Package 'airGRiwrm'

November 8, 2026

Title 'airGR' Integrated Water Resource Management

Version 0.8.1

Description Semi-distributed Precipitation-Runoff Modeling based on 'airGR' package models integrating human infrastructures and their managements.

License AGPL-3

URL https://inrae.github.io/airGRiwrm/, https://github.com/inrae/airGRiwrm#readme

BugReports https://github.com/inrae/airGRiwrm/issues

Depends airGR (>= 1.7.8), R (>= 3.5)

Imports dplyr, graphics, grDevices, jsonlite, png, rlang, stats, utils, zlib

Suggests htmltools, knitr, lubridate, rmarkdown, spelling, testthat, waldo

VignetteBuilder knitr

Encoding UTF-8

Language en-US

LazyData true

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author David Dorchies [aut, cre] (ORCID: <https://orcid.org/0000-0002-6595-7984>), Olivier Delaigue [ctb] (ORCID: <https://orcid.org/0000-0002-7668-8468>), Guillaume Thirel [ctb] (ORCID: <https://orcid.org/0000-0002-1444-1830>)

Maintainer David Dorchies <david.dorchies@inrae.fr>

Repository CRAN

Date/Publication 2026-11-08 13:50:02 UTC

Contents

adjustMeteoToCatchment . . . 3
as.Qm3s . . . 5
Calibration.GRiwrmInputsModel . . . 6
ConvertMeteoSD . . . 9
CreateCalibOptions.GRiwrmInputsModel . . . 10
CreateController . . . 15
CreateGRiwrm . . . 16
CreateInputsCrit.GRiwrmInputsModel . . . 20
CreateInputsModel . . . 23
CreateInputsModel.GRiwrm . . . 24
CreateRunOptions.GRiwrmInputsModel . . . 27
CreateSupervisor . . . 29
extractParam . . . 30
getDefaultAprioriIds . . . 31
getNextTimeSteps . . . 32
getNodeProperties . . . 32
getNodeRanking . . . 34
getSD_Ids . . . 35
isNodeDownstream . . . 35
merge.OutputsModel . . . 36
mermaid . . . 37
plot.GRiwrm . . . 38
plot.GRiwrmOutputsModel . . . 41
plot.OutputsModelReservoir . . . 43
plot.Qm3s . . . 49
reduceGRiwrm . . . 51
RunModel . . . 52
RunModel.GR . . . 52
RunModel.GRiwrmInputsModel . . . 53
RunModel.GRiwrmOutputsModel . . . 59
RunModel.InputsModel . . . 60
RunModel.Supervisor . . . 61
RunModel_Reservoir . . . 64
Severn . . . 72
sort.GRiwrm . . . 73
transferGRparams . . . 73

Index **75**

`adjustMeteoToCatchment` *Compute the meteorological data of a node at the whole-catchment scale*

Description

Replace, on a copy, the meteorological time series of a node by their area-weighted mean over the whole catchment of the node (the node and all its upstream nodes). The original object is not modified: the time series of the object passed in `OutputsModel` are read only.

Usage

```
adjustMeteoToCatchment(
  OM_id,
  id,
  OutputsModel,
  griwrm = attr(OutputsModel, "GRiwrm")
)
```

Arguments

`OM_id`	[object of class *OutputsModel*] the *OutputsModel* object of the node `id` (see RunModel.GRiwrmInputsModel)
`id`	[character] the ID of the node (See CreateGRiwrm)
`OutputsModel`	[object of class *GRiwrmOutputsModel*] the object produced by RunModel.GRiwrmInputsModel, or a list of *OutputsModel* objects with one item by node (names are the node IDs). When a simple list is passed, it must carry the "GRiwrm" attribute or `griwrm` must be provided explicitly (see below)
`griwrm`	[object of class *GRiwrm*] the network of the simulation (See CreateGRiwrm). Defaults to the "GRiwrm" attribute of `OutputsModel` (set by RunModel.GRiwrmInputsModel). When `OutputsModel` is a list which does not carry this attribute, the default resolves to `NULL` and `griwrm` must be passed explicitly, otherwise the catchment of the node cannot be identified.

Details

The meteorological data stored in the *OutputsModel* object of a node come from the meteorological data of its local sub-basin only, whereas the simulated discharge of the node integrates the flows of its whole catchment. This function computes, for each time step, the area-weighted mean of these data over the whole catchment, so that they can be used to compute statistics at the catchment scale (or displayed, see plot.GRiwrmOutputsModel).

The following time series are replaced when they are available in the *OutputsModel* object of the node:

- `Precip` and `PotEvap`;

- for a CemaNeige node, the per-layer series `Temp` (mean air temperature), `Psol` (solid precipitation) and `SnowPack` (snow pack state) stored in `OM_id$CemaNeigeLayers`, one item by elevation layer (See CreateRunOptions).

Each node contributes to the mean with its own time series weighted by the area of its local sub-basin: the part of the catchment for which the meteorological data of the node are the "real" data (See `getNodeLocalArea`). A node cannot contribute when its local sub-basin area is unknown or zero, or when the concerned time series is not available; a missing value in a time series excludes the node from the mean at the concerned time steps. When no node can contribute, the time series of the node `id` itself are returned unchanged (graceful behaviour). When the ID of the node is shared with a Diversion node (two rows of the network with the same `id`, see CreateGRiwrm), the local sub-basin area of the shared ID cannot be computed and the node is simply excluded from the weighted mean (graceful behaviour).

The elevation layers of the CemaNeige nodes are matched by index (layer `i` of a node with layer `i` of the other nodes) and a node with fewer layers does not contribute to the upper layer indices. The elevation of a given layer index slightly differs from one node to the other: this approximation is assumed. `SnowPack` is a simulated state and `Psol` an output of the snow module: both are adjusted as well, for the consistency of the whole-catchment display of the snow and temperature series.

The network is read from the `griwrm` argument, which defaults to the "GRiwrm" attribute set on the *GRiwrmOutputsModel* object by RunModel.GRiwrmInputsModel. When `OutputsModel` is a plain list without this attribute, the default resolves to `NULL` and `griwrm` must be passed explicitly.

Value

A copy of `OM_id` with the meteorological data at the whole-catchment scale.

See Also

`plot.GRiwrmOutputsModel()`, `CreateGRiwrm()`

Examples

```
############################################################
# Meteorological data at the catchment scale of a CemaNeige   #
# network: the network used here is a chain of 2 nodes built  #
# on the airGR example data set (L0123001)                    #
############################################################
data(L0123001, package = "airGR")

# Network: Upstream (360 km2) -> Downstream (720 km2, i.e. a local
# sub-basin of 360 km2)
griwrm <- CreateGRiwrm(data.frame(
  id = c("Upstream", "Downstream"),
  down = c("Downstream", NA),
  length = c(0, NA),
  area = c(360, 720),
  model = rep("RunModel_CemaNeigeGR4J", 2)
))

# The 2 nodes share the meteorological data of the example data set
```

```
InputsPrecip <- matrix(BasinObs$P, ncol = 2, nrow = length(BasinObs$P))
InputsPotEvap <- matrix(BasinObs$E, ncol = 2, nrow = length(BasinObs$E))
InputsTemp <- matrix(BasinObs$T, ncol = 2, nrow = length(BasinObs$T))
InputsHypso <- matrix(rep(BasinInfo$HypsoData, 2), ncol = 2)
InputsZ <- rep(BasinInfo$HypsoData[51], 2)
colnames(InputsPrecip) <- colnames(InputsPotEvap) <- colnames(InputsTemp) <-
  colnames(InputsHypso) <- names(InputsZ) <- griwrm$id

InputsModel <- CreateInputsModel(
  griwrm,
  DatesR = BasinObs$DatesR,
  Precip = InputsPrecip,
  PotEvap = InputsPotEvap,
  TempMean = InputsTemp,
  ZInputs = InputsZ,
  HypsoData = InputsHypso
)

# Short run period (1 month) for a fast example. `Precip` and `PotEvap` are
# requested in `Outputs_Sim` because they are not part of the default
# outputs of a CemaNeige node
RunOptions <- CreateRunOptions(
  InputsModel,
  IndPeriod_Run = seq(1, 30),
  Outputs_Sim = c("DatesR", "Qsim", "CemaNeigeLayers", "Precip", "PotEvap")
)

# `Downstream` receives the simulated flow of `Upstream`: its parameter
# vector starts with the celerity of the routing model (airGR::RunModel_Lag)
ParamCN <- c(
  X1 = 408.774, X2 = 2.646, X3 = 131.264, X4 = 1.174,
  CNX1 = 0.962, CNX2 = 2.249
)
Param <- list(Upstream = ParamCN, Downstream = c(1, ParamCN))
OutputsModels <- RunModel(InputsModel, RunOptions = RunOptions, Param = Param)

# Area-weighted mean of the meteorological data over the whole catchment
# of the node `Downstream` (the local sub-basin areas are 360 km2 each, so
# the mean is the arithmetic mean of the 2 nodes here)
OutputsDownstream <- adjustMeteoToCatchment(
  OutputsModels[["Downstream"]],
  "Downstream",
  OutputsModels
)

# Precipitation at the whole-catchment scale and per-layer temperature
head(OutputsDownstream$Precip)
head(OutputsDownstream$CemaNeigeLayers[[1]]$Temp)
```

`as.Qm3s` *Coerce data.frame or content of a data.frame into a* Qm3s *object ready for plotting*

Description

Coerce data.frame or content of a data.frame into a *Qm3s* object ready for plotting

Usage

```
as.Qm3s(...)
```

Arguments

`...`	A data.frame for a single argument, or the arguments of function data.frame

Value

A data.frame of class *Qm3s*

`Calibration.GRiwrmInputsModel` *Calibration of the parameters of one catchment or a network of sub-catchments*

Description

Calibration algorithm that optimizes the error criterion selected as objective function using the provided functions.

Usage

```
## S3 method for class 'GRiwrmInputsModel'
Calibration(
  InputsModel,
  RunOptions,
  InputsCrit,
  CalibOptions,
  useUpstreamQsim = TRUE,
  forceReservoirObs = TRUE,
  ...
)

## S3 method for class 'InputsModel'
Calibration(InputsModel, RunOptions, CalibOptions, ...)

Calibration(InputsModel, ...)
```

Arguments

`InputsModel`	[object of class *InputsModel* or *GRiwrmInputsModel*] see CreateInputsModel
`RunOptions`	[object of class *RunOptions* or *GRiwrmRunOptions*] see CreateRunOptions
`InputsCrit`	[object of class *InputsCrit* or *GRiwrmInputsCrit*] see CreateInputsCrit
`CalibOptions`	[object of class *CalibOptions* or *GRiwrmCalibOptions*] see CreateCalibOptions for details
`useUpstreamQsim`	boolean describing if simulated (`TRUE`) or observed (`FALSE`) flows are used for calibration. Default is `TRUE`
`forceReservoirObs`	boolean indicating if reservoir observed flows must be forced during the model run. Default is `TRUE` (See details of RunModel_Reservoir)
`...`	further arguments passed to airGR::Calibration, see details

Details

This function can be used either for a catchment (with an *InputsModel* object), for a network (with a *GRiwrmInputsModel* object), or for an ungauged node cluster (with a *Ungauged* object).

Argument classes should be consistent to the usage:

- a `InputsModel` argument of class *InputsModel* must be followed by a `RunOptions` argument of class *RunOptions*, an `InputsCrit` argument of class *InputsCrit* and a `CalibOptions` of class *CalibOptions*
- an `InputsModel` argument of class *GRiwrmInputsModel* must be followed by a `RunOptions` argument of class *GRiwrmRunOptions*, an `InputsCrit` argument of class *GRiwrmInputsCrit* and a `CalibOptions` of class *GRiwrmCalibOptions*

Value

Depending on the class of `InputsModel` argument (respectively `InputsModel` or `GRiwrmInputsModel` object), the returned value is respectively:

- an `OutputsCalib` object (See airGR::Calibration for more details on this object)
- a `GRiwrmOutputsCalib` object which is a list of `OutputsCalib` objects with one item per modeled sub-catchment

See Also

`CreateGRiwrm()`, `CreateInputsModel.GRiwrm()`, `CreateInputsCrit()`, `CreateCalibOptions()`

Examples

```
# Loading catchment data
data(Severn)

##########################################################################
# EXAMPLE 1 - Basic calibration workflow                                 #
##########################################################################
```

```
# Creating catchment network
nodes <- Severn$BasinsInfo[, c("gauge_id", "downstream_id",
                               "distance_downstream", "area")]
nodes$model <- "RunModel_GR4J"
rename_columns <- list(id = "gauge_id",
                       down = "downstream_id",
                       length = "distance_downstream")
griwrm <- CreateGRiwrm(nodes, rename_columns)
griwrm

# Preparation of InputsModel object
BasinsObs <- Severn$BasinsObs
DatesR <- BasinsObs[[1]]$DatesR
PrecipTot <- cbind(sapply(BasinsObs, function(x) {x$precipitation}))
PotEvapTot <- cbind(sapply(BasinsObs, function(x) {x$peti}))
Qobs <- cbind(sapply(BasinsObs, function(x) {x$discharge_spec}))
Precip <- ConvertMeteoSD(griwrm, PrecipTot)
PotEvap <- ConvertMeteoSD(griwrm, PotEvapTot)
InputsModel <- CreateInputsModel(griwrm, DatesR, Precip, PotEvap)

# Calibration period selection
# Set aside warm-up period and use the rest for calibration
IndPeriod_Run <- seq(
  which(InputsModel[[1]]$DatesR ==
        (InputsModel[[1]]$DatesR[1] + 365 * 24 * 60 * 60)),
  length(InputsModel[[1]]$DatesR)
)
IndPeriod_WarmUp <- seq(1, IndPeriod_Run[1] - 1)

# Preparation of RunOptions object
RunOptions <- CreateRunOptions(
  InputsModel,
  IndPeriod_WarmUp = IndPeriod_WarmUp,
  IndPeriod_Run = IndPeriod_Run
)

# Calibration criterion: preparation of the InputsCrit object
InputsCrit <- CreateInputsCrit(
  InputsModel = InputsModel,
  FUN_CRIT = ErrorCrit_KGE2,
  RunOptions = RunOptions,
  Obs = Qobs[IndPeriod_Run, ],
  transfo = "sqrt"
)

# Preparation of CalibOptions object
CalibOptions <- CreateCalibOptions(InputsModel)

if (interactive()) { # Too long for CRAN check...
  # Calibration
  OutputsCalib <- suppressWarnings(
    Calibration(InputsModel, RunOptions, InputsCrit, CalibOptions)
```

```
  )

  # Simulation
  OutputsModels <- RunModel(
    InputsModel,
    RunOptions = RunOptions,
    Param = extractParam(OutputsCalib)
  )
}
```

`ConvertMeteoSD` *Conversion of meteorological data from basin scale to sub-basin scale*

Description

Conversion of meteorological data from basin scale to sub-basin scale

Usage

```
ConvertMeteoSD(x, ...)

## S3 method for class 'GRiwrm'
ConvertMeteoSD(x, meteo, ...)

## S3 method for class 'character'
ConvertMeteoSD(x, griwrm, meteo, ...)

## S3 method for class 'matrix'
ConvertMeteoSD(x, areas, temperature = FALSE, ...)
```

Arguments

`x`	either a `GRiwrm` network description (See CreateGRiwrm), a character id of a node, or a matrix containing meteorological data
`...`	Parameters passed to the methods
`meteo`	matrix or data.frame containing meteorological data. Its colnames should be equal to the ID of the basins
`griwrm`	`GRiwrm` object describing the semi-distributed network (See CreateGRiwrm)
`areas`	numeric vector with the total area of the basin followed by the areas of the upstream basins in km2
`temperature`	logical `TRUE` if the meteorological data contain air temperature. If `FALSE` minimum output values are bounded to zero

Value

matrix a matrix containing the converted meteorological data

See Also

CreateGRiwrm(), CreateInputsModel.GRiwrm()

`CreateCalibOptions.GRiwrmInputsModel`
Creation of the CalibOptions object

Description

This function can be used either for a single catchment (with an *InputsModel* object) or for a network of catchments (with a *GRiwrmInputsModel* object)

Usage

```
## S3 method for class 'GRiwrmInputsModel'
CreateCalibOptions(x, FixedParam = NULL, ...)

CreateCalibOptions(x, FixedParam = NULL, ...)

## S3 method for class 'InputsModel'
CreateCalibOptions(x, FixedParam = NULL, ...)

## S3 method for class 'character'
CreateCalibOptions(x, FixedParam = NULL, ...)

## S3 method for class '`function`'
CreateCalibOptions(x, FixedParam = NULL, ...)

## S3 method for class 'RunModel_Reservoir'
CreateCalibOptions(x, FixedParam = NULL, ...)
```

Arguments

`x`	For a single catchment, it can be an object of class *InputsModel* or a function or a character corresponding to `FUN_MOD` (compliant with **airGR** call). For a network, it should be an object of class *GRiwrmInputsModel*. See CreateInputsModel for details
`FixedParam`	(optional) numeric vector or list of vectors giving the values set for the non-optimized parameter values, as for airGR::CreateCalibOptions (see details)
`...`	arguments passed to airGR::CreateCalibOptions, see details

Details

See airGR::CreateCalibOptions documentation for a complete list of arguments.

With a *GRiwrmInputsModel* object, all arguments are applied on each sub-catchment of the network with some adaptation depending on the model used on each node.

If the argument FixedParam is a numeric vector, it is applied to each node of the network. Beware that parameters must be adapted depending on the use of the routing model and of the CemaNeige model on each node. If FixedParam is a list of numeric vectors, each item of the list will be applied on corresponding nodes. Use the id "*" for applying a setting on the remaining nodes. Example for applying one setting for all the nodes except the id "54057":

```
FixedParam <- list(`*` = c(NA, NA, NA, NA, NA, 0.25, NA, 10, NA),
                   `54057` = c(0.5, NA, NA, NA, NA, 0.25, NA, 10, NA))
```

The argument IsHyst is ignored since it should be defined previously with CreateInputsModel.GRiwrm.

Beware, if you transfer a parameter set from an upstream catchment to a downstream catchment, you must prescribe the celerity parameter (see example).

See transferGRparams for more details about how model parameters are transferred to ungauged nodes.

Value

Depending on the class of the InputsModel argument (respectively InputsModel or GRiwrmInputsModel object), the returned value is respectively:

- a CalibOptions object (See airGR::CreateCalibOptions)
- a GRiwrmCalibOptions object which is a list of CalibOptions objects with one item per modeled sub-catchment

See Also

CreateGRiwrm(), CreateInputsModel.GRiwrm(), CreateRunOptions(), CreateInputsCrit(), Calibration()

Examples

```
# Loading catchment data
data(Severn)

################################################################################
# EXAMPLE 1 - The one where all nodes are calibrated                           #
################################################################################

# Creating catchment network
nodes <- Severn$BasinsInfo[, c("gauge_id", "downstream_id",
                               "distance_downstream", "area")]
nodes$model <- "RunModel_GR4J"
rename_columns <- list(id = "gauge_id",
                       down = "downstream_id",
                       length = "distance_downstream")
griwrm <- CreateGRiwrm(nodes, rename_columns)
griwrm

# Preparation of InputsModel object
BasinsObs <- Severn$BasinsObs
```

```
DatesR <- BasinsObs[[1]]$DatesR
PrecipTot <- cbind(sapply(BasinsObs, function(x) {x$precipitation}))
PotEvapTot <- cbind(sapply(BasinsObs, function(x) {x$peti}))
Qobs <- cbind(sapply(BasinsObs, function(x) {x$discharge_spec}))
Precip <- ConvertMeteoSD(griwrm, PrecipTot)
PotEvap <- ConvertMeteoSD(griwrm, PotEvapTot)
InputsModel <- CreateInputsModel(griwrm, DatesR, Precip, PotEvap)

# Calibration period selection
# Set aside warm-up period and use the rest for calibration
IndPeriod_Run <- seq(
  which(InputsModel[[1]]$DatesR ==
        (InputsModel[[1]]$DatesR[1] + 365 * 24 * 60 * 60)),
  length(InputsModel[[1]]$DatesR)
)
IndPeriod_WarmUp <- seq(1, IndPeriod_Run[1] - 1)

# Preparation of RunOptions object
RunOptions <- CreateRunOptions(
  InputsModel,
  IndPeriod_WarmUp = IndPeriod_WarmUp,
  IndPeriod_Run = IndPeriod_Run
)

# Calibration criterion: preparation of the InputsCrit object
InputsCrit <- CreateInputsCrit(
  InputsModel = InputsModel,
  FUN_CRIT = ErrorCrit_KGE2,
  RunOptions = RunOptions,
  Obs = Qobs[IndPeriod_Run, ],
  transfo = "sqrt"
)

# Preparation of CalibOptions object
CalibOptions <- CreateCalibOptions(InputsModel)

# Calibration
if (interactive()) { # Too long for CRAN check...
  OutputsCalib <- suppressWarnings(
    Calibration(InputsModel, RunOptions, InputsCrit, CalibOptions)
  )

  # Simulation
  OutputsModels <- RunModel(
    InputsModel,
    RunOptions = RunOptions,
    Param = extractParam(OutputsCalib)
  )
}

################################################################################
# EXAMPLE 2 - The one where the node "54032" is ungauged                       #
################################################################################
```

```
# Creating catchment network with ungauged node
nodes$model[nodes$gauge_id == "54032"] <- "Ungauged"
griwrm <- CreateGRiwrm(nodes, rename_columns)

# Preparation of InputsModel object
InputsModel <- CreateInputsModel(griwrm, DatesR, Precip, PotEvap)

# Preparation of RunOptions object
RunOptions <- CreateRunOptions(
  InputsModel,
  IndPeriod_WarmUp = IndPeriod_WarmUp,
  IndPeriod_Run = IndPeriod_Run
)

# Calibration criterion: preparation of the InputsCrit object
InputsCrit <- CreateInputsCrit(
  InputsModel = InputsModel,
  FUN_CRIT = ErrorCrit_KGE2,
  RunOptions = RunOptions,
  Obs = Qobs[IndPeriod_Run, ],
  transfo = "sqrt"
)

# Preparation of CalibOptions object
CalibOptions <- CreateCalibOptions(InputsModel)

# Calibration
if (interactive()) { # Too long for CRAN check...
  OutputsCalib <- suppressWarnings(
    Calibration(InputsModel, RunOptions, InputsCrit, CalibOptions)
  )

  # Simulation
  OutputsModels <- RunModel(
    InputsModel,
    RunOptions = RunOptions,
    Param = extractParam(OutputsCalib)
  )
}

################################################################################
# EXAMPLE 3 - Parameter transfer from donor catchment "54001"                  #
# In this case, we want the "54032" node to be ungauged, and the "54001"       #
# node to give its parameters                                                  #
################################################################################

# Creating catchment network with parameter transfer
nodes$model[nodes$gauge_id == "54032"] <- "Ungauged"
nodes$donor <- as.character(NA)
nodes$donor[nodes$gauge_id == "54032"] <- "54001"
griwrm <- CreateGRiwrm(nodes, rename_columns)
```

```
# Preparation of InputsModel object
InputsModel <- CreateInputsModel(griwrm, DatesR, Precip, PotEvap)

# Preparation of RunOptions object
RunOptions <- CreateRunOptions(
  InputsModel,
  IndPeriod_WarmUp = IndPeriod_WarmUp,
  IndPeriod_Run = IndPeriod_Run
)

# Calibration criterion: preparation of the InputsCrit object
InputsCrit <- CreateInputsCrit(
  InputsModel = InputsModel,
  FUN_CRIT = ErrorCrit_KGE2,
  RunOptions = RunOptions,
  Obs = Qobs[IndPeriod_Run, ],
  transfo = "sqrt"
)

# Preparation of CalibOptions object
CalibOptions <- CreateCalibOptions(InputsModel)

if (interactive()) { # Too long for CRAN check...
  # Calibration
  OutputsCalib <- suppressWarnings(
    Calibration(InputsModel, RunOptions, InputsCrit, CalibOptions)
  )

  # Simulation
  OutputsModels <- RunModel(
    InputsModel,
    RunOptions = RunOptions,
    Param = extractParam(OutputsCalib)
  )
}

################################################################################
# EXAMPLE 4 - Parameter transfer with prescribed celerity parameter          #
# In this case, we want the "54032" node to be ungauged, and the "54029"      #
# node to give its parameters. Beware, the "54029" node has no celerity      #
# parameters, as it is an upstream catchment, but "54032" needs one          #
################################################################################

# Creating catchment network with parameter transfer and celerity
nodes$model[nodes$gauge_id == "54032"] <- "Ungauged"
nodes$donor <- as.character(NA)
nodes$donor[nodes$gauge_id == "54032"] <- "54029"
griwrm <- CreateGRiwrm(nodes, rename_columns)

# Preparation of InputsModel object
InputsModel <- CreateInputsModel(griwrm, DatesR, Precip, PotEvap)

# Preparation of RunOptions object
```

```
RunOptions <- CreateRunOptions(
  InputsModel,
  IndPeriod_WarmUp = IndPeriod_WarmUp,
  IndPeriod_Run = IndPeriod_Run
)

# Calibration criterion: preparation of the InputsCrit object
InputsCrit <- CreateInputsCrit(
  InputsModel = InputsModel,
  FUN_CRIT = ErrorCrit_KGE2,
  RunOptions = RunOptions,
  Obs = Qobs[IndPeriod_Run, ],
  transfo = "sqrt"
)

# Preparation of CalibOptions object with prescribed celerity
# Here you need to prescribe the celerity parameter for node "54032"
CalibOptions <- CreateCalibOptions(
  InputsModel,
  FixedParam = list("54032" = c(1, NA, NA, NA, NA))
)

if (interactive()) { # Too long for CRAN check...
  # Calibration
  OutputsCalib <- suppressWarnings(
    Calibration(InputsModel, RunOptions, InputsCrit, CalibOptions)
  )

  # Simulation
  OutputsModels <- RunModel(
    InputsModel,
    RunOptions = RunOptions,
    Param = extractParam(OutputsCalib)
  )
}
```

`CreateController` *Creation and adding of a controller in a supervisor*

Description

Creation and adding of a controller in a supervisor

Usage

```
CreateController(supervisor, ctrl.id, Y, U, FUN)
```

Arguments

supervisor	Supervisor object, see CreateSupervisor
ctrl.id	character id of the controller (see Details)
Y	character location of the controlled and/or measured variables in the model.
U	character location of the command variables in the model.
FUN	function controller logic which calculates U from Y (see Details)

Details

The ctrl.id is a unique id for finding the controller in the supervisor. If a controller with the same id already exists, it is overwritten by this new one.

FUN should be a function with one numeric parameter. This parameter will receive the measured values of at Y locations as input for the previous time step and returns calculated U. These U will then be applied at their location for the current time step of calculation of the model.

See RunModel.Supervisor and vignettes for examples of use.

Value

a Controller object which is a list with the following items:

- id character: the controller identifier
- U matrix: the list of controls for command variables with each column being the location of the variables and the rows being the values of the variable for the current time steps (empty by default)
- Unames character: location of the command variables
- Y matrix: the lists of controls for controlled variables with each column being the location of the variables and the rows being the values of the variable for the current time steps (empty by default)
- Ynames character: location of the controlled variables
- FUN function: controller logic which calculates U from Y

See Also

RunModel.Supervisor(), CreateSupervisor()

CreateGRiwrm *Generation of a network description containing all hydraulic nodes and the description of their connections*

Description

Generation of a network description containing all hydraulic nodes and the description of their connections

Usage

```
CreateGRiwrm(
  db,
  cols = list(id = "id", down = "down", length = "length", area = "area", model =
    "model", donor = "donor"),
  keep_all = FALSE
)
```

Arguments

db	data.frame description of the network (See details)
cols	list or vector columns of db. By default, mandatory column names are: id, down, length, area and model. Other names can be handled with a named list or vector containing items defined as "required name" = "column name in db" (See details)
keep_all	logical indicating if all columns of db should be kept or if only columns defined in cols should be kept

Details

db is a data.frame which at least contains in its columns:

- a node identifier (column id),
- the identifier and the hydraulic distance to the downstream node (character columns down and numeric columns length in km). The last downstream node should have fields down and length set to NA,
- the total area of the basin at the node location, including upstream basin areas (numeric column area in km2). Direct injection node can have a null area defined by NA
- the model to use (character column model), see section below for details

An optional column donor can be used to manually define which sub-basin will give its parameters to an ungauged node (See Ungauged model below).

Available models in airGRiwrm:

The "model" column should be filled by one of the following:

- One of the hydrological models available in the **airGR** package defined by its RunModel function (i.e.: RunModel_GR4J, RunModel_GR5HCemaneige...)
- RunModel_Reservoir for simulating a reservoir (See: RunModel_Reservoir)
- Ungauged for an ungauged node. The sub-basin inherits hydrological model and parameters from a "donor" sub-basin. If not defined by the user in the column donor, the donor is automatically set to the first gauged node at downstream. This set of sub-basins with the same donor downstream then forms an ungauged node cluster that will be calibrated at once.
- NA for injecting (or abstracting) a flow time series at the location of the node (direct flow injection)
- Diversion for abstracting a flow time series from an existing node transfer it to another node. As a Diversion is attached to an existing node, this node is then described with 2 lines: one for the hydrological model and another one for the diversion

Value

data.frame of class GRiwrm describing the airGR semi-distributed model network, with each line corresponding to a location on the river network and with the following columns:

- id (character): node identifier
- down (character): identifier of the node downstream of the current node (NA for the most downstream node)
- length (numeric): hydraulic distance to the downstream node in km (NA for the most downstream node)
- area (numeric): total area of the basin starting from the current node location (including upstream basin areas) in km2
- model (character): hydrological model to use (NA for using observed flow instead of a runoff model output)
- donor (character): node used as model and calibration parameter "donor" for ungauged nodes. For other types of nodes, if the donor is different than the id, it indicates that the node is embedded in an ungauged node cluster.

Examples

```
library(airGRiwrm)

########################################
# Network of 2 nodes distant of 150 km: #
########################################
# - an upstream reservoir modeled as a direct flow injection (no model)
# - a gauging station downstream a catchment of 360 km² modeled with GR4J
db <- data.frame(
  id = c("Reservoir", "GaugingDown"),
  length = c(150, NA),
  down = c("GaugingDown", NA),
  area = c(NA, 360),
  model = c(NA, "RunModel_GR4J"),
  stringsAsFactors = FALSE
)
griwrm_basic <- CreateGRiwrm(db)
griwrm_basic
# Network diagram with direct flow node in red, intermediate sub-basin in green
if (interactive()) {
  plot(griwrm_basic)
}

################################################
# GR4J semi-distributed model of the Severn River #
################################################
data(Severn)
nodes <- Severn$BasinsInfo
nodes$model <- "RunModel_GR4J"
str(nodes)
# Mismatch column names are renamed to stick with GRiwrm requirements
rename_columns <- list(
```

```
  id = "gauge_id",
  down = "downstream_id",
  length = "distance_downstream"
)
griwrm_severn <- CreateGRiwrm(nodes, rename_columns)
griwrm_severn
# Network diagram with upstream basin nodes in blue, intermediate sub-basin in green
if (interactive()) {
  plot(griwrm_severn)
}

#####################################################################
# Severn network with an ungauged station at nodes 54029 and 54001 #
#####################################################################
nodes_ungauged <- nodes
nodes_ungauged$model[
  nodes_ungauged$gauge_id %in% c("54029", "54001")
] <- "Ungauged"
# By default the first gauged node at downstream is used for parameter calibration (54032)
# Add a `donor`column for defining manually an upstream or sibling donor
nodes_ungauged$donor <- as.character(NA)
nodes_ungauged$donor[nodes_ungauged$id == "54001"] <- "54095"
griwrm_ungauged <- CreateGRiwrm(nodes_ungauged, rename_columns)
griwrm_ungauged
# Network diagram with gauged nodes of vivid color, and ungauged nodes of dull color
if (interactive()) {
  plot(griwrm_ungauged)
}

#############################################################
# Severn network with a Diversion on the node "54029"     #
# to a reservoir which transfer flows to the node "54001" #
# and a withdrawal on the reservoir                       #
#############################################################
nodes_div <- nodes[, c(
  "gauge_id",
  "downstream_id",
  "distance_downstream",
  "model",
  "area"
)]
nodes_div <- rbind(
  nodes_div,
  data.frame(
    gauge_id = c("54029", "Reservoir", "Irrigation_Pump"),
    downstream_id = c("Reservoir", "54001", "Reservoir"),
    distance_downstream = c(10, 5, 0),
    model = c("Diversion", "RunModel_Reservoir", NA),
    area = c(NA, NA, NA)
  )
)
griwrm_div <- CreateGRiwrm(nodes_div, rename_columns)
# Network diagram figures Diversion node by a red frame and a red arrow
```

```
if (interactive()) {
  plot(griwrm_div, orientation = "TB")
}

# It's also possible to custom the diagram's look with mermaid directives
# (See details in plot.GRiwrm help topic)
if (interactive()) {
  plot(
    griwrm_div,
   header = "%%{init: {'flowchart': {'nodeSpacing': 30, 'rankSpacing': 30, 'curve': 'linear'}}}%%"
  )
}
```

`CreateInputsCrit.GRiwrmInputsModel`
Creation of the InputsCrit object required to the `ErrorCrit` *functions*

Description

This function can be used either for a catchment (with an *InputsModel* object) or for a network (with a *GRiwrmInputsModel* object)

Usage

```
## S3 method for class 'GRiwrmInputsModel'
CreateInputsCrit(
  InputsModel,
  FUN_CRIT = ErrorCrit_KGE2,
  RunOptions,
  Obs,
  AprioriIds = getDefaultAprioriIds(InputsModel),
  k = 0.15,
  AprCelerity = 1,
  ...
)

## S3 method for class 'InputsModel'
CreateInputsCrit(InputsModel, FUN_CRIT, ...)

CreateInputsCrit(InputsModel, ...)
```

Arguments

`InputsModel`	object of class *InputsModel* or *GRiwrmInputsModel*. See CreateInputsModel
`FUN_CRIT`	[function (atomic or list)] error criterion function (e.g. airGR::ErrorCrit_RMSE, airGR::ErrorCrit_NSE)
`RunOptions`	object of class *RunOptions* or *GRiwrmRunOptions*, see CreateRunOptions

Obs	numeric, matrix or data.frame series of observed flows, see details
AprioriIds	(optional) named list or named vector whose entries are character vectors (ids of a priori sub-catchments) or numeric vectors (user-defined a priori parameters), used for the parameter regularization (see details)
k	(optional) numeric weight coefficient used in the parameter regularization (See airGR::CreateInputsCrit_Lavenne)
AprCelerity	(optional) numeric Default celerity used as a priori parameter for upstream catchments
...	arguments passed to airGR::CreateInputsCrit, see details

Details

See airGR::CreateInputsCrit documentation for a complete list of arguments.

Obs argument is equivalent to the same argument in airGR::CreateInputsCrit except that it must be a matrix or a data.frame if InputsModel is a *GRiwrmInputsModel* object. Then, each column of the matrix or data.frame represents the observations of one of the simulated node with the name of the columns representing the id of each node.

With a *GRiwrmInputsModel* object, all arguments are applied on each sub-catchments of the network.

Parameter regularization consists of defining a priori parameters which are used in a composed criterion based on the formula proposed by Lavenne et al. (2019) (See airGR::CreateInputsCrit_Lavenne). The parameter AprioriIds allows to define which neighbor sub-catchments are used for providing a priori parameters. Its format is as follows: `AprioriIds <- list("Downstream sub-catchment 1" = c("A priori upstream sub-catchment 1", ...))` where the quoted strings are the ids of the sub-catchments. The nodes providing a priori parameters must be calibrated before the current one. The sequence order of calibration can be checked with getNodeRanking. If the latter is not adequate, this order can be forced by setting the node providing a priori parameters as donor of the current node in CreateGRiwrm. See vignettes for more details. The parameter AprCelerity is a default value used as a priori for the parameter 'Celerity' in case of an upstream catchment (without celerity parameter) is used as a priori catchment. In the calibration process, all a priori parameter sets are tested and the one getting the best ErrorCrit score is used for the parameter regularization. By default, the immediate upstream catchments are used as a priori catchments, as determined by getDefaultAprioriIds.

The entries of AprioriIds can also be numeric vectors of user-defined a priori parameters. In that case, the given parameters are used as the a priori parameters of the De Lavenne criterion of the named sub-catchment instead of the parameters of an upstream sub-catchment: `AprioriIds <- list("Sub-catchment 1" = c("Upstream sub-catchment 1"), "Sub-catchment 2" = c(1, 50, -1, 3, 2))`. A user-defined a priori parameter must have one value per model parameter of the sub-catchment (in the same order as the Param argument of airGR::RunModel: e.g. `c(celerity, X1, X2, X3, X4)` for a GR4J sub-catchment with celerity). User-defined a priori parameters are used as-is: no default celerity is added and no X4 rescaling is applied.

Value

Depending on the class of InputsModel argument (respectively InputsModel and GRiwrmInputsModel object), the returned value is respectively:

- a `InputsCrit` object (See airGR::CreateInputsCrit)
- a `GRiwrmInputsCrit` object which is a list of `InputsCrit` objects with one item per modeled sub-catchment

References

De Lavenne, A., Andréassian, V., Thirel, G., Ramos, M.-H., Perrin, C., 2019. A Regularization Approach to Improve the Sequential Calibration of a Semidistributed Hydrological Model. Water Resources Research 55, 8821–8839. doi:10.1029/2018WR024266

See Also

`CreateGRiwrm()`, `CreateInputsModel.GRiwrm()`, `CreateRunOptions()`, `CreateCalibOptions()`, `Calibration()`

Examples

```
# Loading catchment data
data(Severn)

# Creating catchment network
nodes <- Severn$BasinsInfo[, c("gauge_id", "downstream_id",
                               "distance_downstream", "area")]
nodes$model <- "RunModel_GR4J"
rename_columns <- list(id = "gauge_id",
                       down = "downstream_id",
                       length = "distance_downstream")
griwrm <- CreateGRiwrm(nodes, rename_columns)

# Preparation of InputsModel object
BasinsObs <- Severn$BasinsObs
DatesR <- BasinsObs[[1]]$DatesR
PrecipTot <- cbind(sapply(BasinsObs, function(x) {x$precipitation}))
PotEvapTot <- cbind(sapply(BasinsObs, function(x) {x$peti}))
Qobs <- cbind(sapply(BasinsObs, function(x) {x$discharge_spec}))
Precip <- ConvertMeteoSD(griwrm, PrecipTot)
PotEvap <- ConvertMeteoSD(griwrm, PotEvapTot)
InputsModel <- CreateInputsModel(griwrm, DatesR, Precip, PotEvap)

# Calibration period selection
# Set aside warm-up period and use the rest for calibration
IndPeriod_Run <- seq(
  which(InputsModel[[1]]$DatesR ==
        (InputsModel[[1]]$DatesR[1] + 365 * 24 * 60 * 60)),
  length(InputsModel[[1]]$DatesR)
)
IndPeriod_WarmUp <- seq(1, IndPeriod_Run[1] - 1)

# Preparation of RunOptions object
RunOptions <- CreateRunOptions(
  InputsModel,
  IndPeriod_WarmUp = IndPeriod_WarmUp,
```

```
  IndPeriod_Run = IndPeriod_Run
)

# De Lavenne criterion with the default node-based a priori parameters
InputsCrit <- CreateInputsCrit(
  InputsModel = InputsModel,
  FUN_CRIT = ErrorCrit_KGE2,
  RunOptions = RunOptions,
  Obs = Qobs[IndPeriod_Run, ],
  transfo = "sqrt"
)

# De Lavenne criterion with a user-defined a priori parameter for node
# "54057" instead of the parameters of an upstream sub-catchment:
# one value per model parameter of the node (e.g. for a GR4J sub-catchment
# with celerity: celerity, X1, X2, X3, X4)
InputsCrit <- CreateInputsCrit(
  InputsModel = InputsModel,
  FUN_CRIT = ErrorCrit_KGE2,
  RunOptions = RunOptions,
  Obs = Qobs[IndPeriod_Run, ],
  AprioriIds = list(
    "54057" = c(1, 50, -1, 3, 2),
    "54032" = "54001",
    "54001" = "54095"
  ),
  transfo = "sqrt"
)
```

`CreateInputsModel` *Generic function for creating* `InputsModel` *object for either* **airGR** *or* **airGRiwrm**

Description

See the methods CreateInputsModel.GRiwrm for **airGRiwrm** and airGR::CreateInputsModel for **airGR**.

Usage

```
CreateInputsModel(x, ...)

## Default S3 method:
CreateInputsModel(x, ...)
```

Arguments

`x`	First parameter determining which InputsModel object is created
`...`	further arguments passed to or from other methods.

Value

InputsModel or GRiwrmInputsObject object

See Also

`CreateInputsModel.GRiwrm()`, `airGR::CreateInputsModel()`

`CreateInputsModel.GRiwrm` *Creation of an InputsModel object for an* **airGRiwrm** *network*

Description

Creation of an InputsModel object for an **airGRiwrm** network

Usage

```
## S3 method for class 'GRiwrm'
CreateInputsModel(
  x,
  DatesR,
  Precip = NULL,
  PotEvap = NULL,
  Qinf = NULL,
  Qobs = NULL,
  Qmin = NULL,
  Qrelease = NULL,
  PrecipScale = TRUE,
  TempMean = NULL,
  TempMin = NULL,
  TempMax = NULL,
  ZInputs = NULL,
  HypsoData = NULL,
  NLayers = 5L,
  IsHyst = FALSE,
  FUN_REGUL = NULL,
  ...
)
```

Arguments

`x`	GRiwrm object containing diagram of the semi-distributed model (See Create-GRiwrm)
`DatesR`	POSIXt vector of dates
`Precip`	(optional) matrix or data.frame of numeric containing precipitation in [mm per time step]. Column names correspond to node IDs

`PotEvap` (optional) matrix or data.frame of numeric containing potential evaporation [mm per time step]. Column names correspond to node IDs

`Qinf` (optional) matrix or data.frame of numeric containing observed flows. It must be provided only for nodes of type "Direct injection" and "Diversion". See CreateGRiwrm for details about these node types. Unit is [mm per time step] for nodes with an area, and [m³ per time step] for nodes with `area=NA`. Column names correspond to node IDs. Negative flows are abstracted from the model and positive flows are injected to the model

`Qobs` (deprecated) use `Qinf` instead

`Qmin` (optional) matrix or data.frame of numeric containing minimum flows that must be provided downstream of a node with a Diversion [m³ per time step]. Default is zero. Column names correspond to node IDs

`Qrelease` (optional) matrix or data.frame of numeric containing targeted release flows by nodes using the RunModel_Reservoir model [m³ per time step]. Column names correspond to node IDs

`PrecipScale` (optional) logical vector indicating if the mean of the precipitation interpolated on the elevation layers must be kept or not, required to create CemaNeige module inputs, default `TRUE` (the mean of the precipitation is kept to the original value). Optional names correspond to node IDs

`TempMean` (optional) matrix or data.frame of time series of mean air temperature [°C], required to create the CemaNeige module inputs. Column names correspond to node IDs

`TempMin` (optional) matrix or data.frame of time series of minimum air temperature [°C], possibly used to create the CemaNeige module inputs. Column names correspond to node IDs

`TempMax` (optional) matrix or data.frame of time series of maximum air temperature [°C], possibly used to create the CemaNeige module inputs. Column names correspond to node IDs

`ZInputs` (optional) numeric vector giving the mean elevation of the Precip and Temp series (before extrapolation) [m], possibly used to create the CemaNeige module input. Names correspond to node IDs

`HypsoData` (optional) matrix or data.frame containing 101 numeric rows: min, q01 to q99 and max of catchment elevation distribution [m], if not defined a single elevation is used for CemaNeige. Column names correspond to node IDs

`NLayers` (optional) numeric vector (integer) giving the number of elevation layers requested [-], required to create CemaNeige module inputs, default=5. Optional names correspond to node IDs

`IsHyst` logical indicating if the hysteresis version of CemaNeige is used. See details of `airGR::CreateRunOptions()`.

`FUN_REGUL` list of functions for local regulation (See details)

`...` used for compatibility with S3 methods

Details

Meteorological data are needed for the nodes of the network that represent a catchment simulated by a rainfall-runoff model. Instead of `airGR::CreateInputsModel()` that has numeric vector as time series inputs, this function uses matrix or data.frame with the id of the sub-catchments as column names. For single values (`ZInputs` or `NLayers`), the function requires named vector with the id of the sub-catchment as name item. If an argument is optional, only the column or the named item has to be provided.

See `airGR::CreateInputsModel()` documentation for details concerning each argument

The number of rows of `Precip`, `PotEvap`, `Qinf`, `Qmin`, `Qrelease`, `TempMean`, `TempMin`, `TempMax` must be the same as the length of `DatesR` (each row corresponds to a time step defined in `DatesR`).

`NA` values inside `Qrelease` (for specific time steps and nodes) can be used to run the corresponding reservoir in pass-through mode, i.e. without any water storage for those time steps.

For various examples of use see topics `RunModel.GRiwrmInputsModel()`, `RunModel_Reservoir()`, and `RunModel.Supervisor()`.

For example of use of Direct Injection nodes, see vignettes "V03_Open-loop_influenced_flow" and "V04_Closed-loop_regulated_withdrawal".

For example of use of Diversion nodes, see example in `RunModel.GRiwrmInputsModel()` topic and vignette "V06_Modelling_regulated_diversion".

The `FUN_REGUL` parameter:

`FUN_REGUL` argument is a named list of functions that modify the node `InputsModel` before sending it to the node's model. This feature is useful for modifying data such as `InputsModel$Qdiv` or `InputsModel$Qrelease` giving simulated flows already available from upstream nodes. Each item of the list has a name corresponding to the node on which the function is applied. Each function must follow this interface: `function(InputsModel, RunOptions, OutputsModel, env)` where the arguments are:

- `InputsModel`, the *InputsModel* object of the current node
- `RunOptions`, the *RunOptions* object of the current node
- `OutputsModel`, the *GRiwrmOutputsModel* object of the upstream and sibling nodes that have been already computed when the computation of the current node occurs
- `env`, the environment of the `RunModel.GRiwrmInputsModel()` function

The functions embedded in `FUN_REGUL` should all return the argument `InputsModel` after calculation.

Value

A *GRiwrmInputsModel* object which is a list of *InputsModel* objects created by `airGR::CreateInputsModel()` with one item per modeled sub-catchment.

See Also

`CreateGRiwrm()`, `CreateRunOptions()`, `RunModel.GRiwrmInputsModel()`

Examples

```
# Loading catchment data
data(Severn)

# Creating catchment network
nodes <- Severn$BasinsInfo[, c("gauge_id", "downstream_id", "distance_downstream", "area")]
nodes$model <- "RunModel_GR4J"
rename_columns <- list(id = "gauge_id",
                       down = "downstream_id",
                       length = "distance_downstream")
griwrm <- CreateGRiwrm(nodes, rename_columns)
griwrm

# Preparation of InputsModel object
BasinsObs <- Severn$BasinsObs
DatesR <- BasinsObs[[1]]$DatesR
PrecipTot <- cbind(sapply(BasinsObs, function(x) {x$precipitation}))
PotEvapTot <- cbind(sapply(BasinsObs, function(x) {x$peti}))
Qobs <- cbind(sapply(BasinsObs, function(x) {x$discharge_spec}))
Precip <- ConvertMeteoSD(griwrm, PrecipTot)
PotEvap <- ConvertMeteoSD(griwrm, PotEvapTot)
InputsModel <- CreateInputsModel(griwrm, DatesR, Precip, PotEvap)
```

CreateRunOptions.GRiwrmInputsModel

Creation of the RunOptions object

Description

This function can be used either for a catchment (with an *InputsModel* object) or for a network (with a *GRiwrmInputsModel* object)

Usage

```
## S3 method for class 'GRiwrmInputsModel'
CreateRunOptions(x, IniStates = NULL, Outputs_Sim = MinimalOutputsSim, ...)

CreateRunOptions(x, ...)

## S3 method for class 'InputsModel'
CreateRunOptions(x, ...)

## S3 method for class 'character'
CreateRunOptions(x, InputsModel, ...)

## S3 method for class '`function`'
CreateRunOptions(x, InputsModel, ...)
```

```
## S3 method for class 'RunModel_Reservoir'
CreateRunOptions(x, ...)
```

Arguments

x	For a single catchment, it can be an object of class *InputsModel* or a function or a character corresponding to FUN_MOD (compliant with **airGR** call). For a network, it should be an object of class *GRiwrmInputsModel*. See CreateInputsModel for details
IniStates	(optional) object containing initial model states. Can be: • A single IniStates object when calling CreateRunOptions.InputsModel • A named list of IniStates objects when calling CreateRunOptions.GRiwrmInputsModel, with names matching node IDs See airGR::CreateIniStates for details
Outputs_Sim	(optional) outputs to be returned by the model run. Can be: • "all" to return all available simulated outputs for each node (previous default behaviour, but not recommended for large networks) • A character vector, applied to every node, to restrict the returned outputs (default: c("DatesR", "WarmUpQsim", "Qsim", "Qsim_m3", "QsimDown", "Param", "StateEnd")). • A named list (names matching node IDs) to define outputs per node, each item being a character vector. On a semi-distributed network, an explicit selection is completed with the minimal default output set so that the outputs required to propagate the simulated flows between nodes are available (#98). Outputs_Sim = "all" is not affected. See details. See airGR::CreateRunOptions for the list of output names supported by each rainfall-runoff model.
...	arguments passed to airGR::CreateRunOptions, see details
InputsModel	object of class *InputsModel* (only used to be consistent with the original airGR::CreateRunOptions which has FUN_MOD as first parameter) see airGR::CreateInputsModel for details

Details

The default output selection has been reduced compared to previous versions of the package (where it was "all") in order to keep the size of the resulting *GRiwrmOutputsModel* object reasonable for a network with a large number of nodes (See issue inrae/airGRiwrm#141). The default keeps only the outputs required to propagate simulated flows to downstream nodes (Qsim, Qsim_m3), to warm-up semi-distributed nodes correctly (WarmUpQsim), to carry the model parameters (Param) and the model states at the end of the run (StateEnd) which are needed for De Lavenne parameter regularization and hot restart. For CemaNeige models, the CemaNeigeLayers output is added automatically: it carries the CemaNeige layer output series required by SCA-based error criteria (a reduced CemaNeige output set would change the model run in airGR). Precip is also added automatically on CemaNeige models: the default plot of a CemaNeige node requires it in the *OutputsModel* to compute the precipitation panel axis as soon as the per-layer solid precipitation (Psol)

is present (upstream airGR `ylimP` limitation). It is a recorded meteo input, not a Fortran CemaNeige output: the CemaNeige `NOutputs` and the model run are unchanged. Use `Outputs_Sim = "all"` to restore the previous behaviour.

On a semi-distributed network (a network with at least one semi-distributed node, i.e. a node whose sub-catchment receives the routed flows of upstream nodes), an explicit `Outputs_Sim` selection is completed with the minimal default output set (the default selection, See above, #141). This is required because airGR computes the discharge of each semi-distributed node by routing the upstream flows and aggregating them with the local runoff: the node's own outputs are passed to airGR's `RunModel_Lag` as `QcontribDown`, which requires in particular `Qsim` and, when the warm-up period is non-zero, `WarmUpQsim`. Without this completion, an explicit selection like `Outputs_Sim = c("Qsim")` fails with the error "Time series WarmUpQsim in 'QcontribDown' should have the same length as 'RunOptions$IndPeriod_WarmUp'" (#98).

See airGR::CreateRunOptions documentation for a complete list of arguments.

If `x` argument is a *GRiwrmInputsModel* object, `IniStates` must be a list of numeric object of class *IniStates* with one item per modeled sub-catchment.

With a *GRiwrmInputsModel* object, all arguments are applied on each sub-catchments of the network.

For examples of use see topics RunModel.GRiwrmInputsModel, RunModel_Reservoir, and RunModel.Supervisor.

Value

Depending on the class of `InputsModel` argument (respectively *InputsModel* and *GRiwrmInputsModel* object), the returned value is respectively:

- a `RunOptions` object (See airGR::CreateRunOptions)
- a `GRiwrmRunOptions` object which is a list of `RunOptions` objects with one item per modeled sub-catchment

See Also

`CreateGRiwrm()`, `CreateInputsModel.GRiwrm()`, `RunModel.GRiwrmInputsModel()`

`CreateSupervisor` *Creation of a Supervisor for handling regulation in a model*

Description

Creation of a Supervisor for handling regulation in a model

Usage

```
CreateSupervisor(InputsModel, TimeStep = 1L)
```

Arguments

`InputsModel`	[object of type `GRiwrmInputsModel`] inputs of the model
`TimeStep`	integer number of time steps between each supervision

Details

See RunModel.Supervisor and vignettes for examples of use.

Value

A `Supervisor` object which is an environment containing all the necessary variables to run a supervised simulation, such as:

- `DatesR` base::POSIXct: vector of date from `InputsModel`
- `InputsModel`: a copy of `InputsModel` provided by CreateInputsModel.GRiwrm
- `griwrm`: a copy of `griwrm` provided by CreateGRiwrm
- `Controllers` list: list of the controllers used in the supervised simulation (See CreateController)
- `idx.output` integer: index of the current time step output in the modeled series (updated during simulation)
- `idx.output_previous` integer: index of the previous output time step
- `ts.date` base::POSIXct: date/time of the current time step for controller calculations
- `ts.index0` integer: index of the time step preceding the start of the simulation period
- `controller.id` character: identifier of the current controller being applied

See Also

`RunModel.Supervisor()`, `CreateController()`

`extractParam` *Extract calibrated parameters*

Description

Extract list of parameters from the output of Calibration.GRiwrmInputsModel which can be directly used as argument `Param` of RunModel.GRiwrmInputsModel, RunModel.Supervisor, and RunModel.GRiwrmOutputsModel.

Usage

```
extractParam(x)

## S3 method for class 'GRiwrmOutputsCalib'
extractParam(x)

## S3 method for class 'GRiwrmOutputsModel'
extractParam(x)
```

Arguments

x	Object of class GRiwrmOutputsModel returned by Calibration.GRiwrmInputsModel

Details

See vignettes and example of RunModel_Reservoir for examples of use.

Value

A list containing representing the calibrated parameters of each modeled node as a numeric vector.

See Also

Calibration, RunModel.GRiwrmInputsModel, RunModel.Supervisor

---

getDefaultAprioriIds *Get default AprioriIds from direct upstream nodes of each node*

Description

Get default AprioriIds from direct upstream nodes of each node

Usage

```
getDefaultAprioriIds(InputsModel)
```

Arguments

InputsModel	object of class *InputsModel* or *GRiwrmInputsModel*. See CreateInputsModel

Value

A list named with node Ids and containing the Ids of the upstream nodes that can be used for apriori parameters.

getNextTimeSteps *Get the next time steps date/time of a simulation*

Description

Get the next time steps date/time of a simulation

Usage

```
getNextTimeSteps(x, TimeStep = 1L)
```

Arguments

x	Object returned by RunModel.GRiwrmInputsModel, RunModel.Supervisor, or RunModel.GRiwrmOutputsModel
TimeStep	integer number of time steps to get after the end of the simulation

Value

A base::POSIXct vector containing the date/time of the time steps following the end of the simulation.

getNodeProperties *Properties of GRiwrm nodes*

Description

getNodeProperties returns properties of a single node, and

Usage

```
getNodeProperties(id, griwrm)

getAllNodesProperties(griwrm)
```

Arguments

id	character Id of the node in the GRiwrm object
griwrm	[GRiwrm object] describing the network of the semi-distributed model (See CreateGRiwrm)

Details

A "Gauged" node is either a node containing a model that is already calibrated (parameters are already fixed) or a node containing a model where observations are available for calibration.

A "Ungauged" node is a node containing a model which derives its parameters from another "donor" node.

Value

`getNodeProperties` returns a list with the following items:

- "position" (character): Position of the node in the network ("Upstream" or "Intermediate")
- "DirectInjection" (logical): is the node a Direct Injection node?
- "Diversion" (logical): is the node a Diversion node?
- "Reservoir" (logical): is the node a Reservoir?
- "airGR" (logical): is the node contains an airGR model?
- "calibration" (character): describe if the node is a "Gauged", or an "Ungauged" station, (see details), or "NA" otherwise
- "Upstream" (logical): is the node an upstream node?
- "RunOff" (logical): is the node contains an hydrological model?

`getAllNodesProperties` returns a data.frame constituted from the list returned by `getNodeProperties` for all nodes.

See Also

`CreateGRiwrm()`

Examples

```
################################################################################
# Severn network with :                                                        #
# - a Diversion on the node "54001" which transfer flows to the node "540029" #
# - node 54002 as a Direct Injection node                                      #
################################################################################
data(Severn)
nodes <- Severn$BasinsInfo
nodes$model <- "RunModel_GR4J"
str(nodes)
nodes <- nodes[, c(
  "gauge_id",
  "downstream_id",
  "distance_downstream",
  "model",
  "area"
)]
# Add a Diversion node from node "54001" to "54029"
nodes <- rbind(
  nodes,
  data.frame(
    gauge_id = "54001",
    downstream_id = "54029",
    distance_downstream = 20,
    model = "Diversion",
    area = NA
  )
)
```

```
# Set node '54002' as a Direct Injection node
nodes$model[nodes$id == "54002"] <- NA
# Mismatch column names are renamed to stick with GRiwrm requirements
rename_columns <- list(
  id = "gauge_id",
  down = "downstream_id",
  length = "distance_downstream"
)
# Create GRiwrm object and display properties
griwrm <- CreateGRiwrm(nodes, rename_columns)

str(getNodeProperties("54001", griwrm))

getAllNodesProperties(griwrm)
```

`getNodeRanking` *Sorting of the nodes from upstream to downstream for RunModel and Calibration*

Description

Sorting of the nodes from upstream to downstream for RunModel and Calibration

Usage

```
getNodeRanking(griwrm)
```

Arguments

`griwrm`	[object of class `GRiwrm`] see CreateGRiwrm for details

Details

The sort is done by searching upstream nodes in the networks recursively. Ungauged node clusters are processed by cluster and the algorithm tries to process ungauged nodes which receive their parameters from upstream or sibling node after their donor node. Use `options(debug = TRUE)` to get details on how the sort is performed.

Value

A character vector containing ordered node ids

See Also

`CreateGRiwrm()`

getSD_Ids *Function to get the IDs of sub-basins using SD model or not*

Description

Function to get the IDs of sub-basins using SD model or not

Usage

```
getSD_Ids(InputsModel, add_diversions = FALSE)

getNoSD_Ids(InputsModel, include_diversion = TRUE)
```

Arguments

InputsModel	GRiwrmInputsModel object
add_diversions	logical for adding upstream nodes due to diversion
include_diversion	logical for including diversion nodes

Value

character IDs of the sub-basins using SD model or not

isNodeDownstream *Check if a node is downstream or upstream another one*

Description

Check if a node is downstream or upstream another one

Usage

```
isNodeDownstream(x, current_node, candidate_node)

## S3 method for class 'GRiwrmInputsModel'
isNodeDownstream(x, current_node, candidate_node)

## S3 method for class 'GRiwrm'
isNodeDownstream(x, current_node, candidate_node)

isNodeUpstream(x, current_node, candidate_node)

## S3 method for class 'GRiwrm'
isNodeUpstream(x, current_node, candidate_node)
```

```
## S3 method for class 'GRiwrmInputsModel'
isNodeUpstream(x, current_node, candidate_node)
```

Arguments

x	GRiwrmInputsModel object (see CreateInputsModel.GRiwrm) or GRiwrm (See CreateGRiwrm)
current_node	character with the id of the current node
candidate_node	character with the id of the node for which we want to know if it is downstream or upstream current_node

Value

logical TRUE if the node with the id down_candidate is downstream or upstream the node with the id current_node

---

merge.OutputsModel *Merge Two outputs of airGR simulations*

Description

Merge Two outputs of airGR simulations

Usage

```
## S3 method for class 'OutputsModel'
merge(x, y, ...)

## S3 method for class 'GRiwrmOutputsModel'
merge(x, y, ...)
```

Arguments

x, y	**OutputsModel** objects from airGR::RunModel, RunModel.GRiwrmInputsModel, RunModel.GRiwrmOutputsModel, RunModel.Supervisor
...	Not used

Value

An object **OutputsModel** with merged times series of simulation results.

`mermaid` *Plot a mermaid diagram*

Description

These functions download the diagram from https://mermaid.ink which generates the image.

Usage

```
mermaid(
  diagram,
  format = "png",
  theme = "default",
  dir.dest = tempdir(),
  file.dest = paste0(rlang::hash(link), ".", format),
  server = Sys.getenv("MERMAID_URL", "https://mermaid.ink"),
  link = mermaid_gen_link(diagram, theme = theme, format = format, server = server)
)

mermaid_gen_link(
  diagram,
  theme = "default",
  format = "png",
  server = Sys.getenv("MERMAID_URL", "https://mermaid.ink")
)

## S3 method for class 'mermaid'
plot(x, add = FALSE, ...)
```

Arguments

`diagram`	Diagram in mermaid markdown-like language or file (as a connection or file name) containing a diagram specification
`format`	Image format (either `"jpg"`, or `"png"`, or `"svg"`)
`theme`	Mermaid theme (See available themes in Mermaid documentation)
`dir.dest`	Destination folder for the downloaded image. This parameter is ignored if `file.dest` contains a folder path.
`file.dest`	Path to the downloaded image. It's combined with `dir.dest` if it only contains the name of the file without a folder path.
`server`	URL of the server used to generate the link
`link`	Link generated by mermaid_gen_link
`x`	character mermaid diagram dialect
`add`	logical to add the diagram on the existing plot
`...`	Other argument passed to mermaid

Details

Compared to the `diagrammeR::mermaid` function, the generated image or plot is not a HTMLwidget and can be knit in pdf through latex and moreover, its size can be controlled with `fig.width` and `fig.height`.

If the generation failed (due to internet connection failure or syntax error in mermaid script), the functions raises no error (see `mermaid` returned value).

By default, `mermaid` uses the service https://mermaid.ink. One can define an alternative server to use with the environment variable `MERMAID_URL`.

Value

- `mermaid` returns the path to the downloaded image or `NA` if the download failed. In this latter case, get the error message in the attribute "error".
- `mermaid_gen_link` returns the link to the web service which generates the diagram
- `plot.mermaid` produces a R plot with the mermaid diagram

Nothing, used for side effect.

Examples

```
diagram <- "flowchart LR\n  A --> B"
mermaid_gen_link(diagram)
if (interactive()){
f <- mermaid(diagram)
f

# For displaying the diagram in Rmarkdown document
knitr::include_graphics(mermaid(diagram))

# Clean temporary folder
unlink(f)
}

if (interactive()){
s <- "flowchart LR
A --> B"
class(s) <- c("mermaid", class(s))
plot(s)
}
```

`plot.GRiwrm` *Plot of a diagram representing the network structure of a GRiwrm object*

Description

Plot a GRiwrm diagram using online mermaid diagram generator.

Usage

```
## S3 method for class 'GRiwrm'
plot(
  x,
  display = TRUE,
  orientation = "LR",
  with_donors = TRUE,
 box_colors = c(UpstreamUngauged = "#eef", UpstreamGauged = "#aaf", IntermediateUngauged
  = "#efe", IntermediateGauged = "#afa", Reservoir = "#9de", DirectInjection = "#faa"),
  defaultClassDef = "stroke:#333",
  header = "%%{init: {'theme': 'neutral'} }%%",
  footer = NULL,
  ...
)
```

Arguments

x	[GRiwrm object] data to display. See CreateGRiwrm for details
display	logical if TRUE plots the diagram, returns the mermaid code otherwise
orientation	character orientation of the graph. Possible values are "LR" (left-right), "RL" (right-left), "TB" (top-bottom), or "BT" (bottom-top).
with_donors	logical for drawing boxes around ungauged nodes and their donors
box_colors	list containing the color used for the different types of nodes
defaultClassDef	character default style apply to all boxes
header	mermaid script to add before the generated script (see Details)
footer	mermaid script to add after the generated script
...	further parameters passed to mermaid

Details

header parameter allows to add any mermaid code injected before the graph instruction. It is notably useful for injecting directives that impact the format of the graph. See mermaid documentation on directives for more details and also the complete list of available directives.

By default, this uses the service https://mermaid.ink. One can define an alternative server to use with the environment variable MERMAID_URL (See mermaid for more details).

Value

Mermaid code of the diagram if display is FALSE, otherwise the function returns the diagram itself.

See Also

CreateGRiwrm()

Examples

```
library(airGRiwrm)

########################################
# Network of 2 nodes distant of 150 km: #
########################################
# - an upstream reservoir modeled as a direct flow injection (no model)
# - a gauging station downstream a catchment of 360 km² modeled with GR4J
db <- data.frame(
  id = c("Reservoir", "GaugingDown"),
  length = c(150, NA),
  down = c("GaugingDown", NA),
  area = c(NA, 360),
  model = c(NA, "RunModel_GR4J"),
  stringsAsFactors = FALSE
)
griwrm_basic <- CreateGRiwrm(db)
griwrm_basic
# Network diagram with direct flow node in red, intermediate sub-basin in green
if (interactive()) {
  plot(griwrm_basic)
}

################################################
# GR4J semi-distributed model of the Severn River #
################################################
data(Severn)
nodes <- Severn$BasinsInfo
nodes$model <- "RunModel_GR4J"
str(nodes)
# Mismatch column names are renamed to stick with GRiwrm requirements
rename_columns <- list(
  id = "gauge_id",
  down = "downstream_id",
  length = "distance_downstream"
)
griwrm_severn <- CreateGRiwrm(nodes, rename_columns)
griwrm_severn
# Network diagram with upstream basin nodes in blue, intermediate sub-basin in green
if (interactive()) {
  plot(griwrm_severn)
}

#####################################################################
# Severn network with an ungauged station at nodes 54029 and 54001 #
#####################################################################
nodes_ungauged <- nodes
nodes_ungauged$model[
  nodes_ungauged$gauge_id %in% c("54029", "54001")
] <- "Ungauged"
# By default the first gauged node at downstream is used for parameter calibration (54032)
# Add a `donor`column for defining manually an upstream or sibling donor
```

```
nodes_ungauged$donor <- as.character(NA)
nodes_ungauged$donor[nodes_ungauged$id == "54001"] <- "54095"
griwrm_ungauged <- CreateGRiwrm(nodes_ungauged, rename_columns)
griwrm_ungauged
# Network diagram with gauged nodes of vivid color, and ungauged nodes of dull color
if (interactive()) {
  plot(griwrm_ungauged)
}

############################################################
# Severn network with a Diversion on the node "54029"     #
# to a reservoir which transfer flows to the node "54001" #
# and a withdrawal on the reservoir                       #
############################################################
nodes_div <- nodes[, c(
  "gauge_id",
  "downstream_id",
  "distance_downstream",
  "model",
  "area"
)]
nodes_div <- rbind(
  nodes_div,
  data.frame(
    gauge_id = c("54029", "Reservoir", "Irrigation_Pump"),
    downstream_id = c("Reservoir", "54001", "Reservoir"),
    distance_downstream = c(10, 5, 0),
    model = c("Diversion", "RunModel_Reservoir", NA),
    area = c(NA, NA, NA)
  )
)
griwrm_div <- CreateGRiwrm(nodes_div, rename_columns)
# Network diagram figures Diversion node by a red frame and a red arrow
if (interactive()) {
  plot(griwrm_div, orientation = "TB")
}

# It's also possible to custom the diagram's look with mermaid directives
# (See details in plot.GRiwrm help topic)
if (interactive()) {
  plot(
    griwrm_div,
   header = "%%{init: {'flowchart': {'nodeSpacing': 30, 'rankSpacing': 30, 'curve': 'linear'}}}%%"
  )
}
```

---

plot.GRiwrmOutputsModel

Function which creates screen plots giving an overview of the model outputs in the GRiwrm network

---

Description

Function which creates screen plots giving an overview of the model outputs in the GRiwrm network

Usage

```
## S3 method for class 'GRiwrmOutputsModel'
plot(x, Qobs = NULL, Vobs = NULL, unit = "m3/s", ...)
```

Arguments

`x`	[object of class *GRiwrmOutputsModel*] see RunModel.GRiwrmInputsModel for details
`Qobs`	(optional) matrix time series of observed flows (for the same time steps than simulated) (mm/time step) with one column by hydrological model output named with the node ID (See CreateGRiwrm for details)
`Vobs`	(optional) matrix time series of observed or targeted storage (for the same time steps than simulated) [m3] with one column by hydrological model output named with the node ID (See CreateGRiwrm for details)
`unit`	(optional) character flows unit ("m3/s" or "mm")
`...`	Further arguments for airGR::plot.OutputsModel and plot

Details

For examples of use see topics RunModel.GRiwrmInputsModel, RunModel_Reservoir, and RunModel.Supervisor.

The simulated discharge of a node integrates the flows of the whole catchment of the node, while the meteorological time series stored in the *OutputsModel* of the node (precipitation and potential evapotranspiration) come from the meteorological data of its local sub-basin only. Therefore, when these meteorological time series are available in the *GRiwrmOutputsModel* object (i.e. requested in `Outputs_Sim`, see CreateRunOptions), they are replaced on the fly, without modifying the object, by their area-weighted mean over the whole catchment of the node (the node and all its upstream nodes), so that the displayed meteorology reflects the whole catchment (See issue inrae/airGRiwrm#178).

For a CemaNeige node, the same adjustment is applied to the per-layer series of `CemaNeigeLayers` displayed by airGR::plot.OutputsModel: the mean air temperature (`Temp`), the solid precipitation (`Psol`) and the snow pack state (`SnowPack`) of each elevation layer are replaced by their area-weighted mean over the whole catchment of the node, as well as the mean over the layers that airGR derives from them. The other displayed series are not adjusted because they are either already at the scale of the whole catchment (the simulated discharge `Qsim`) or simulated at this scale (the actual evapotranspiration `AE`).

When the ID of a node is shared with a Diversion node (two rows of the network with the same `id`, see CreateGRiwrm), the local sub-basin area of the shared ID cannot be computed and the node is simply excluded from the weighted mean (graceful behaviour).

Value

list of plots.

See Also

`adjustMeteoToCatchment()` to compute statistics from the meteorological data at the whole-catchment scale.

`plot.OutputsModelReservoir`

Plot simulated reservoir volume, inflows and released flows time series on a reservoir node

---

Description

Plot simulated reservoir volume, inflows and released flows time series on a reservoir node

Usage

```
## S3 method for class 'OutputsModelReservoir'
plot(x, Qobs = NULL, Vobs = NULL, which = NULL, ...)
```

Arguments

`x`	Object returned by RunModel_Reservoir
`Qobs`	(optional) numeric time series of targeted released flow [m3/time step]
`Vobs`	(optional) numeric time series of observed or targeted volume curve [m3]
`which`	Not used (for compatibility with airGR::plot.OutputsModel)
`...`	Further arguments passed to plot.Qm3s

Value

Function used for side effect.

Examples

```
#######################################################
# Daily time step simulation of a reservoir filled by #
# one catchment supplying a constant released flow    #
#######################################################

library(airGRiwrm)
data(L0123001)

# Inflows comes from a catchment of 360 km² modeled with GR4J
# The reservoir receives directly the inflows
db <- data.frame(
```

```
  id = c(BasinInfo$BasinCode, "Reservoir"),
  length = c(0, NA),
  down = c("Reservoir", NA),
  area = c(BasinInfo$BasinArea, NA),
  model = c("RunModel_GR4J", "RunModel_Reservoir"),
  stringsAsFactors = FALSE
)
griwrm <- CreateGRiwrm(db)
if (interactive()) {
  plot(griwrm)
}

# This catchment inflows a reservoir of maximum capacity Vmax
Vmax <- 4E6 # in m3

# Formatting of GR4J inputs for airGRiwrm (matrix or data.frame with one
# column by sub-basin and node IDs as column names)
Precip <- matrix(BasinObs$P, ncol = 1)
colnames(Precip) <- BasinInfo$BasinCode
PotEvap <- matrix(BasinObs$E, ncol = 1)
colnames(PotEvap) <- BasinInfo$BasinCode

# We propose to compute the constant released flow from
# the Q20 of the natural flow
# The value is in m3 by time step (day)
(Qrelease <- quantile(BasinObs$Qls, na.rm = TRUE, probs = 0.2) / 1000 * 86400)

# Formatting of reservoir released flow inputs for airGRiwrm (matrix or data.frame
# with one column by node and node IDs as column names)
Qrelease <- data.frame(Reservoir = rep(Qrelease, length(BasinObs$DatesR)))

InputsModel <- CreateInputsModel(
  griwrm,
  DatesR = BasinObs$DatesR,
  Precip = Precip,
  PotEvap = PotEvap,
  Qrelease = Qrelease
)

## run period selection
Ind_Run <- seq(
  which(format(BasinObs$DatesR, format = "%Y-%m-%d") == "1990-01-01"),
  which(format(BasinObs$DatesR, format = "%Y-%m-%d") == "1999-12-31")
)

# Creation of the GRiwrmRunOptions object with the initial states
# of the reservoir set to 0 (empty reservoir)
RunOptions <- CreateRunOptions(
  InputsModel,
  IndPeriod_Run = Ind_Run,
  IndPeriod_WarmUp = seq.int(Ind_Run[1] - 365, length.out = 365),
  IniStates = list(Reservoir = c("Reservoir.V" = 0))
)
```

```
# calibration criterion: preparation of the InputsCrit object
Qobs <- data.frame("L0123001" = BasinObs$Qmm[Ind_Run])
InputsCrit <- CreateInputsCrit(
  InputsModel,
  ErrorCrit_KGE2,
  RunOptions = RunOptions,
  Obs = Qobs
)

# preparation of CalibOptions object with fixed parameters for the reservoir
# The capacity of the reservoir is set to Vmax and the inflow celerity is 0.5 m/s.
CalibOptions <- CreateCalibOptions(
  InputsModel,
  FixedParam = list(Reservoir = c(Vmax = Vmax, celerity = 0.5))
)

OC <- Calibration(
  InputsModel = InputsModel,
  RunOptions = RunOptions,
  InputsCrit = InputsCrit,
  CalibOptions = CalibOptions
)

# Model parameters
Param <- extractParam(OC)
str(Param)

# Running simulation
OutputsModel <- RunModel(InputsModel, RunOptions, Param)

# Plot the simulated flows and volumes on all nodes
Qobs <- cbind(BasinObs$Qmm[Ind_Run], Qrelease[Ind_Run, ])
colnames(Qobs) <- griwrm$id
plot(OutputsModel, Qobs = Qobs)

# The plot for the reservoir can also be plotted alone
plot(OutputsModel$Reservoir, Qobs = Qobs[, "Reservoir"])

########################################################
# Daily time step simulation of a reservoir tracking  #
# an objective filling curve using a local regulation #
########################################################

# The objective here is to simulate the same reservoir as above
# but with new rules:
# - A minimum flow downstream the reservoir defined as:
(Qmin <- Qrelease[1, ])
# - A maximum release flow of 20 m3/s for flood mitigation
(Qmax <- 20 * 86400)
# - An annual objective filling curve managing floods and droughts by
# trying to keep the reservoir volume between 2 and 8 Mm3:
Vobj <- approx(c(1, 150, 300, 366), c(1E6, 3.5E6, 0.5E6, 1E6), seq(366))
```

```
plot(Vobj, type = "l", col = "red", lty = 2)

# The regulation function takes InputsModel of the reservoir node and the
# global GRiwrm OutputsModel as arguments and returns a modified
# InputsModel used by RunModel_Reservoir afterward
fun_factory_Regulation_Reservoir <- function(Vini, Vobj, Qmin, Qmax, Vmax) {
  function(InputsModel, RunOptions, OutputsModel, env) {
    # Release flow time series initialisation
    Qrelease <- rep(0, length(InputsModel$DatesR))
    # Build inflows time series from upstream Qsim (warmup & run)
    Qinflows <- Qrelease
    IPR_all <- c(RunOptions$IndPeriod_WarmUp, RunOptions$IndPeriod_Run)
    Qinflows[IPR_all] <- c(
      OutputsModel$L0123001$RunOptions$WarmUpQsim_m3,
      OutputsModel$L0123001$Qsim_m3
    )
    # Reservoir volume initialisation
    V <- Vini
    # Loop over simulation time steps (warmup & run periods)
    for (ts in IPR_all) {
      # Update reservoir volume with inflows
      V <- V + Qinflows[ts]
      # Rule #1: follow the objective filling curve (lower priority)
      j <- as.numeric(format(InputsModel$DatesR[ts], "%j"))
      Vobj_ts <- approx(Vobj, xout = j)$y
      Qrelease[ts] <- V - Vobj_ts
      # Rule #2: Release cannot be less than Qmin
      Qrelease[ts] <- max(Qmin, Qrelease[ts])
      # Rule #3: Release cannot be more than Qmax
      Qrelease[ts] <- min(Qmax, Qrelease[ts])
      # Update reservoir volume after release
      V <- V - Qrelease[ts]
      # Rule #4: hard constraints on the reservoir (full or empty?)
      if (V < 0) {
        Qrelease[ts] <- Qrelease[ts] + V
        V <- 0
      }
      V <- min(V, Vmax)
    }
    InputsModel$Qrelease <- Qrelease
    return(InputsModel)
  }
}
# A call to fun_factory_Regulation_Reservoir returns the regulation
# function with the parameters Qmin, Qmax, Vobj enclosed in the environment
# of the function
Regulation_Reservoir <-
  fun_factory_Regulation_Reservoir(
    RunOptions$Reservoir$IniStates,
    Vobj,
    Qmin,
    Qmax,
    Vmax
```

```
  )

# Then we need to update InputsModel in order to take into account the regulation
# function instead of predefined Qrelease in the previous study case
IM_reg <- CreateInputsModel(
  griwrm,
  DatesR = BasinObs$DatesR,
  Precip = Precip,
  PotEvap = PotEvap,
  Qrelease = Qrelease,
  FUN_REGUL = list(Reservoir = Regulation_Reservoir)
)

# And we can finally run the simulation!
OM_reg <- RunModel(IM_reg, RunOptions, Param)

# And plot the new result
plot(OM_reg$Reservoir, Vobs = Vobj$y[lubridate::yday(OM_reg$Reservoir$DatesR)])

# The supervised (closed-loop) simulation calls the control function at
# each time step from R, which is too slow for R CMD check. It is wrapped
# in \donttest so that R CMD check (and CRAN) skip it; it is still executed
# by the package CI (devtools::run_examples(run_donttest = TRUE)).
# See also the RunModel.Supervisor example.

############################################################
# Supervised reservoir management tracking an objective    #
# filling curve using a PID Controller with Vsim as        #
# controlled variable: Y = Reservoir$Vsim                  #
############################################################

# The objective here is to use a Supervisor with a Controller
# that regulates the reservoir to follow an annual objective
# filling curve defined by Vobj.
#
# This example demonstrates the Supervisor pattern for reservoir
# regulation, where the control logic is called at each time step
# during the simulation, using actual model outputs as feedback.
#
# The key difference from the local regulation example above is that
# in the Supervisor pattern, the control function receives actual
# simulated values from the model at each step, enabling true
# closed-loop control.

# PID control logic factory:
# Creates a control function that uses Proportional-Derivative (PD)
# control to track the objective volume curve.
#
# Parameters enclosed in the function environment:
# - Vobj: objective filling curve (volume by day of year)
# - Qmin: minimum release flow (m3/day)
# - Qmax: maximum release flow (m3/day)
#
```

```
# PID formula: U = Kp * error + Kd * dError
# where:
#   error = Vsim - Vobj_ts (volume error)
#   dError = error - prevError (rate of change of error)
#   Kp = proportional gain, Kd = derivative gain
#
# The derivative term compensates for the one-step delay in the
# Supervisor loop (Y values are from idx.output_previous) by anticipating
# the error trend and providing preemptive correction.
factoryReservoirLogic <- function(Vobj, Qmin, Qmax) {
  # PID control gains
  Kp <- 1 # Proportional gain: reacts to current volume error
  Kd <- 1 # Derivative gain: anticipates future error trend
  prevError <- 0 # Previous time step error (for derivative term)

  # Control logic function called by the Supervisor at each time step
  #
  # Input: Y = matrix of controlled variables from the model
  #        Y[1] = Vsim from the Reservoir node (simulated volume)
  #
  # Output: U = control action (release flow to apply)
  function(Y) {
    # Read simulated volume from the model
    Vsim <- Y[1]

    # Get day-of-year for the objective curve lookup
    j <- as.numeric(format(sv$ts.date, "%j"))
    Vobj_ts <- approx(Vobj, xout = j)$y

    # Compute volume error: positive means Vsim > Vobj (excess water)
    error <- Vsim - Vobj_ts

    # PD control action:
    # P term: proportional to current error
    # D term: proportional to error rate of change (dError)
    # The D term provides preemptive correction based on error trend
    dError <- error - prevError
    U <- Kp * error + Kd * dError

    # Store error for next time step derivative calculation
    prevError <- error

    # Apply physical constraints on release
    U <- max(Qmin, U)
    U <- min(Qmax, U)

    return(U)
  }
}

# Build the control logic function with parameters enclosed in environment
ReservoirLogic <- factoryReservoirLogic(
  Vobj = Vobj,
```

```
  Qmin = Qmin,
  Qmax = Qmax
)

# Create InputsModel with zero Qrelease (will be controlled by the Controller)
QreleaseEmpty <- data.frame(Reservoir = rep(0, length(BasinObs$DatesR)))
IM_Sup <- CreateInputsModel(
  griwrm,
  DatesR = BasinObs$DatesR,
  Precip = Precip,
  PotEvap = PotEvap,
  Qrelease = QreleaseEmpty
)

# Create the Supervisor object from InputsModel
sv <- CreateSupervisor(IM_Sup)

# Create the Controller:
# Y = "Reservoir$Vsim": reads simulated volume from Reservoir node
# U = "Reservoir$Qrelease": sets release flow for Reservoir node
# FUN = ReservoirLogic: the PD control function defined above
CreateController(
  sv,
  ctrl.id = "ReservoirObjectiveCurve",
  Y = "Reservoir$Vsim",
  U = "Reservoir$Qrelease",
  FUN = ReservoirLogic
)

# Run options with warm-up period starting from empty reservoir
RO_Sup <- CreateRunOptions(
  IM_Sup,
  IndPeriod_Run = Ind_Run,
  IndPeriod_WarmUp = seq.int(Ind_Run[1] - 365, length.out = 365),
  IniStates = list(Reservoir = c("Reservoir.V" = 0))
)

# Run the supervised simulation
OM_Sup <- RunModel(sv, RunOptions = RO_Sup, Param = Param)

# Plot the supervised simulation results
# Compare Vsim with the objective filling curve
plot(
  OM_Sup$Reservoir,
  Vobs = Vobj$y[lubridate::yday(OM_Sup$Reservoir$DatesR)]
)
```

`plot.Qm3s` *Plot of a* `Qm3s` *object (time series of simulated flows)*

Description

This function plot time series of flow rate in m3/s. It's a method for object of class "Qm3s" which can be directly called by `plot`. It can also be called as a function `plot.Qm3s` if the first parameter has the good format.

Usage

```
## S3 method for class 'Qm3s'
plot(
  x,
  type = "l",
  xlab = "Date",
  ylab = expression("Flow rate (m"^"3" * "/s)"),
  main = "Simulated flows",
  col = grDevices::hcl.colors(ncol(x) - 1, "Zissou 1"),
  legend = colnames(x)[-1],
  legend.cex = 0.7,
  legend.x = "topright",
  legend.y = NULL,
  lty = 1,
  mgp = c(2.5, 1, 0),
  ...
)
```

Arguments

`x`	data.frame with a first column with POSIXt dates and followings columns with flows at each node of the network
`type`	character plot type (See plot.default), default "l"
`xlab`	character label for the x axis, default to "Date"
`ylab`	character label for the y axis, default to "Flow (m3/s)"
`main`	character main title for the plot, default to "Simulated flows"
`col`	character plotting colors (See par)
`legend`	character see parameter `legend` of legend. Set it to NULL to hide the legend
`legend.cex`	character `cex` parameter for the text of the legend (See par)
`legend.x, legend.y`	Legend position, see `x` and `y` parameters in graphics::legend
`lty`	character or numeric The line type (See par)
`mgp`	The margin line for the axis title, axis labels and axis line (See par)
`...`	Further arguments to pass to the matplot functions

Details

For examples of use see topics RunModel.GRiwrmInputsModel, RunModel_Reservoir, and RunModel.Supervisor.

Value

Screen plot window.

`reduceGRiwrm` *Reduce the size of a GRiwrm by selecting the subset of nodes corresponding to a downstream node*

Description

Reduce the size of a GRiwrm by selecting the subset of nodes corresponding to a downstream node

Usage

```
reduceGRiwrm(griwrm, down_node, check = FALSE)
```

Arguments

griwrm	A *GRiwrm* object (See CreateGRiwrm)
down_node	The ID of the downstream node of the reduced *GRiwrm*
check	logical Check the consistency of the reduced *GRiwrm*

Value

A *GRiwrm* object only containing nodes located upstream the given downstream node

Examples

```
data(Severn)
nodes <- Severn$BasinsInfo
nodes$model <- "RunModel_GR4J"
str(nodes)
# Mismatch column names are renamed to stick with GRiwrm requirements
rename_columns <- list(id = "gauge_id",
                       down = "downstream_id",
                       length = "distance_downstream")
griwrm_severn <- CreateGRiwrm(nodes, rename_columns)
griwrm_severn
# Network diagram with upstream basin nodes in blue, intermediate sub-basin in green
plot(griwrm_severn)
plot(reduceGRiwrm(griwrm_severn, "54032"))
```

RunModel — *RunModel function for both* **airGR** *InputsModel and GRiwrmInputsModel object*

Description

RunModel function for both **airGR** InputsModel and GRiwrmInputsModel object

Usage

```
RunModel(x, ...)
```

Arguments

x	[object of class *InputsModel* or *GRiwrmInputsModel*] see CreateInputsModel for details
...	further arguments passed to or from other methods

Value

Either a list of OutputsModel object (for GRiwrmInputsModel) or an OutputsModel object (for InputsModel)

RunModel.GR — *Run of a rainfall-runoff model on a sub-basin*

Description

Run ONLY the rainfall-runoff model, upstream flow routing and Diversions are not processed.

Usage

```
## S3 method for class 'GR'
RunModel(x, RunOptions, Param, ...)
```

Arguments

x	[object of class InputsModel] InputsModel for airGR::RunModel
RunOptions	[object of class *RunOptions*] see airGR::CreateRunOptions for details
Param	numeric vector of model parameters (See details for SD lag model)
...	further arguments passed to or from other methods

Details

This function runs airGR::RunModel (without lag) and add an item Qsim_m3 to the returned *OutputsModel* object.

Value

[list] see RunModel_GR4J or RunModel_CemaNeigeGR4J for details.

If InputsModel parameter has been created for using a semi-distributed (SD) lag model (See CreateInputsModel), the list value contains an extra item named QsimDown which is a numeric series of simulated discharge [mm/time step] related to the run-off contribution of the downstream sub-catchment.

RunModel.GRiwrmInputsModel

RunModel function for GRiwrmInputsModel *object*

Description

RunModel function for *GRiwrmInputsModel* object

Usage

```
## S3 method for class 'GRiwrmInputsModel'
RunModel(x, RunOptions, Param, forceReservoirObs = FALSE, ...)
```

Arguments

x	[object of class *GRiwrmInputsModel*] see CreateInputsModel.GRiwrm for details
RunOptions	[object of class *GRiwrmRunOptions*] see CreateRunOptions.GRiwrmInputsModel for details
Param	list parameter values. The list item names are the IDs of the sub-basins. Each item is a numeric vector
forceReservoirObs	boolean indicating if reservoir observed flows must be forced during the model run. Default is TRUE (See details of RunModel_Reservoir)
...	Further arguments for compatibility with S3 methods

Value

An object of class *GRiwrmOutputsModel*. This object is a list of *OutputsModel* objects produced by RunModel.InputsModel for each node of the semi-distributed model.

It also contains the following attributes (see attr):

- "Qm3s": a data.frame containing the dates of simulation and one column by node with the simulated flows in cubic meters per seconds (See plot.Qm3s)
- "GRiwrm": a copy of the *GRiwrm* object produced by CreateGRiwrm and used for the simulation
- "TimeStep": time step of the simulation in seconds

See Also

CreateGRiwrm(), CreateInputsModel.GRiwrm(), CreateRunOptions()

Examples

```
######################################################################
# Run the `airGR::RunModel_Lag` example in the GRiwrm fashion way #
# Simulation of a reservoir with a purpose of low-flow mitigation #
######################################################################

## ---- preparation of the InputsModel object

## loading package and catchment data
library(airGRiwrm)
data(L0123001)

## ---- specifications of the reservoir

## the reservoir withdraws 1 m3/s when it's possible considering the flow observed in the basin
Qupstream <- matrix(
  -sapply(BasinObs$Qls / 1000 - 1, function(x) {
    min(1, max(0, x, na.rm = TRUE))
  }),
  ncol = 1
)

## except between July and September when the reservoir releases 3 m3/s for low-flow mitigation
month <- as.numeric(format(BasinObs$DatesR, "%m"))
Qupstream[month >= 7 & month <= 9] <- 3
Qupstream <- Qupstream * 86400 ## Conversion in m3/day

## the reservoir is not an upstream subcachment: its areas is NA
BasinAreas <- c(NA, BasinInfo$BasinArea)

## delay time between the reservoir and the catchment outlet is 2 days and the distance is 150 km
LengthHydro <- 150
## with a delay of 2 days for 150 km, the flow velocity is 75 km per day
Velocity <- (LengthHydro * 1e3 / 2) / (24 * 60 * 60) ## Conversion km/day -> m/s

# This example is a network of 2 nodes which can be describe like this:
db <- data.frame(
  id = c("Reservoir", "GaugingDown"),
  length = c(LengthHydro, NA),
  down = c("GaugingDown", NA),
  area = c(NA, BasinInfo$BasinArea),
  model = c(NA, "RunModel_GR4J"),
  stringsAsFactors = FALSE
)

# Create GRiwrm object from the data.frame
griwrm <- CreateGRiwrm(db)
if (interactive()) {
```

```
  plot(griwrm)
}

# Formatting observations for the hydrological models
# Each input data should be a matrix or a data.frame with the good id in the name of the column
Precip <- matrix(BasinObs$P, ncol = 1)
colnames(Precip) <- "GaugingDown"
PotEvap <- matrix(BasinObs$E, ncol = 1)
colnames(PotEvap) <- "GaugingDown"

# Observed flows contain flows that are directly injected in the model
Qinf = matrix(Qupstream, ncol = 1)
colnames(Qinf) <- "Reservoir"

# Creation of the GRiwrmInputsModel object (= a named list of InputsModel objects)
InputsModels <- CreateInputsModel(
  griwrm,
  DatesR = BasinObs$DatesR,
  Precip = Precip,
  PotEvap = PotEvap,
  Qinf = Qinf
)
str(InputsModels)

## run period selection
Ind_Run <- seq(
  which(format(BasinObs$DatesR, format = "%Y-%m-%d") == "1990-01-01"),
  which(format(BasinObs$DatesR, format = "%Y-%m-%d") == "1999-12-31")
)

# Creation of the GRiwmRunOptions object
RunOptions <- CreateRunOptions(InputsModels, IndPeriod_Run = Ind_Run)
str(RunOptions)

# Parameters of the SD models should be encapsulated in a named list
ParamGR4J <- c(X1 = 257.238, X2 = 1.012, X3 = 88.235, X4 = 2.208)
Param <- list(`GaugingDown` = c(Velocity, ParamGR4J))

# RunModel for the whole network
OutputsModels <- RunModel(InputsModels, RunOptions = RunOptions, Param = Param)
str(OutputsModels)

# Compare regimes of the simulation with reservoir and observation of natural flow
plot(
  OutputsModels,
  data.frame(GaugingDown = BasinObs$Qmm[Ind_Run]),
  which = "Regime"
)

# Plot together simulated flows (m3/s) of the reservoir and the gauging station
plot(attr(OutputsModels, "Qm3s"))
```

```
#########################################################
# Run the Severn example provided with this package    #
# A natural catchment composed with 6 gauging stations #
#########################################################

data(Severn)
nodes <- Severn$BasinsInfo
nodes$model <- "RunModel_GR4J"
# Mismatch column names are renamed to stick with GRiwrm requirements
rename_columns <- list(
  id = "gauge_id",
  down = "downstream_id",
  length = "distance_downstream"
)
g_severn <- CreateGRiwrm(nodes, rename_columns)

# Network diagram with upstream basin nodes in blue, intermediate sub-basin in green
if (interactive()) {
  plot(g_severn)
}

# Format CAMEL-GB meteorological dataset for airGRiwrm inputs
BasinsObs <- Severn$BasinsObs
DatesR <- BasinsObs[[1]]$DatesR
PrecipTot <- cbind(sapply(BasinsObs, function(x) {
  x$precipitation
}))
PotEvapTot <- cbind(sapply(BasinsObs, function(x) {
  x$peti
}))

# Precipitation and Potential Evaporation are related to the whole catchment
# at each gauging station. We need to compute them for intermediate catchments
# for use in a semi-distributed model
Precip <- ConvertMeteoSD(g_severn, PrecipTot)
PotEvap <- ConvertMeteoSD(g_severn, PotEvapTot)

# CreateInputsModel object
IM_severn <- CreateInputsModel(g_severn, DatesR, Precip, PotEvap)

# GRiwrmRunOptions object
# Run period is set aside the one-year warm-up period
IndPeriod_Run <- seq(
  which(
    IM_severn[[1]]$DatesR == (IM_severn[[1]]$DatesR[1] + 365 * 24 * 60 * 60)
  ),
  length(IM_severn[[1]]$DatesR) # Until the end of the time series
)
IndPeriod_WarmUp <- seq(1, IndPeriod_Run[1] - 1)

RO_severn <- CreateRunOptions(
  IM_severn,
  IndPeriod_WarmUp = IndPeriod_WarmUp,
```

```
  IndPeriod_Run = IndPeriod_Run
)

# Load parameters of the model from Calibration in vignette V02
P_severn <- readRDS(system.file(
  "vignettes",
  "ParamV02.RDS",
  package = "airGRiwrm"
))

# Run the simulation
OM_severn <- RunModel(IM_severn, RunOptions = RO_severn, Param = P_severn)

# Plot results of simulated flows in m3/s
Qm3s <- attr(OM_severn, "Qm3s")
plot(Qm3s[1:150, ])


######################################################################
# An example of water withdrawal for irrigation with restriction #
# modeled with a Diversion node on the Severn river              #
######################################################################

# A diversion is added at gauging station "54001"
nodes_div <- nodes[, c(
  "gauge_id",
  "downstream_id",
  "distance_downstream",
  "area",
  "model"
)]
names(nodes_div) <- c("id", "down", "length", "area", "model")
nodes_div <- rbind(
  nodes_div,
  data.frame(
    id = "54001", # location of the diversion
    down = NA, # the abstracted flow goes outside
    length = NA, # down=NA, so length=NA
    area = NA, # no area, diverted flow is in m3/day
    model = "Diversion"
  )
)

g_div <- CreateGRiwrm(nodes_div)
# The node "54001" is surrounded in red to show the diverted node
if (interactive()) {
  plot(g_div)
}

# Computation of the irrigation withdraw objective
irrigMonthlyPlanning <- c(
  0.0,
  0.0,
```

```
  1.2,
  2.4,
  3.2,
  3.6,
  3.6,
  2.8,
  1.8,
  0.0,
  0.0,
  0.0
)
names(irrigMonthlyPlanning) <- month.abb
irrigMonthlyPlanning
DatesR_month <- as.numeric(format(DatesR, "%m"))
# Withdrawn flow calculated for each day is negative
Qirrig <- matrix(-irrigMonthlyPlanning[DatesR_month] * 86400, ncol = 1)
colnames(Qirrig) <- "54001"

# Minimum flow to remain downstream the diversion is 12 m3/s
Qmin <- matrix(12 * 86400, nrow = length(DatesR), ncol = 1)
colnames(Qmin) = "54001"

# Creation of GRimwrInputsModel object
IM_div <- CreateInputsModel(
  g_div,
  DatesR,
  Precip,
  PotEvap,
  Qinf = Qirrig,
  Qmin = Qmin
)

# RunOptions and parameters are unchanged, we can directly run the simulation
OM_div <- RunModel(IM_div, RunOptions = RO_severn, Param = P_severn)

# Retrieve diverted flow at "54001" and convert it from m3/day to m3/s
Qdiv_m3s <- OM_div$`54001`$Qdiv_m3 / 86400

# Plot the diverted flow for the year 2003
Ind_Plot <- which(
  OM_div[[1]]$DatesR >= as.POSIXct("2003-01-01", tz = "UTC") &
    OM_div[[1]]$DatesR <= as.POSIXct("2003-12-31", tz = "UTC")
)
dfQdiv <- as.Qm3s(
  DatesR = OM_div[[1]]$DatesR[Ind_Plot],
  Diverted_flow = Qdiv_m3s[Ind_Plot]
)

oldpar <- par(mfrow = c(2, 1), mar = c(2.5, 4, 1, 1))
plot(dfQdiv)

# Plot natural and influenced flow at station "54001"
df54001 <- cbind(
```

```
  attr(OM_div, "Qm3s")[Ind_Plot, c("DatesR", "54001")],
  attr(OM_severn, "Qm3s")[Ind_Plot, "54001"]
)
names(df54001) <- c("DatesR", "54001 with irrigation", "54001 natural flow")
df54001 <- as.Qm3s(df54001)
plot(df54001, ylim = c(0, 70))
abline(h = 12, col = "green", lty = "dotted")
par(oldpar)
```

`RunModel.GRiwrmOutputsModel` *RunModel for hot restart after a previous simulation period*

Description

This function restarts a simulation using the state at the end of a previous simulation (`GRiwrmOutputsModel` object `x`). It allows redefining the boundary conditions `Qinf`, `Qrelease`, and `Qmin` for the new period.

Usage

```
## S3 method for class 'GRiwrmOutputsModel'
RunModel(
  x,
  InputsModel,
  RunOptions,
  IndPeriod_Run = which(InputsModel[[1]]$DatesR %in% DatesR),
  DatesR = getNextTimeSteps(x),
  Qinf = NULL,
  Qrelease = NULL,
  Qmin = NULL,
  merge_outputs = TRUE,
  ...
)
```

Arguments

`x`	`GRiwrmOutputsModel` object resulting from a previous run.
`InputsModel`	`GRiwrmInputsModel` object (see CreateInputsModel.GRiwrm) or `Supervisor` object (see CreateSupervisor).
`RunOptions`	List of run options created with CreateRunOptions.
`IndPeriod_Run`	Integer vector indicating the indices of the time steps to run. Must start at the index immediately following the previous run.
`DatesR`	(optional) `POSIXt` vector of dates for the simulation period. See details.

Qinf	(optional) matrix or data.frame of numeric observed flows for nodes of type "Direct injection" and "Diversion" (m³ per time step). Column names correspond to node IDs. Negative flows are abstracted from the model and positive flows are injected to the model. See details.
Qrelease	(optional) matrix or data.frame of numeric release flows by nodes using the model RunModel_Reservoir (m³ per time step). See details.
Qmin	(optional) matrix or data.frame of numeric minimum flows for downstream of a Diversion node (m³ per time step). Default is zero. Column names correspond to node IDs. See details.
merge_outputs	logical Merge simulation outputs with the one provided in argument x.
...	Further arguments for compatibility with S3 methods.

Details

IndPeriod_Run or DatesR must be continuous periods starting the time step after the last simulation time step of the GRiwrmOutputsModel object provided through the argument x.

Qinf, Qmin, and Qrelease are used to overwrite the corresponding arguments provided to CreateInputsModel.GRiwrm for the period to be simulated. Therefore, the number of rows of these arguments must correspond to IndPeriod_Run or DatesR lengths.

Value

An object of class GRiwrmOutputsModel (see RunModel.GRiwrmInputsModel for details).

See Also

CreateGRiwrm(), CreateInputsModel.GRiwrm(), CreateRunOptions()

Vignette "V07_Combine_tactical_operational_management" in package airGRiwrm

---

RunModel.InputsModel *Wrapper for airGR::RunModel for one sub-basin*

Description

Wrapper for airGR::RunModel for one sub-basin

Usage

```
## S3 method for class 'InputsModel'
RunModel(x = NULL, RunOptions, Param, FUN_MOD = NULL, InputsModel = NULL, ...)
```

Arguments

x	[object of class *InputsModel*] see airGR::CreateInputsModel for details
RunOptions	[object of class *RunOptions*] see CreateRunOptions for details
Param	[numeric] vector of model parameters (See details for SD lag model)
FUN_MOD	[function] hydrological model function (e.g. RunModel_GR4J, RunModel_CemaNeigeGR4J)
InputsModel	[object of class *InputsModel*] see CreateInputsModel for details
...	Further arguments for compatibility with S3 methods

Value

[object of class *OutputsModel*] returned by airGR::RunModel (See Value section of airGR::RunModel_GR4J) completed by new items:

- Qsim_m3: simulated flow in cubic meters per time step
- Qover_m3 volumes of over abstractions which occurs when RunModel_Lag warns for negative simulated flows
- Qnat: only present in case of Diversion in the node, simulated flow in mm per time step before application of the Diversion
- Qdiv_m3: only present in case of Diversion in the node, simulated diverted flow in cubic meters per time step. The latter differs from the flows time series provided in argument Qinf of CreateInputsModel.GRiwrm by the limitation of diversion applied by the minimum flow threshold Qmin to keep flowing in the river

---

RunModel.Supervisor *RunModel function for a Supervisor*

Description

This function acts as RunModel.GRiwrmInputsModel but for a model containing a Supervisor.

Usage

```
## S3 method for class 'Supervisor'
RunModel(x, RunOptions, Param, Yinit = NULL, ...)
```

Arguments

x	Object of class Supervisor see CreateSupervisor for details
RunOptions	Object of class GRiwrmRunOptions, see CreateRunOptions for details
Param	list of parameter values (See . The list item names are the IDs of the sub-basins. Each item is a vector of numerical parameters
Yinit	list of initial values for the Y variables of the controllers (See details)
...	Further arguments for compatibility with S3 methods

Details

`Yinit` is used for allowing to run the supervisor at the first supervision time step when no simulation data are available. It's a list with items named by the controller ids, and each item is a matrix with one column by controlled and/or measured variable `Y` for this controller and one row by supervision time step (See CreateSupervisor).

Value

GRiwrmOutputsModel object which is a list of *OutputsModel* objects (See airGR::RunModel) for each node of the semi-distributed model

Examples

```
################################################################################
# An example of reservoir management on an hypothetical dam at station "54095"
# on the Severn river build to support low-flows at "54057"
################################################################################
# A minimum flow of 50 m3/s is maintained at the dam location and an extra-release
# is provided when the flow at the downstream station "54057" cross a minimum
# threshold of 65 m3/s. The dam has a storage capacity of 650 millions m3
################################################################################
library(airGRiwrm)

# Load Severn network information
data(Severn)
nodes <- Severn$BasinsInfo[, c(
  "gauge_id",
  "downstream_id",
  "distance_downstream",
  "area"
)]
nodes$model <- "RunModel_GR4J"

# Insert a dam downstream the location the gauging station 54095
# The dam is a direct injection node
nodes$downstream_id[nodes$gauge_id == "54095"] <- "Dam"
nodes$distance_downstream[nodes$gauge_id == "54095"] <- 0
nodes <- rbind(
  nodes,
  data.frame(
    gauge_id = "Dam",
    downstream_id = "54001",
    distance_downstream = 42,
    area = NA,
    model = "RunModel_Reservoir"
  )
)

griwrm <- CreateGRiwrm(
  nodes,
  list(id = "gauge_id", down = "downstream_id", length = "distance_downstream")
)
```

```
if (interactive()) {
  plot(griwrm)
}

# Format meteorological inputs for CreateInputs
BasinsObs <- Severn$BasinsObs
DatesR <- BasinsObs[[1]]$DatesR
PrecipTot <- cbind(sapply(BasinsObs, function(x) {
  x$precipitation
}))
PotEvapTot <- cbind(sapply(BasinsObs, function(x) {
  x$peti
}))
Precip <- ConvertMeteoSD(griwrm, PrecipTot)
PotEvap <- ConvertMeteoSD(griwrm, PotEvapTot)

# Create a release flow time series for the dam
# This release will be modified by the Supervisor
# We initiate it with the natural flow for having a good initialization of the
# model at the first time step of the running period
Qrelease <- data.frame(
  Dam = BasinsObs$`54095`$discharge_spec *
    griwrm$area[griwrm$id == "54095"] *
    1E3
)

# InputsModel object
IM_severn <- CreateInputsModel(
  griwrm,
  DatesR,
  Precip,
  PotEvap,
  Qrelease = Qrelease
)

# Initialization of the Supervisor
sv <- CreateSupervisor(IM_severn)

# Dam management is modeled by a controller
# This controller releases a minimum flow Qmin and provides
# extra release if flow measured somewhere is below Qthreshold
# Flow is expressed in m3 / time step
# Y[1] = runoff flow at gauging station 54095 filling the reservoir
# Y[2] = flow at gauging station 54057, location of the low-flow objective
# The returned value is the release calculated at the reservoir
# We need to enclose the Supervisor variable and other parameters in
# the environment of the function with a function returning the logic function
factoryDamLogic <- function(sv, Qmin, Qthreshold) {
  function(Y) {
    # Estimate natural flow at low-flow support location
    Qnat <- Y[1] - Y[2]
    # The release is the max between: low-flow support and minimum flow
    U <- max(Qthreshold - Qnat, Qmin)
```

```
    return(U)
  }
}

# And define a final function enclosing logic and parameters together
funDamLogic <- factoryDamLogic(
  sv = sv, # The Supervisor which store the released flow
  Qmin = 50 * 86400, # Min flow to maintain downstream the reservoir (m3/day)
  Qthreshold = 65 * 86400 # Min flow threshold to support at station 54057 (m3/day)
)

CreateController(
  sv,
  "DamRelease",
  Y = c("54057", "Dam"),
  U = "Dam",
  FUN = funDamLogic
)

# GRiwrmRunOptions object simulation of the hydrological year 2002-2003
IndPeriod_Run <- which(
  DatesR >= as.POSIXct("2002-11-01", tz = "UTC") &
    DatesR <= as.POSIXct("2003-11-01", tz = "UTC")
)
IndPeriod_WarmUp <- seq.int(IndPeriod_Run[1] - 366, IndPeriod_Run[1] - 1)
RO_severn <- CreateRunOptions(
  IM_severn,
  IndPeriod_WarmUp = IndPeriod_WarmUp,
  IndPeriod_Run = IndPeriod_Run
)

# Load parameters of the model from Calibration in vignette V02
P_severn <- readRDS(system.file(
  "vignettes",
  "ParamV02.RDS",
  package = "airGRiwrm"
))

# Set the reservoir parameters: maximum storage capacity and celerity of inflows
# As the distance between the upstream node "54095" and the dam is 0 km, the celerity
# doesn't have any effect. However it must be positive.
P_severn$Dam <- c(Vmax = 650E6, celerity = 1)

# The Supervisor is used instead of InputsModel for running the model
OM_dam <- RunModel(sv, RunOptions = RO_severn, Param = P_severn)

# Plotting the time series of flows and reservoir storage
plot(OM_dam$Dam)
```

RunModel_Reservoir *Run with a reservoir model*

Description

The reservoir model is a model combining a lag model and the calculation of the water storage time series according to the released flow time series from the `Qrelease` parameter of CreateInputsModel.GRiwrm.

Usage

```
RunModel_Reservoir(InputsModel, RunOptions, Param)
```

Arguments

`InputsModel`	[object of class *InputsModel*] see `CreateInputsModel` for details
`RunOptions`	[object of class *RunOptions*] see `CreateRunOptions` for details
`Param`	numeric vector of length 2 containing (1) the maximum capacity of the reservoir and (2) the celerity in m/s of the upstream inflows.

Details

During calibration, the simulated flow corresponds exactly to the released flow in order to get downstream flows corresponding to observed flows. On a regular RunModel process, the simulated flow also corresponds to the released flow except when the reservoir is empty (release flow is limited) or full (release flow is completed by excess inflows). The same treatment is applied to diverted flows in case of Diversion model applied in the reservoir.

By default, the initial reservoir volume at the beginning of the warm-up period is equal to the half of the maximum reservoir capacity.

The parameters of the model can't be calibrated and must be fixed during the calibration process by using this instruction after the call to CreateCalibOptions:

```
CalibOptions[[id_of_the_reservoir]]$FixedParam <- c(Vmax, celerity)
```

Initial states of the model consists in the initial volume storage in the reservoir and can be defined with the `IniStates` in the call to CreateRunOptions.GRiwrmInputsModel (See example below).

The final state of the reservoir is stored in `OutputsModel$StateEnd` and can be reused for starting a new simulation with the following instruction:

```
RunOptions[[id_of_the_reservoir]]$IniStates <- unlist(OutputsModel[[id_of_the_reservoir]]$StateEnd)
```

Direct injection nodes connected to a reservoir nodes act as water injections or withdrawals directly in the reservoir volume (whatever the length between the direct injection nodes and the reservoir node). The abstraction volumes that cannot be operated due to an empty reservoir are notified by the item `Qover_m3` in the returned *OutputsModel* object.

Value

An *OutputsModel* object like the one return by airGR::RunModel but completed with the items:

- `Vsim`: representing the water volume time series in m3
- `Qsim_m3`: flow released by the reservoir in cubic meters by time step (see Details)

- `Qdiv_m3`: only present in case of Diversion in the node, diverted flow in cubic meters per time step. The latter differs from the flows time series provided in argument `Qinf` of CreateInputsModel.GRiwrm by the limitation due to an empty reservoir
- `Qover_m3`: only present in case of Diversion in the node, diverted volumes that cannot be operated due to an empty reservoir

Examples

```
########################################################
# Daily time step simulation of a reservoir filled by #
# one catchment supplying a constant released flow    #
########################################################

library(airGRiwrm)
data(L0123001)

# Inflows comes from a catchment of 360 km² modeled with GR4J
# The reservoir receives directly the inflows
db <- data.frame(
  id = c(BasinInfo$BasinCode, "Reservoir"),
  length = c(0, NA),
  down = c("Reservoir", NA),
  area = c(BasinInfo$BasinArea, NA),
  model = c("RunModel_GR4J", "RunModel_Reservoir"),
  stringsAsFactors = FALSE
)
griwrm <- CreateGRiwrm(db)
if (interactive()) {
  plot(griwrm)
}

# This catchment inflows a reservoir of maximum capacity Vmax
Vmax <- 4E6 # in m3

# Formatting of GR4J inputs for airGRiwrm (matrix or data.frame with one
# column by sub-basin and node IDs as column names)
Precip <- matrix(BasinObs$P, ncol = 1)
colnames(Precip) <- BasinInfo$BasinCode
PotEvap <- matrix(BasinObs$E, ncol = 1)
colnames(PotEvap) <- BasinInfo$BasinCode

# We propose to compute the constant released flow from
# the Q20 of the natural flow
# The value is in m3 by time step (day)
(Qrelease <- quantile(BasinObs$Qls, na.rm = TRUE, probs = 0.2) / 1000 * 86400)

# Formatting of reservoir released flow inputs for airGRiwrm (matrix or data.frame
# with one column by node and node IDs as column names)
Qrelease <- data.frame(Reservoir = rep(Qrelease, length(BasinObs$DatesR)))

InputsModel <- CreateInputsModel(
  griwrm,
```

```
  DatesR = BasinObs$DatesR,
  Precip = Precip,
  PotEvap = PotEvap,
  Qrelease = Qrelease
)

## run period selection
Ind_Run <- seq(
  which(format(BasinObs$DatesR, format = "%Y-%m-%d") == "1990-01-01"),
  which(format(BasinObs$DatesR, format = "%Y-%m-%d") == "1999-12-31")
)

# Creation of the GRiwrmRunOptions object with the initial states
# of the reservoir set to 0 (empty reservoir)
RunOptions <- CreateRunOptions(
  InputsModel,
  IndPeriod_Run = Ind_Run,
  IndPeriod_WarmUp = seq.int(Ind_Run[1] - 365, length.out = 365),
  IniStates = list(Reservoir = c("Reservoir.V" = 0))
)

# calibration criterion: preparation of the InputsCrit object
Qobs <- data.frame("L0123001" = BasinObs$Qmm[Ind_Run])
InputsCrit <- CreateInputsCrit(
  InputsModel,
  ErrorCrit_KGE2,
  RunOptions = RunOptions,
  Obs = Qobs
)

# preparation of CalibOptions object with fixed parameters for the reservoir
# The capacity of the reservoir is set to Vmax and the inflow celerity is 0.5 m/s.
CalibOptions <- CreateCalibOptions(
  InputsModel,
  FixedParam = list(Reservoir = c(Vmax = Vmax, celerity = 0.5))
)

OC <- Calibration(
  InputsModel = InputsModel,
  RunOptions = RunOptions,
  InputsCrit = InputsCrit,
  CalibOptions = CalibOptions
)

# Model parameters
Param <- extractParam(OC)
str(Param)

# Running simulation
OutputsModel <- RunModel(InputsModel, RunOptions, Param)

# Plot the simulated flows and volumes on all nodes
Qobs <- cbind(BasinObs$Qmm[Ind_Run], Qrelease[Ind_Run, ])
```

```
colnames(Qobs) <- griwrm$id
plot(OutputsModel, Qobs = Qobs)

# The plot for the reservoir can also be plotted alone
plot(OutputsModel$Reservoir, Qobs = Qobs[, "Reservoir"])

#######################################################
# Daily time step simulation of a reservoir tracking  #
# an objective filling curve using a local regulation #
#######################################################

# The objective here is to simulate the same reservoir as above
# but with new rules:
# - A minimum flow downstream the reservoir defined as:
(Qmin <- Qrelease[1, ])
# - A maximum release flow of 20 m3/s for flood mitigation
(Qmax <- 20 * 86400)
# - An annual objective filling curve managing floods and droughts by
# trying to keep the reservoir volume between 2 and 8 Mm3:
Vobj <- approx(c(1, 150, 300, 366), c(1E6, 3.5E6, 0.5E6, 1E6), seq(366))
plot(Vobj, type = "l", col = "red", lty = 2)

# The regulation function takes InputsModel of the reservoir node and the
# global GRiwrm OutputsModel as arguments and returns a modified
# InputsModel used by RunModel_Reservoir afterward
fun_factory_Regulation_Reservoir <- function(Vini, Vobj, Qmin, Qmax, Vmax) {
  function(InputsModel, RunOptions, OutputsModel, env) {
    # Release flow time series initialisation
    Qrelease <- rep(0, length(InputsModel$DatesR))
    # Build inflows time series from upstream Qsim (warmup & run)
    Qinflows <- Qrelease
    IPR_all <- c(RunOptions$IndPeriod_WarmUp, RunOptions$IndPeriod_Run)
    Qinflows[IPR_all] <- c(
      OutputsModel$L0123001$RunOptions$WarmUpQsim_m3,
      OutputsModel$L0123001$Qsim_m3
    )
    # Reservoir volume initialisation
    V <- Vini
    # Loop over simulation time steps (warmup & run periods)
    for (ts in IPR_all) {
      # Update reservoir volume with inflows
      V <- V + Qinflows[ts]
      # Rule #1: follow the objective filling curve (lower priority)
      j <- as.numeric(format(InputsModel$DatesR[ts], "%j"))
      Vobj_ts <- approx(Vobj, xout = j)$y
      Qrelease[ts] <- V - Vobj_ts
      # Rule #2: Release cannot be less than Qmin
      Qrelease[ts] <- max(Qmin, Qrelease[ts])
      # Rule #3: Release cannot be more than Qmax
      Qrelease[ts] <- min(Qmax, Qrelease[ts])
      # Update reservoir volume after release
      V <- V - Qrelease[ts]
      # Rule #4: hard constraints on the reservoir (full or empty?)
```

```
      if (V < 0) {
        Qrelease[ts] <- Qrelease[ts] + V
        V <- 0
      }
      V <- min(V, Vmax)
    }
    InputsModel$Qrelease <- Qrelease
    return(InputsModel)
  }
}
# A call to fun_factory_Regulation_Reservoir returns the regulation
# function with the parameters Qmin, Qmax, Vobj enclosed in the environment
# of the function
Regulation_Reservoir <-
  fun_factory_Regulation_Reservoir(
    RunOptions$Reservoir$IniStates,
    Vobj,
    Qmin,
    Qmax,
    Vmax
  )

# Then we need to update InputsModel in order to take into account the regulation
# function instead of predefined Qrelease in the previous study case
IM_reg <- CreateInputsModel(
  griwrm,
  DatesR = BasinObs$DatesR,
  Precip = Precip,
  PotEvap = PotEvap,
  Qrelease = Qrelease,
  FUN_REGUL = list(Reservoir = Regulation_Reservoir)
)

# And we can finally run the simulation!
OM_reg <- RunModel(IM_reg, RunOptions, Param)

# And plot the new result
plot(OM_reg$Reservoir, Vobs = Vobj$y[lubridate::yday(OM_reg$Reservoir$DatesR)])

# The supervised (closed-loop) simulation calls the control function at
# each time step from R, which is too slow for R CMD check. It is wrapped
# in \donttest so that R CMD check (and CRAN) skip it; it is still executed
# by the package CI (devtools::run_examples(run_donttest = TRUE)).
# See also the RunModel.Supervisor example.

############################################################
# Supervised reservoir management tracking an objective    #
# filling curve using a PID Controller with Vsim as        #
# controlled variable: Y = Reservoir$Vsim                  #
############################################################

# The objective here is to use a Supervisor with a Controller
# that regulates the reservoir to follow an annual objective
```

```
# filling curve defined by Vobj.
#
# This example demonstrates the Supervisor pattern for reservoir
# regulation, where the control logic is called at each time step
# during the simulation, using actual model outputs as feedback.
#
# The key difference from the local regulation example above is that
# in the Supervisor pattern, the control function receives actual
# simulated values from the model at each step, enabling true
# closed-loop control.

# PID control logic factory:
# Creates a control function that uses Proportional-Derivative (PD)
# control to track the objective volume curve.
#
# Parameters enclosed in the function environment:
# - Vobj: objective filling curve (volume by day of year)
# - Qmin: minimum release flow (m3/day)
# - Qmax: maximum release flow (m3/day)
#
# PID formula: U = Kp * error + Kd * dError
# where:
#   error = Vsim - Vobj_ts (volume error)
#   dError = error - prevError (rate of change of error)
#   Kp = proportional gain, Kd = derivative gain
#
# The derivative term compensates for the one-step delay in the
# Supervisor loop (Y values are from idx.output_previous) by anticipating
# the error trend and providing preemptive correction.
factoryReservoirLogic <- function(Vobj, Qmin, Qmax) {
  # PID control gains
  Kp <- 1 # Proportional gain: reacts to current volume error
  Kd <- 1 # Derivative gain: anticipates future error trend
  prevError <- 0 # Previous time step error (for derivative term)

  # Control logic function called by the Supervisor at each time step
  #
  # Input: Y = matrix of controlled variables from the model
  #        Y[1] = Vsim from the Reservoir node (simulated volume)
  #
  # Output: U = control action (release flow to apply)
  function(Y) {
    # Read simulated volume from the model
    Vsim <- Y[1]

    # Get day-of-year for the objective curve lookup
    j <- as.numeric(format(sv$ts.date, "%j"))
    Vobj_ts <- approx(Vobj, xout = j)$y

    # Compute volume error: positive means Vsim > Vobj (excess water)
    error <- Vsim - Vobj_ts

    # PD control action:
```

```
    # P term: proportional to current error
    # D term: proportional to error rate of change (dError)
    # The D term provides preemptive correction based on error trend
    dError <- error - prevError
    U <- Kp * error + Kd * dError

    # Store error for next time step derivative calculation
    prevError <- error

    # Apply physical constraints on release
    U <- max(Qmin, U)
    U <- min(Qmax, U)

    return(U)
  }
}

# Build the control logic function with parameters enclosed in environment
ReservoirLogic <- factoryReservoirLogic(
  Vobj = Vobj,
  Qmin = Qmin,
  Qmax = Qmax
)

# Create InputsModel with zero Qrelease (will be controlled by the Controller)
QreleaseEmpty <- data.frame(Reservoir = rep(0, length(BasinObs$DatesR)))
IM_Sup <- CreateInputsModel(
  griwrm,
  DatesR = BasinObs$DatesR,
  Precip = Precip,
  PotEvap = PotEvap,
  Qrelease = QreleaseEmpty
)

# Create the Supervisor object from InputsModel
sv <- CreateSupervisor(IM_Sup)

# Create the Controller:
# Y = "Reservoir$Vsim": reads simulated volume from Reservoir node
# U = "Reservoir$Qrelease": sets release flow for Reservoir node
# FUN = ReservoirLogic: the PD control function defined above
CreateController(
  sv,
  ctrl.id = "ReservoirObjectiveCurve",
  Y = "Reservoir$Vsim",
  U = "Reservoir$Qrelease",
  FUN = ReservoirLogic
)

# Run options with warm-up period starting from empty reservoir
RO_Sup <- CreateRunOptions(
  IM_Sup,
  IndPeriod_Run = Ind_Run,
```

```
  IndPeriod_WarmUp = seq.int(Ind_Run[1] - 365, length.out = 365),
  IniStates = list(Reservoir = c("Reservoir.V" = 0))
)

# Run the supervised simulation
OM_Sup <- RunModel(sv, RunOptions = RO_Sup, Param = Param)

# Plot the supervised simulation results
# Compare Vsim with the objective filling curve
plot(
  OM_Sup$Reservoir,
  Vobs = Vobj$y[lubridate::yday(OM_Sup$Reservoir$DatesR)]
)
```

`Severn` *Catchment attributes and hydro-meteorological timeseries for some gauging stations on the Severn River*

Description

Catchment attributes and hydro-meteorological timeseries for some gauging stations on the Severn River

Usage

```
Severn
```

Format

a list with 2 items:

- "BasinsInfo" which contains a data.frame with Gauging station identifier, name, coordinates (GPS), area (km2), mean elevation (m), station type, flow period start and end, the bank full flow (m3/s), the identifier of the following downstream station and the distance to the following downstream station
- "BasinObs" which contains a list with an item by gauging station which contains a data.frame with base::POSIXct dates, precipitations (mm/time step), potential evapotranspiration (mm/time step) and measured flows (mm/time step)

Source

These data are extracted from the CAMEL-GB dataset.

Coxon, G.; Addor, N.; Bloomfield, J.P.; Freer, J.; Fry, M.; Hannaford, J.; Howden, N.J.K.; Lane, R.; Lewis, M.; Robinson, E.L.; Wagener, T.; Woods, R. (2020). Catchment attributes and hydro-meteorological timeseries for 671 catchments across Great Britain (CAMELS-GB). NERC Environmental Information Data Centre. (Dataset). doi:10.5285/8344E4F3D2EA44F58AFA86D2987543A9

`sort.GRiwrm` *Sort a GRiwrm network in upstream-downstream order ready for Calibration*

Description

It Uses getNodeRanking for determining the best order for calibration and leaves direct injection nodes at the tail of the list.

Usage

```
## S3 method for class 'GRiwrm'
sort(x, decreasing = FALSE, ...)
```

Arguments

`x`	A *GRiwrm* object (See CreateGRiwrm)
`decreasing`	logical. Should the sort be increasing or decreasing? Not available for partial sorting.
`...`	arguments to be passed to or from methods or (for the default methods and objects without a class) to `sort.int`.

Value

The sorted *GRiwrm* object in upstream-downstream order ready for Calibration

`transferGRparams` *Transfer GR parameters from one donor sub-basin to a receiver sub-basin*

Description

This function is used by `Calibration.GRiwrmInputsModel` for transferring parameters to ungauged nodes and for providing a priori parameters in parameter regularization (See CreateInputsCrit).

Usage

```
transferGRparams(
  InputsModel,
  Param,
  donor,
  receiver,
  default_param = NULL,
  verbose = FALSE
)
```

Arguments

`InputsModel`	Object of class GRiwrmInputsModel, see CreateInputsModel.GRiwrm for details
`Param`	numeric vector of GR model parameters
`donor`	character id of the node which gives its parameters
`receiver`	character id of the node which receives the parameters from the donor
`default_param`	numeric vector of GR model parameters if parameters are missing from the donor
`verbose`	logical Add information message on donor and receiver

Details

`donor` and `receiver` nodes should have the same GR model with the same snow module configuration.

The transfer takes care of:

- the presence/absence of hydraulic routing parameters between the donor and the receiver (But `default_param` should be provided if the receiver has more parameters than the donor)
- the transformation of the X4 parameters of GR models. Indeed, this parameter is correlated to the catchment area. Therefore, it has to be rescaled from the surface of the donor catchment to the surface of the receiver catchment, following the relationship suggested by Lobligeois (2014): $X4_{receiver} = X4_{donor} \times (S_{receiver}/S_{donor})^{0.3}$.

Value

A numeric vector of transferred parameters

References

Lobligeois, F., 2014. Mieux connaitre la distribution spatiale des pluies améliore-t-il la modélisation des crues ? Diagnostic sur 181 bassins versants français. Thèse de Doctorat, Irstea (Antony), AgroParisTech (Paris), 312 pp. https://webgr.inrae.fr/Media/Files/biblio-theses/2014_lobligeois_these.pdf

Index

∗ **datasets**
    Severn, 72

adjustMeteoToCatchment, 3
adjustMeteoToCatchment(), *43*
airGR::Calibration, *7*
airGR::CreateCalibOptions, *10, 11*
airGR::CreateIniStates, *28*
airGR::CreateInputsCrit, *21, 22*
airGR::CreateInputsCrit_Lavenne, *21*
airGR::CreateInputsModel, *23*, *28*, *61*
airGR::CreateInputsModel(), *24*, *26*
airGR::CreateRunOptions, *28, 29*, *52*
airGR::CreateRunOptions(), *25*
airGR::ErrorCrit_NSE, *20*
airGR::ErrorCrit_RMSE, *20*
airGR::plot.OutputsModel, *42, 43*
airGR::RunModel, *21*, *36*, *52*, *60–62*, *65*
airGR::RunModel_GR4J, *61*
as.Qm3s, 5
attr, *53*

base::POSIXct, *30*, *72*

Calibration, *31*
Calibration
        *(*Calibration.GRiwrmInputsModel*)*, 6
Calibration(), *11*, *22*
Calibration.GRiwrmInputsModel, 6, *30, 31*
character, *9, 10*, *16–18*, *21*, *28*, *30*, *32–37*, *39*, *42*, *50*, *74*
colnames, *9*
ConvertMeteoSD, 9
CreateCalibOptions, *7*, *65*
CreateCalibOptions
        *(*CreateCalibOptions.GRiwrmInputsModel*)*, 10
CreateCalibOptions(), *7*, *22*
CreateCalibOptions.GRiwrmInputsModel, 10
CreateController, 15, *30*
CreateController(), *30*
CreateGRiwrm, *3, 4*, *9*, 16, *21*, *24, 25*, *30*, *32*, *34*, *36*, *39*, *42*, *51*, *53*, *73*
CreateGRiwrm(), *4*, *7*, *10, 11*, *22*, *26*, *29*, *33, 34*, *39*, *54*, *60*
CreateInputsCrit, *7*, *73*
CreateInputsCrit
        *(*CreateInputsCrit.GRiwrmInputsModel*)*, 20
CreateInputsCrit(), *7*, *11*
CreateInputsCrit.GRiwrmInputsModel, 20
CreateInputsModel, *7*, *10*, *20*, 23, *28*, *31*, *52, 53*, *61*, *65*
CreateInputsModel.GRiwrm, *11*, *23*, 24, *30*, *36*, *53*, *59–61*, *65, 66*, *74*
CreateInputsModel.GRiwrm(), *7*, *10, 11*, *22*, *24*, *29*, *54*, *60*
CreateRunOptions, *4*, *7*, *20*, *42*, *59*, *61*, *65*
CreateRunOptions
        *(*CreateRunOptions.GRiwrmInputsModel*)*, 27
CreateRunOptions(), *11*, *22*, *26*, *54*, *60*
CreateRunOptions.GRiwrmInputsModel, 27, *28*, *53*, *65*
CreateRunOptions.InputsModel, *28*
CreateSupervisor, *16*, 29, *59*, *61, 62*
CreateSupervisor(), *16*

data.frame, *6*, *9*, *17, 18*, *21*, *24–26*, *33*, *50*, *53*, *72*

environment, *26*, *30*
extractParam, 30

function, *10*, *16*, *28*
getAllNodesProperties
        *(*getNodeProperties*)*, 32

getDefaultAprioriIds, *21*, 31
getNextTimeSteps, 32
getNodeProperties, 32
getNodeRanking, *21*, 34, *73*
getNoSD_Ids *(*getSD_Ids*)*, 35
getSD_Ids, 35
graphics::legend, *50*
GRiwrm, *24*, *36*
GRiwrm *(*CreateGRiwrm*)*, 16
GRiwrmInputsModel, *35, 36*, *74*
GRiwrmOutputsModel, *31*
GRiwrmRunOptions, *61*

IniStates, *28*
integer, *30*, *32*
isNodeDownstream, 35
isNodeUpstream *(*isNodeDownstream*)*, 35

legend, *50*
list, *3, 4*, *7*, *11*, *17*, *21, 22*, *25, 26*, *28–31*, *33*, *39*, *43*, *52, 53*, *61*, *72*
logical, *9*, *17*, *25*, *33*, *35–37*, *39*, *51*, *74*

matplot, *50*
matrix, *9*, *16*, *21*, *24–26*, *42*
merge.GRiwrmOutputsModel *(*merge.OutputsModel*)*, 36
merge.OutputsModel, 36
mermaid, *37*, 37, *39*
mermaid_gen_link, *37*
mermaid_gen_link *(*mermaid*)*, 37

NA, *18*
NULL, *50*
numeric, *9–11*, *16–18*, *21*, *24–26*, *29*, *31*, *43*, *50*, *52, 53*, *65*, *74*

par, *50*
plot, *42*
plot.default, *50*
plot.GRiwrm, 38
plot.GRiwrmOutputsModel, *3*, 41
plot.GRiwrmOutputsModel(), *4*
plot.mermaid *(*mermaid*)*, 37
plot.OutputsModelReservoir, 43
plot.Qm3s, *43*, 49, *53*
POSIXt, *24*, *50*

reduceGRiwrm, 51
RunModel, 52
RunModel.GR, 52
RunModel.GRiwrmInputsModel, *3, 4*, *29–32*, *36*, *42*, *50*, 53, *60, 61*
RunModel.GRiwrmInputsModel(), *26*, *29*
RunModel.GRiwrmOutputsModel, *30*, *32*, *36*, 59
RunModel.InputsModel, *53*, 60
RunModel.Supervisor, *16*, *29–32*, *36*, *42*, *50*, 61
RunModel.Supervisor(), *16*, *26*, *30*
RunModel_CemaNeigeGR4J, *53*, *61*
RunModel_GR4J, *53*, *61*
RunModel_Reservoir, *7*, *17*, *25*, *29*, *31*, *42, 43*, *50*, *53*, 64
RunModel_Reservoir(), *26*

Severn, 72
sort.GRiwrm, 73
Supervisor, *61*

transferGRparams, *11*, 73

vector, *11*, *17*, *21*, *25, 26*, *31*, *34*, *53*, *74*