Package 'RBaM'

July 10, 2025

Type Package

Title Bayesian Modeling: Estimate a Computer Model and Make Uncertain Predictions

Version 1.0.1

Description An interface to the 'BaM' (Bayesian Modeling) engine, a 'Fortran'-based executable aimed at estimating a model with a Bayesian approach and using it for prediction, with a particular focus on uncertainty quantification. Classes are defined for the various building blocks of 'BaM' inference (model, data, error models, Markov Chain Monte Carlo (MCMC) samplers, predictions).
The typical usage is as follows:
(1) specify the model to be estimated;
(2) specify the inference setting (dataset, parameters, error models...);
(3) perform Bayesian-MCMC inference;
(4) read, analyse and use MCMC samples;
(5) perform prediction experiments.
Technical details are available (in French) in
Renard (2017) <https://hal.science/hal-02606929v1>.
Examples of applications include
Mansanarez et al. (2019) <doi:10.1029/2018WR023389>,
Le Coz et al. (2021) <doi:10.1002/hyp.14169>,
Perret et al. (2021) <doi:10.1029/2020WR027745>,
Darienzo et al. (2021) <doi:10.1029/2020WR028607> and
Perret et al. (2023) <doi:10.1061/JHEND8.HYENG-13101>.

License GPL-3

Encoding UTF-8

LazyData true

URL https://github.com/BaM-tools/RBaM

BugReports https://github.com/BaM-tools/RBaM/issues

Depends R (>= 4.0.0)

Imports ggplot2, gridExtra, R.utils, utils, grDevices, stats, rjson, rlang, tools, tidyr

Suggests rmarkdown

RoxygenNote 7.2.3

NeedsCompilation no

Author Benjamin Renard [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-8447-5430>),
INRAE [fnd],
Ministère de la Transition Ecologique - SCHAPI [fnd]

Maintainer Benjamin Renard <benjamin.renard@inrae.fr>

Repository CRAN

Date/Publication 2025-07-10 14:40:05 UTC

Contents

BaM *Run BaM*

Description

Run BaM.exe

Usage

```
BaM(
  workspace,
  mod,
  data,
  remnant = rep(list(remnantErrorModel()), mod$nY),
  mcmc = mcmcOptions(),
  cook = mcmcCooking(),
  summary = mcmcSummary(),
  residuals = residualOptions(),
  pred = NULL,
  doCalib = TRUE,
  doPred = FALSE,
  na.value = -9999,
  run = TRUE,
  preClean = FALSE,
  dir.exe = .BAM_PATH,
  name.exe = "BaM",
  predMaster_fname = "Config_Pred_Master.txt"
)
```

Arguments

workspace	Character, directory where config and result files are stored.
mod	model object, the model to be calibrated
data	dataset object, calibration data
remnant	list of remnantErrorModel objects. WARNING: make sure you use a list of length mod$nY (even if mod$nY=1!)
mcmc	mcmcOptions object, MCMC simulation number and options
cook	mcmcCooking object, properties of MCMC cooking (burn and slice)
summary	mcmcSummary object, properties of MCMC summary

residuals	residualOptions object, properties of residual analysis
pred	list of prediction objects, properties of prediction experiments
doCalib	Logical, do Calibration? (mcmc+cooking+summary+residuals)
doPred	Logical, do Prediction?
na.value	numeric, value used for NAs when writing the dataset in BaM format
run	Logical, run BaM? if FALSE, just write config files.
preClean	Logical, start by cleaning up workspace? Be careful, this will delete all files in the workspace, including old results!
dir.exe	Character, directory where BaM executable stands.
name.exe	Character, name of the executable without extension ('BaM' by default).
predMaster_fname	Character, name of configuration file pointing to all prediction experiments.

Value

Nothing: just write config files and runs the executable.

Examples

```
# Fitting a rating curve - see https://github.com/BaM-tools/RBaM
workspace=tempdir()
D=dataset(X=SauzeGaugings['H'],Y=SauzeGaugings['Q'],Yu=SauzeGaugings['uQ'],data.dir=workspace)
# Parameters of the low flow section control: activation stage k, coefficient a and exponent c
k1=parameter(name='k1',init=-0.5,prior.dist='Uniform',prior.par=c(-1.5,0))
a1=parameter(name='a1',init=50,prior.dist='LogNormal',prior.par=c(log(50),1))
c1=parameter(name='c1',init=1.5,prior.dist='Gaussian',prior.par=c(1.5,0.05))
# Parameters of the high flow channel control: activation stage k, coefficient a and exponent c
k2=parameter(name='k2',init=1,prior.dist='Gaussian',prior.par=c(1,1))
a2=parameter(name='a2',init=100,prior.dist='LogNormal',prior.par=c(log(100),1))
c2=parameter(name='c2',init=1.67,prior.dist='Gaussian',prior.par=c(1.67,0.05))
# Define control matrix: columns are controls, rows are stage ranges.
controlMatrix=rbind(c(1,0),c(0,1))
# Stitch it all together into a model object
M=model(ID='BaRatin',
        nX=1,nY=1, # number of input/output variables
        par=list(k1,a1,c1,k2,a2,c2), # list of model parameters
     xtra=xtraModelInfo(object=controlMatrix)) # use xtraModelInfo() to pass the control matrix
# Call BaM to write configuration files. To actually run BaM, use run=TRUE,
# but BaM executable needs to be downloaded first (use downloadBaM())
BaM(workspace=workspace,mod=M,data=D,run=FALSE)
```

`dataset` *dataset object constructor.*

Description

Creates a new instance of a 'dataset' object

Usage

```
dataset(
  X,
  Y,
  data.dir = getwd(),
  data.fname = "CalibrationData.txt",
  fname = "Config_Data.txt",
  Xu = NULL,
  Xb = NULL,
  Xb.indx = NULL,
  Yu = NULL,
  Yb = NULL,
  Yb.indx = NULL,
  VAR.indx = NULL
)
```

Arguments

`X`	data frame, observed input variables.
`Y`	data frame, observed output variables (same number of rows as X).
`data.dir`	Character, directory where a copy of the dataset will be written if required. Default is the current working directory, but you may prefer to use the BaM workspace.
`data.fname`	Character, data file name.
`fname`	Character, configuration file name.
`Xu`	data frame, random uncertainty in X, expressed as a standard deviation. Same dimension as X.
`Xb`	data frame, systematic uncertainty in X, expressed as a standard deviation. Same dimension as X.
`Xb.indx`	data frame, index of systematic errors in X. Same dimension as X.
`Yu`	data frame, random uncertainty in Y, expressed as a standard deviation. Same dimension as Y.
`Yb`	data frame, systematic uncertainty in Y, expressed as a standard deviation. Same dimension as Y.
`Yb.indx`	data frame, index of systematic errors in Y. Same dimension as Y.
`VAR.indx`	data frame, indices used for defining how VAR parameters vary.

Value

An object of class 'dataset'.

Examples

```
X=data.frame(input1=rnorm(100),input2=rnorm(100))
Y=data.frame(output=X$input1+0.8*X$input2+0.1*rnorm(100))
workspace=tempdir()
d <- dataset(X=X,Y=Y,data.dir=workspace)
```

`densityPlot` *densityPlot*

Description

returns a histogram+density ggplot (or a list thereof if several columns in sim)

Usage

```
densityPlot(sim, xlab = "values", col = "black")
```

Arguments

`sim`	vector or matrix or data frame, MCMC simulations
`xlab`	Character, label of x-axis to be used if sim has no names
`col`	Color

Value

A ggplot (or a list thereof if several columns in sim)

Examples

```
# Create Monte Carlo samples
n=1000
sim=data.frame(p1=rnorm(n),p2=rlnorm(n),p3=runif(n))
# create density plot for each component
figures=densityPlot(sim)
```

`downloadBaM` *BaM downloader*

Description

Download BaM executable

Usage

```
downloadBaM(
  destFolder,
  url = NULL,
  os = Sys.info()["sysname"],
  quiet = FALSE,
  ...
)
```

Arguments

`destFolder`	character string, folder where BaM executable will be downloaded.
`url`	character string, the url from which BaM should be downloaded. When NULL, the url is determined automatically by using GitHub API to determine the latest release and the file corresponding to the OS.
`os`	character string, operating system, e.g. 'Linux', 'Windows' or 'Darwin'.
`quiet`	logical, if TRUE, suppress status messages.
`...`	arguments passed to function 'download.file'

Value

nothing - just download the file.

Examples

```
try(downloadBaM(destFolder=tempdir()))
```

`getAwPfromBathy` *Bathymetry interpreter*

Description

Compute Area A(h), width w(h) and wet perimeter P(h) from a bathymetry profile (a,z).

Usage

```
getAwPfromBathy(
  bathy,
  hgrid = seq(min(bathy[, 2]), max(bathy[, 2]), diff(range(bathy[, 2]))/1000),
  segmentLength = sum(sqrt(apply(apply(bathy, 2, diff)^2, 1, sum)))/1000
)
```

Arguments

bathy	data frame, 2 columns containing abscissa (increasing values) and stage.
hgrid	numeric vector, grid of h values where A, w and P are computed. By default 1000 values in the range of bathymetry's z.
segmentLength	numeric, segment length for bathymetry subsampling. By default 1/1000 of the total bathymetry's perimeter.

Value

A 4-column dataframe containing h, A(h), w(h) and P(h)

Examples

```
bathy=data.frame(a=c(0,0,0,1,2,2,4,6,8),h=c(3,2,0,-0.5,0,2,2.0001,2.3,3))
plot(bathy,type='l')
df=getAwPfromBathy(bathy)
plot(df$h,df$A,type='l')
plot(df$h,df$w,type='l')
plot(df$h,df$P,type='l')
```

---

getCatalogue *BaM catalogue*

Description

Distributions and models available in BaM

Usage

```
getCatalogue(printOnly = FALSE)
```

Arguments

printOnly	Logical, should the catalogue be returned or only printed?

Value

If `printOnly==FALSE`, a list with the following fields:

distributions available univariate distributions.

models available models.

Examples

```
catalogue <- getCatalogue()
getCatalogue(printOnly=TRUE)
```

`getNames` *Get object names*

Description

getNames from an object or a list of objects having a $name field (e.g. parameters)

Usage

```
getNames(loo, name = "name")
```

Arguments

`loo`	List Of Objects
`name`	character, string denoting the name field

Value

A character vector containing names

Examples

```
pars <- list(parameter(name='par1',init=0),
             parameter(name='par2',init=0),
             parameter(name='Third parameter',init=0))
getNames(pars)
```

`getParNames` *Get parameter names*

Description

Get parameter names for a distribution d

Usage

```
getParNames(d)
```

Arguments

`d`	Character (possibly vector), distribution (possibly distributions)

Value

A character vector with parameter names.

Examples

```
parnames <- getParNames('GEV')
npar <- length(getParNames('Gumbel'))
```

`mcmcCooking` *mcmcCooking constructor.*

Description

Creates a new instance of a 'mcmcCooking' object

Usage

```
mcmcCooking(
  fname = "Config_Cooking.txt",
  result.fname = "Results_Cooking.txt",
  burn = 0.5,
  nSlim = 10
)
```

Arguments

`fname`	Character, configuration file name.
`result.fname`	Character, result file name.
`burn`	numeric, burn factor, >=0 and <1. 0.4 means the first 40 percent of MCMC samples are discarded).
`nSlim`	Integer, slimming period: 10 means only one MCMC sample every 10 is kept (after burning).

Value

An object of class 'mcmcCooking'.

Examples

```
m <- mcmcCooking()
```

`mcmcOptions` *mcmcOptions object constructor.*

Description

Creates a new instance of a 'mcmcOptions' object

Usage

```
mcmcOptions(
  fname = "Config_MCMC.txt",
  result.fname = "Results_MCMC.txt",
  nAdapt = 100,
  nCycles = 100,
  minMoveRate = 0.1,
  maxMoveRate = 0.5,
  downMult = 0.9,
  upMult = 1.1,
  multFactor = 0.1,
  manualMode = FALSE,
  thetaStd = 9999,
  gammaStd = list(9999)
)
```

Arguments

`fname`	Character, configuration file name.
`result.fname`	Character, result file name.
`nAdapt`	Integer, adaptation period: jump sizes are increased/decreased every Nadapt iterations to comply with the desired moving rates.
`nCycles`	Integer, number of adaptation cycles (total number of iterations is hence Nadapt * Ncycles).
`minMoveRate`	Numeric in (0;1), lower bound for the desired move rate interval.
`maxMoveRate`	Numeric in (0;1), upper bound for the desired move rate interval.
`downMult`	Numeric in (0:1), multiplication factor used to decrease jump size when move rate is too low.
`upMult`	Numeric (>1, avoid 1/dowMult) multiplication factor used to increase jump size when move rate is too high.
`multFactor`	Numeric >0, multiplicative factor to set initial jump standard deviations to multFactor*|initValue| (AUTO mode).
`manualMode`	logical, should jump standard deviations be entered manually?
`thetaStd`	Numeric vector (>0), jump standard deviations for model parameters theta (MANUAL mode).
`gammaStd`	list of numeric vectors (>0), size = number of output variables of the model. Jump standard deviations for structural error parameters gamma of each output variable (MANUAL mode).

Value

An object of class 'mcmcOptions'.

Examples

```
m <- mcmcOptions()
```

`mcmcSummary` *mcmcSummary constructor.*

Description

Creates a new instance of a 'mcmcSummary' object

Usage

```
mcmcSummary(
  fname = "Config_Summary.txt",
  result.fname = "Results_Summary.txt",
  DIC.fname = "Results_DIC.txt",
  xtendedMCMC.fname = ""
)
```

Arguments

`fname`	Character, configuration file name.
`result.fname`	Character, summary file name.
`DIC.fname`	Character, DIC file name. Not computed if empty string.
`xtendedMCMC.fname`	Character, xtended MCMC file name. Not written if empty string.

Value

An object of class 'mcmcSummary'.

Examples

```
m <- mcmcSummary()
```

MeyrasGaugings *Meyras Gaugings*

Description

Stage-discharge gaugings from the hydrometric station 'the Ardèche River at Meyras'. See https://en.wikipedia.org/wiki/Ardè for a description of the river See https://doi.org/10.1029/2018WR023389 for an article using this dataset

Usage

```
MeyrasGaugings
```

Format

A data frame with 104 rows and 4 variables:

h Stage (m)

Q Discharge (m3/s)

uQ Discharge uncertainty (m3/s) expressed as a standard deviation

Period Stability period on which a single rating curve can be used

model *model object constructor.*

Description

Creates a new instance of a 'model' object

Usage

```
model(
  fname = "Config_Model.txt",
  ID = "Linear",
  nX = 1,
  nY = 1,
  par = list(parameter("Xeffect", 1, prior.dist = "FlatPrior")),
  xtra = xtraModelInfo()
)
```

Arguments

fname	Character, configuration file name.
ID	Character, model ID. Type 'getCatalogue()' for available models.
nX	Integer, number of input variables.
nY	Integer, number of output variables.
par	list of parameter objects, parameters of the model.
xtra	xtraModelInfo object.

Value

An object of class 'model'.

Examples

```
# defaut linear regression model Y=aX+b
mod <- model()
# BaRatin model for a single-control rating curve Y=a(X-b)^c
mod <- model(ID='BaRatin',nX=1,nY=1,
             par=list(parameter('a',10,prior.dist='LogNormal',prior.par=c(log(10),0.1)),
                      parameter('b',-1,prior.dist='Gaussian',prior.par=c(-1,1)),
                      parameter('c',5/3,prior.dist='Gaussian',prior.par=c(5/3,0.05))),
             xtra=xtraModelInfo(object=matrix(1,nrow=1,ncol=1)))
```

---

parameter *parameter object constructor.*

Description

Creates a new instance of a 'parameter' object

Usage

```
parameter(name, init, prior.dist = "FlatPrior", prior.par = NULL)
```

Arguments

name	character, parameter name.
init	numeric, initial guess.
prior.dist	character, prior distribution.
prior.par	numeric vector, prior parameters

Value

An object of class 'parameter'.

Examples

```
p <- parameter(name='par',init=0,prior.dist='Gaussian',prior.par=c(0,1))
```

parameter_VAR *Varying parameter object constructor.*

Description

Creates a new instance of a 'parameter_VAR' object

Usage

```
parameter_VAR(
  name,
  index,
  d,
  init,
  prior.dist = rep("FlatPrior", length(init)),
  prior.par = rep(list(NULL), length(init))
)
```

Arguments

name	character, parameter name.
index	character, name of column in VAR.indx (see ?dataset) containing the index for this varying parameter
d	dataset object, the dataset containing (amongst other things) the index above
init	numeric vector, initial guesses for each instance of the VAR parameter.
prior.dist	character vector, prior distribution for each instance of the VAR parameter.
prior.par	list of numeric vectors, prior parameters for each instance of the VAR parameter

Value

An object of class 'parameter_VAR'.

Examples

```
X=data.frame(input1=rnorm(100),input2=rnorm(100))
Y=data.frame(output=X$input1+0.8*X$input2+0.1*rnorm(100))
VAR.indx=data.frame(indx=c(rep(1,50),rep(2,50)))
workspace=tempdir()
d <- dataset(X=X,Y=Y,data.dir=workspace,VAR.indx=VAR.indx)
p <- parameter_VAR(name='par',index='indx',d=d,
                   init=c(-1,1,2),
                   prior.dist=c('Gaussian','FlatPrior','Triangle'),
                   prior.par=list(c(-1,1),NULL,c(2,0,5)))
```

`prediction` *prediction object constructor.*

Description

Creates a new instance of a 'prediction' object

Usage

```
prediction(
  X,
  spagFiles,
  data.dir = getwd(),
  data.fnames = paste0("X", 1:length(X), ".pred"),
 fname = paste0("Config_Pred_", paste0(sample(c(letters, LETTERS, 0:9), 6), collapse =
   ""), ".txt"),
  doParametric = FALSE,
  doStructural = rep(FALSE, length(spagFiles)),
  transposeSpag = TRUE,
  priorNsim = NULL,
  envFiles = paste0(tools::file_path_sans_ext(spagFiles), ".env"),
  consoleProgress = TRUE,
  spagFiles_state = NULL,
  transposeSpag_state = TRUE,
  envFiles_state = switch(is.null(spagFiles_state) + 1,
    paste0(tools::file_path_sans_ext(spagFiles_state), ".env"), NULL),
  parSamples = NULL
)
```

Arguments

`X`	data frame or list of dataframes / matrices, representing the values taken by the input variables. • If X is a dataframe, then each column is interpreted as one input variable, and consequently inputs are not replicated (=> no input uncertainty). • If X is a list, then each element of the list is a matrix associated with one input variable, and the columns of this matrix are replications (=> input uncertainty is propagated). All matrices in the list should have the same number of rows, columns are recycled if needed.
`spagFiles`	Character vector (size nY, the number of output variables). Name of the files containing the spaghettis for each output variable. NOTE: provide file names only, not full paths. Using a '.spag' extension is a good practice.
`data.dir`	Character, directory where a copies of the dataset X will be written if required (1 file per variable in X). Default is the current working directory, but you may prefer to use the BaM workspace.
`data.fnames`	Character, data file names.

fname	Character, configuration file name.
doParametric	Logical, propagate parametric uncertainty? If FALSE, maxpost parameters are used.
doStructural	Logical, propagate structural uncertainty for each output variable? (size nY)
transposeSpag	Logical. If FALSE, spaghettis are written horizontally (row-wise), otherwise they will be transposed so that each spaghetti is a column.
priorNsim	Integer, number of samples from the prior distribution for 'prior prediction' experiments. If negative or NULL (default), posterior samples are used.
envFiles	Character vector (size nY, the number of output variables). Name of the files containing the envelops (e.g. prediction intervals) computed from the spaghettis for each output variable. By default, same name as spaghetti files but with a '.env' extension. If NULL, envelops are not computed.
consoleProgress	Logical, print progress in BaM.exe console?
spagFiles_state	Character vector (size nState, the number of state variables), same as spagFiles but for states rather than outputs. If NULL, states are not predicted. Note that only parametric uncertainty is propagated for state variables since they are not observed. Consequently, structural uncertainty = 0 and total uncertainty = parametric uncertainty.
transposeSpag_state	Logical. Same as transposeSpag, but for states rather than outputs.
envFiles_state	Character vector (size nState, the number of state variables), same as envFiles, but for states rather than outputs.
parSamples	data frame, parameter samples that will replace the MCMC-generated one for this prediction.

Value

An object of class 'prediction'.

Examples

```
#--------------
# Example using the twoPopulations dataset, containing 101 values for
# 3 input variables (time t, temperature at site 1 T1, temperature at site 2 T2)
# and 2 output variables (population at site 1 P1, population at site 2 P2).
pred=prediction(X=twoPopulations[,1:3],spagFiles=c('P1.spag','P2.spag'))
#--------------
# Alternative example showing how to propagate uncertainty in some of
# the input variables (here, temperatures T1 and T2)
# Create 100 noisy replicates for T1, representing uncertainty
T1rep=matrix(rnorm(101*100,mean=twoPopulations$T1,sd=0.1),nrow=101,ncol=100)
# Same for T2
T2rep=matrix(rnorm(101*100,mean=twoPopulations$T2,sd=0.1),nrow=101,ncol=100)
# Create prediction object
pred=prediction(X=list(twoPopulations$t,T1rep,T2rep),spagFiles=c('P1.spag','P2.spag'))
```

readMCMC *MCMC Reader*

Description

Read raw MCMC samples, return cooked (burnt & sliced) ones

Usage

```
readMCMC(
  file = "Results_Cooking.txt",
  burnFactor = 0,
  slimFactor = 1,
  sep = "",
  reportFile = NULL,
  panelPerCol = 10,
  panelHeight = 3,
  panelWidth = 23/panelPerCol
)
```

Arguments

file	Character, full path to MCMC file.
burnFactor	Numeric, burn factor. 0.1 means the first 10 are discarded.
slimFactor	Integer, slim factor. 10 means that only one iteration every 10 is kept.
sep	Character, separator used in MCMC file.
reportFile	Character, full path to pdf report file, not created if NULL
panelPerCol	Integer, max number of panels per column
panelHeight	Numeric, height of each panel
panelWidth	Numeric, width of each panel

Value

A data frame containing the cooked mcmc samples.

Examples

```
# Create Monte Carlo samples and write them to file
n=4000
sim=data.frame(p1=rnorm(n),p2=rlnorm(n),p3=runif(n))
workspace=tempdir()
write.table(sim,file=file.path(workspace,'MCMC.txt'),row.names=FALSE)
# Read file, burn the first half and keep every other row
M=readMCMC(file=file.path(workspace,'MCMC.txt'),burnFactor=0.5,slimFactor=2)
dim(M)
```

remnantErrorModel *remnantErrorModel object constructor.*

Description

Creates a new instance of a 'remnantErrorModel' object

Usage

```
remnantErrorModel(
  fname = "Config_RemnantSigma.txt",
  funk = "Linear",
  par = list(parameter("g1", 1, prior.dist = "FlatPrior+"), parameter("g2", 0.1,
    prior.dist = "FlatPrior+"))
)
```

Arguments

fname	Character, configuration file name.
funk	Character, function f used in remnant sdev = f(Ysim). Available: 'Constant', 'Proportional', 'Linear' (default), 'Exponential', 'Gaussian'.
par	list of parameter objects, parameters of the function above. respectively, npar= 1,1,2,3,3

Value

An object of class 'remnantErrorModel'.

Examples

```
r <- remnantErrorModel()
```

residualOptions *residualOptions constructor.*

Description

Creates a new instance of a 'residualOptions' object

Usage

```
residualOptions(
  fname = "Config_Residuals.txt",
  result.fname = "Results_Residuals.txt"
)
```

Arguments

fname	Character, configuration file name.
result.fname	Character, result file name.

Value

An object of class 'residualOptions'.

Examples

```
r <- residualOptions()
```

---

runOptions *runOptions constructor.*

Description

Creates a new instance of a 'runOptions' object

Usage

```
runOptions(
  fname = "Config_RunOptions.txt",
  doMCMC = TRUE,
  doSummary = TRUE,
  doResiduals = TRUE,
  doPrediction = FALSE
)
```

Arguments

fname	Character, configuration file name.
doMCMC	logical, do MCMC sampling?
doSummary	logical, do MCMC summarizing?
doResiduals	logical, do residuals analysis?
doPrediction	logical, do prediction experiments?

Value

An object of class 'runOptions'.

Examples

```
o <- runOptions()
```

SauzeGaugings *Sauze Gaugings*

Description

Stage-discharge gaugings from the hydrometric station 'the Ardèche River at Sauze-St-Martin'. See https://en.wikipedia.org/wiki/Ardèche_(river) for a description of the river See https://hal.science/hal-00934237 for an article using this dataset

Usage

```
SauzeGaugings
```

Format

A data frame with 38 rows and 3 variables:

H Stage (m)

Q Discharge (m3/s)

uQ Discharge uncertainty (m3/s) expressed as a standard deviation

setPathToBaM *Path to BaM*

Description

Set path to BaM executable

Usage

```
setPathToBaM(dir.exe, quiet = FALSE)
```

Arguments

dir.exe	character string, folder where BaM executable is located. A NULL value resets BaM directory to 'unknown' by removing the config folder where it was stored on your computer (use 'tools::R_user_dir(package="RBaM",which="config")' to locate this folder).
quiet	logical, if TRUE, suppress status messages.

Value

nothing - just write a config file.

Examples

```
setPathToBaM(dir.exe=tempdir())
```

toString.dataset *dataset to string*

Description

Convert an object of class 'dataset' into a ready-to-write vector of string

Usage

```
## S3 method for class 'dataset'
toString(x, ...)
```

Arguments

x	dataset object, object to be converted.
...	Optional arguments.

Value

A string ready to be printed or written.

Examples

```
X=data.frame(input1=rnorm(100),input2=rnorm(100))
Y=data.frame(output=X$input1+0.8*X$input2+0.1*rnorm(100))
workspace=tempdir()
d <- dataset(X=X,Y=Y,data.dir=workspace)
toString(d)
```

toString.mcmcCooking *mcmcCooking to string*

Description

Convert an object of class 'mcmcCooking' into a ready-to-write vector of string

Usage

```
## S3 method for class 'mcmcCooking'
toString(x, ...)
```

Arguments

x	mcmcCooking object, object to be converted.
...	Optional arguments.

Value

A string ready to be printed or written.

Examples

```
toString(mcmcCooking())
```

`toString.mcmcOptions` *mcmcOptions to string*

Description

Convert an object of class 'mcmcOptions' into a ready-to-write vector of string

Usage

```
## S3 method for class 'mcmcOptions'
toString(x, ...)
```

Arguments

`x`	mcmcOptions object, object to be converted.
`...`	Optional arguments.

Value

A string ready to be printed or written.

Examples

```
toString(mcmcOptions())
```

`toString.mcmcSummary` *mcmcSummary to string*

Description

Convert an object of class 'mcmcSummary' into a ready-to-write vector of string

Usage

```
## S3 method for class 'mcmcSummary'
toString(x, ...)
```

Arguments

x	mcmcSummary object, object to be converted.
...	Optional arguments.

Value

A string ready to be printed or written.

Examples

```
toString(mcmcSummary())
```

---

toString.model *model to string*

Description

Convert an object of class 'model' into a ready-to-write vector of string

Usage

```
## S3 method for class 'model'
toString(x, ...)
```

Arguments

x	model object, object to be converted.
...	Optional arguments.

Value

A string ready to be printed or written.

Examples

```
toString(model())
```

toString.parameter *parameter to string*

Description

Convert an object of class 'parameter' into a ready-to-write vector of string

Usage

```
## S3 method for class 'parameter'
toString(x, ...)
```

Arguments

x	parameter object, object to be converted.
...	Optional arguments.

Value

A string ready to be printed or written.

Examples

```
p <- parameter(name='par',init=0,prior.dist='Gaussian',prior.par=c(0,1))
toString(p)
```

toString.parameter_VAR *parameter_VAR to string*

Description

Convert an object of class 'parameter_VAR' into a ready-to-write vector of string

Usage

```
## S3 method for class 'parameter_VAR'
toString(x, ...)
```

Arguments

x	parameter_VAR object, object to be converted.
...	Optional arguments.

Value

A string ready to be printed or written.

Examples

```
X=data.frame(input1=rnorm(100),input2=rnorm(100))
Y=data.frame(output=X$input1+0.8*X$input2+0.1*rnorm(100))
VAR.indx=data.frame(indx=c(rep(1,50),rep(2,50)))
workspace=tempdir()
d <- dataset(X=X,Y=Y,data.dir=workspace,VAR.indx=VAR.indx)
p <- parameter_VAR(name='par',index='indx',d=d,
                   init=c(-1,1,2),
                   prior.dist=c('Gaussian','FlatPrior','Triangle'),
                   prior.par=list(c(-1,1),NULL,c(2,0,5)))
toString(p)
```

---

toString.prediction *prediction to string*

Description

Convert an object of class 'prediction' into a ready-to-write vector of string

Usage

```
## S3 method for class 'prediction'
toString(x, ...)
```

Arguments

x	prediction object, object to be converted.
...	Optional arguments.

Value

A string ready to be printed or written.

Examples

```
pred=prediction(X=twoPopulations[,1:3],spagFiles=c('P1.spag','P2.spag'))
toString(pred)
```

toString.remnantErrorModel *remnantErrorModel to string*

Description

Convert an object of class 'remnantErrorModel' into a ready-to-write vector of string

Usage

```
## S3 method for class 'remnantErrorModel'
toString(x, ...)
```

Arguments

x	remnantErrorModel object, object to be converted.
...	Optional arguments.

Value

A string ready to be printed or written.

Examples

```
toString(remnantErrorModel())
```

toString.residualOptions *residualOptions to string*

Description

Convert an object of class 'residualOptions' into a ready-to-write vector of string

Usage

```
## S3 method for class 'residualOptions'
toString(x, ...)
```

Arguments

x	residualOptions object, object to be converted.
...	Optional arguments.

Value

A string ready to be printed or written.

Examples

```
toString(residualOptions())
```

`toString.runOptions` *runOptions to string*

Description

Convert an object of class 'runOptions' into a ready-to-write vector of string

Usage

```
## S3 method for class 'runOptions'
toString(x, ...)
```

Arguments

`x`	runOptions object, object to be converted.
`...`	Optional arguments.

Value

A string ready to be printed or written.

Examples

```
toString(runOptions())
```

`tracePlot` *MCMC reporting*

Description

2DO (adapt from STooDs): Generate pdf report files summarizing mcmc samples

Usage

```
tracePlot(sim, ylab = "values", keep = NULL, col = "black", psize = 0.5)
```

Arguments

sim	vector or matrix or data frame, MCMC simulations
ylab	Character, label of y-axis to be used if sim has no names
keep	Integer vector, indices of samples to be kept in cooked MCMC sample
col	Color
psize	Numeric, point size

Details

tracePlot

returns a trace plot ggplot (or a list thereof if several columns in sim)

Value

A ggplot (or a list thereof if several columns in sim)

Examples

```
# Create Monte Carlo samples
n=1000
sim=data.frame(p1=rnorm(n),p2=rlnorm(n),p3=runif(n))
# create trace plot for each component
figures=tracePlot(sim)
```

---

twoPopulations *Evolution of two populations*

Description

Size of two populations of the same species put in two different environments, as a function of time and local temperature.

- Input variables: time t, temperature at site 1 T1, temperature at site 2 T2.
- Output variables: population size at site 1 P1, population size at site 2 P2.
- Data are synthetically generated from a logistic model.

Usage

```
twoPopulations
```

Format

An object of class `data.frame` with 101 rows and 5 columns.

violinPlot *violinPlot*

Description

returns a violinplot ggplot

Usage

```
violinPlot(sim, ylab = "values", col = "black")
```

Arguments

sim	vector or matrix or data frame, MCMC simulations
ylab	Character, label of y-axis
col	Color

Value

A ggplot

Examples

```
# Create Monte Carlo samples
n=1000
sim=data.frame(p1=rnorm(n),p2=rlnorm(n),p3=runif(n))
# create violin plot comparing all components
figure=violinPlot(sim)
```

writePredInputs *Write prediction inputs*

Description

Write input data of the prediction into files

Usage

```
writePredInputs(o)
```

Arguments

o	prediction object

Value

nothing - just write to files.

Examples

```
temp=tempdir()
pred=prediction(X=twoPopulations[,1:3],spagFiles=c('P1.spag','P2.spag'),
                data.dir=temp)
writePredInputs(pred)
```

`xtraModelInfo` *xtraModelInfo constructor.*

Description

Creates a new instance of a 'xtraModelInfo' object containing extra model information

Usage

```
xtraModelInfo(fname = "Config_Xtra.txt", object = NULL)
```

Arguments

`fname`	Character, configuration file name.
`object`	any R object containing xtra model info - typically a list of stuff. The content and meaning of 'object' is completely model-specific.

Value

An object of class 'xtraModelInfo'.

Examples

```
x <- xtraModelInfo()
```

Index