function that implements the
#'        simulation code
#' @param site the site where the process is implemented if it is remote or
#'        the package where the function is defined if \code{service} is a R function.
#' @param protocol a string from "http", "system", "R" respectively representing if
#'        the process is implemented remotety, locally or as R function.
#' @param simulParameters a named list or a vector of class
#'        [\code{\linkS4class{mtkParameter}}] representing the parameters
#'        necessary to run the process.
#' @param ready a logical to indicate if the process is ready to run.
#' @param result an object of a class derived from  [\code{\linkS4class{mtkSimulatorResult}}] to
#' @param name the processing step associated with this process. It must be "evaluation" for the
#' @return an object of class \code{\linkS4class{mtkSimulator}}
#' @export mtkSimulator
```

Acknowledgements

This vignette was typed using the Sweave package (Leisch, 2002[1]).

References

[1] F. Leisch – "Sweave: Dynamic generation of statistical reports using literate data analysis", *Compstat 2002 — Proceedings in Computational Statistics* (W. Härdle et B. Rönz, éds.), Physica Verlag, Heidelberg, 2002, ISBN 3-7908-1517-9, p. 575–580.