Package 'airGRiwrm'

May 7, 2026

Title 'airGR' Integrated Water Resource Management

Version 0.7.0

Date 2024-09-22

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 (>= 1.7.0), R (>= 3.5)

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

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

VignetteBuilder knitr

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.3.2

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 2024-09-22 18:20:02 UTC

Contents

as.Qm3s . . . 2
Calibration.GRiwrmInputsModel . . . 3
ConvertMeteoSD . . . 4
CreateCalibOptions.GRiwrmInputsModel . . . 5
CreateController . . . 7
CreateGRiwrm . . . 8
CreateInputsCrit.GRiwrmInputsModel . . . 11
CreateInputsModel . . . 13
CreateInputsModel.GRiwrm . . . 13
CreateRunOptions.GRiwrmInputsModel . . . 16
CreateSupervisor . . . 17
extractParam . . . 18
getNodeProperties . . . 18
getNodeRanking . . . 20
getNoSD_Ids . . . 21
getSD_Ids . . . 21
isNodeDownstream . . . 22
mermaid . . . 22
plot.GRiwrm . . . 24
plot.GRiwrmOutputsModel . . . 27
plot.OutputsModelReservoir . . . 28
plot.Qm3s . . . 30
reduceGRiwrm . . . 31
RunModel . . . 32
RunModel.GR . . . 33
RunModel.GRiwrmInputsModel . . . 33
RunModel.InputsModel . . . 38
RunModel.Supervisor . . . 39
RunModel_Reservoir . . . 42
Severn . . . 45
sort.GRiwrm . . . 46
transferGRparams . . . 46

Index **48**

`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,
  ...
)

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

Calibration(InputsModel, ...)

## S3 method for class 'Ungauged'
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`
...	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*, a `InputsCrit` argument of class *InputsCrit* and a `CalibOptions` of class *CalibOptions*
- a `InputsModel` argument of class *GRiwrmInputsModel* must be followed by a `RunOptions` argument of class *GRiwrmRunOptions*, a `InputsCrit` argument of class *GRiwrmInputsCrit* and a `CalibOptions` of class *GRiwrmCalibOptions*

See the vignettes for examples.

Value

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

- a `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()`

`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 catchment (with an *InputsModel* object) or for a network (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	a numeric vector as for airGR::CreateCalibOptions, or a list giving the values of non-optimized parameters (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-catchments 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. Parameters are adapted depending on the use of the routing model and the CemaNeige model on each node. If FixedParam is a list of numeric, 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.

Value

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

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

See Also

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

`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 (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 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
## Not run:
plot(griwrm_basic)
```

```
## End(Not run)

################################################
# 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
## Not run:
plot(griwrm_severn)

## End(Not run)

#####################################################################
# 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
## Not run:
plot(griwrm_ungauged)

## End(Not run)

#############################################################
# 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
## Not run:
plot(griwrm_div, orientation = "TB")

## End(Not run)

# It's also possible to custom the diagram's look with mermaid directives
# (See details in plot.GRiwrm help topic)
## Not run:
plot(
  griwrm_div,
 header = "%%{init: {'flowchart': {'nodeSpacing': 30, 'rankSpacing': 30, 'curve': 'linear'}}}%%"
)

## End(Not run)
```

`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_NSE,
  RunOptions,
  Obs,
  AprioriIds = NULL,
  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 of character 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-catchment is used for providing a priori parameters. Its format is as follows: `AprioriIds <- c("Downstream sub-catchment 1" = "A priori upstream sub-catchment 1", ...)` where the quoted strings are the ids of the sub-catchments. The node 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.

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()

`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 a* **airGRiwrm** *network*

Description

Creation of an InputsModel object for a **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 = 5,
  IsHyst = FALSE,
  ...
)
```

Arguments

x	[GRiwrm object] diagram of the semi-distributed model (See CreateGRiwrm)
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 to let 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 release flows by nodes using the model `RunModel_Reservoir` [m3 per time step]
PrecipScale	(optional) named vector of logical 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)

TempMean	(optional) matrix or data.frame of time series of mean air temperature [°C], required to create the CemaNeige module inputs
TempMin	(optional) matrix or data.frame of time series of minimum air temperature [°C], possibly used to create the CemaNeige module inputs
TempMax	(optional) matrix or data.frame of time series of maximum air temperature [°C], possibly used to create the CemaNeige module inputs
ZInputs	(optional) named vector of numeric giving the mean elevation of the Precip and Temp series (before extrapolation) [m], possibly used to create the CemaNeige module input
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
NLayers	(optional) named vector of numeric integer giving the number of elevation layers requested -, required to create CemaNeige module inputs, default=5
IsHyst	logical boolean indicating if the hysteresis version of CemaNeige is used. See details of airGR::CreateRunOptions.
...	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-catchment 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 input.

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

For 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".

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()`

`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, ...)

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, ...)
```

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) numeric object or list of numeric object of class *IniStates*, see airGR::CreateIniStates for details
`...`	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

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`	numeric 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` 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)
- some internal state variables updated during simulation (`ts.index`, `ts.previous`, `ts.date`, `ts.index0`, `controller.id`)

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 and RunModel.Supervisor.

Usage

```
extractParam(x)
```

Arguments

`x`	A *GRiwrmOutputsModel* object returned by Calibration.GRiwrmInputsModel

Details

See vignettes and example of RunModel_Reservoir for examples of use.

Value

A named list of numeric vector containing the calibrated parameters of each modeled node.

See Also

Calibration, RunModel.GRiwrmInputsModel, RunModel.Supervisor

`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()`

getNoSD_Ids *Function to obtain the ID of sub-basins not using SD model*

Description

Function to obtain the ID of sub-basins not using SD model

Usage

```
getNoSD_Ids(InputsModel, include_diversion = TRUE)
```

Arguments

InputsModel	[GRiwrmInputsModel object]
include_diversion	logical for including diversion nodes

Value

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

getSD_Ids *Function to obtain the ID of sub-basins using SD model*

Description

Function to obtain the ID of sub-basins using SD model

Usage

```
getSD_Ids(InputsModel, add_diversions = FALSE)
```

Arguments

InputsModel	[GRiwrmInputsModel object]
add_diversions	logical for adding upstream nodes with diversion

Value

character IDs of the sub-basins using SD model

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 object] (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

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),
  link = mermaid_gen_link(diagram, theme = theme, format = format)
)

mermaid_gen_link(
  diagram,
  theme = "default",
  format = "png",
  server = "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.
`link`	Link generated by mermaid_gen_link
`server`	URL of the server used to generate the 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).

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

```
## Not run:
diagram <- "flowchart LR\n  A --> B"
mermaid_gen_link(diagram)
f <- mermaid(diagram)
f

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

# Clean temporary folder
unlink(f)

## End(Not run)

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 of a diagram representing the network structure of a GRiwrm object

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.

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
## Not run:
plot(griwrm_basic)

## End(Not run)

#################################################
# 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
## Not run:
plot(griwrm_severn)

## End(Not run)

#####################################################################
# 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
## Not run:
plot(griwrm_ungauged)

## End(Not run)

#############################################################
# 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
## Not run:
plot(griwrm_div, orientation = "TB")

## End(Not run)

# It's also possible to custom the diagram's look with mermaid directives
# (See details in plot.GRiwrm help topic)
## Not run:
plot(
  griwrm_div,
 header = "%%{init: {'flowchart': {'nodeSpacing': 30, 'rankSpacing': 30, 'curve': 'linear'}}}%%"
)

## End(Not run)
```

`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, 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)
`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.

Value

list of plots.

`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, ...)
```

Arguments

`x`	Object returned by RunModel_Reservoir
`Qobs`	(optional) numeric time series of observed released flow [m3/time step]
`...`	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)
## Not run:
plot(griwrm)

## End(Not run)

# 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 median of the natural flow
# The value is in m3 by time step (day)
Qrelease <- median(BasinObs$Qls, na.rm = TRUE) / 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,
                                 Qinf = 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 GRiwmRunOptions object
RunOptions <- CreateRunOptions(
  InputsModel,
  IndPeriod_Run = Ind_Run,
  IndPeriod_WarmUp = seq.int(Ind_Run[1] - 365, length.out = 365)
)

# Initial states of the reservoir can be provided by the user
# For example for starting with an empty reservoir...
RunOptions[["Reservoir"]]$IniStates <- 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
CalibOptions <- CreateCalibOptions(InputsModel)
```

```
# Parameters of RunModel_Reservoir must be fixed
CalibOptions[["Reservoir"]]$FixedParam <- c(Vmax = 30E6, 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)
# N.B. "Observed releases" should be considered as "Target releases" here

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

`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

Function which performs a single model run with the provided function over the selected period.

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 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, ...)
```

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
...	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)
## Not run:
plot(griwrm)

## End(Not run)

# 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
## Not run:
plot(g_severn)

## End(Not run)

# 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
## Not run:
plot(g_div)

## End(Not run)

# 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.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 GRiwrmInputsModel object*

Description

RunModel function for a GRiwrmInputsModel object

Usage

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

Arguments

x	[object of class Supervisor] see CreateSupervisor for details
RunOptions	[object of class *GRiwrmRunOptions*] see [CreateRunOptions.GRiwrm] for details
Param	list parameter values. The list item names are the IDs of the sub-basins. Each item is a vector of numerical parameters
...	Further arguments for compatibility with S3 methods

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"))
## Not run:
plot(griwrm)

## End(Not run)

# 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
Qinf <- data.frame(
  Dam = BasinsObs$`54095`$discharge_spec * griwrm$area[griwrm$id == "54095"] * 1E3
)

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

# 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
oldpar <- par(mfrow=c(2,1),
              mar = c(2,3.3,1.2,0.5),
              mgp = c(2,1,0))
plot(attr(OM_dam, "Qm3s")[, c("DatesR", "54095", "Dam", "54057")],
     ylim = c(0, 200))
Vres <- as.Qm3s(DatesR = OM_dam$Dam$DatesR,
                   "Simulated volume" = OM_dam$Dam$Vsim / 1E6)
plot(Vres,
     main = "Simulated reservoir storage",
     ylab = expression("Storage (Mm" ^ "3" * ")"))
par(oldpar)
```

`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

The simulated flow corresponds to the released flow except when the reservoir is empty (release flow is limited) or full (release flow is completed by inflows excess).

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 following instruction after the call to CreateRunOptions.GRiwrmInputsModel:

`RunOptions[[id_of_the_reservoir]]$IniStates <- c("Reservoir.V" = initial_volume_m3)`

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)
## Not run:
```

```
plot(griwrm)

## End(Not run)

# 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 median of the natural flow
# The value is in m3 by time step (day)
Qrelease <- median(BasinObs$Qls, na.rm = TRUE) / 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,
                                 Qinf = 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 GRiwmRunOptions object
RunOptions <- CreateRunOptions(
  InputsModel,
  IndPeriod_Run = Ind_Run,
  IndPeriod_WarmUp = seq.int(Ind_Run[1] - 365, length.out = 365)
)

# Initial states of the reservoir can be provided by the user
# For example for starting with an empty reservoir...
RunOptions[["Reservoir"]]$IniStates <- 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
CalibOptions <- CreateCalibOptions(InputsModel)

# Parameters of RunModel_Reservoir must be fixed
CalibOptions[["Reservoir"]]$FixedParam <- c(Vmax = 30E6, 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)
# N.B. "Observed releases" should be considered as "Target releases" here

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

`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 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 hydrometeorological 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

Usage

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

Arguments

InputsModel	A *GRiwrmInputsModel* object (See CreateInputsModel.GRiwrm)
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
- the transformation of the X4 parameters of GR models

Value

A numeric vector with transferred parameters

Index

∗ **datasets**
    Severn, 45
-, *15*

airGR::Calibration, *3, 4*
airGR::CreateCalibOptions, *6*
airGR::CreateIniStates, *16*
airGR::CreateInputsCrit, *12*
airGR::CreateInputsCrit_Lavenne, *12*
airGR::CreateInputsModel, *13*, *15, 16*, *39*
airGR::CreateInputsModel(), *13*
airGR::CreateRunOptions, *15–17*, *33*
airGR::ErrorCrit_NSE, *12*
airGR::ErrorCrit_RMSE, *12*
airGR::plot.OutputsModel, *27*
airGR::RunModel, *33*, *38–40*, *43*
airGR::RunModel_GR4J, *39*
as.Qm3s, 2
attr, *34*

Calibration, *18*
Calibration *(*Calibration.GRiwrmInputsModel*)*, 3
Calibration(), *6*, *13*
Calibration.GRiwrmInputsModel, 3, *18*
character, *5–9*, *12*, *16*, *18–23*, *25*, *27*, *31*, *47*
colnames, *5*
ConvertMeteoSD, 4
CreateCalibOptions, *3*, *43*
CreateCalibOptions *(*CreateCalibOptions.GRiwrmInputsModel*)*, *5*
CreateCalibOptions(), *4*, *13*
CreateCalibOptions.GRiwrmInputsModel, *5*
CreateController, 7, *17*
CreateController(), *17*
CreateGRiwrm, *5*, 8, *12*, *14*, *17, 18*, *20*, *22*, *25*, *27*, *32*, *34*, *46*
CreateGRiwrm(), *4–6*, *13*, *15*, *17*, *19, 20*, *25*, *34*
CreateInputsCrit, *3*
CreateInputsCrit *(*CreateInputsCrit.GRiwrmInputsModel*)*, 11
CreateInputsCrit(), *4*, *6*
CreateInputsCrit.GRiwrmInputsModel, 11
CreateInputsModel, *3*, *6*, *11*, 13, *16*, *32, 33*, *39*, *42*
CreateInputsModel.GRiwrm, *6*, *13*, 13, *17*, *22*, *34*, *39*, *42, 43*, *47*
CreateInputsModel.GRiwrm(), *4–6*, *13*, *17*, *34*
CreateRunOptions, *3*, *12*, *39*, *42*
CreateRunOptions *(*CreateRunOptions.GRiwrmInputsModel*)*, 16
CreateRunOptions(), *6*, *13*, *15*, *34*
CreateRunOptions.GRiwrmInputsModel, 16, *34*, *43*
CreateSupervisor, *7*, 17, *39*
CreateSupervisor(), *7*

data.frame, *2, 3*, *5*, *8, 9*, *12*, *14, 15*, *19*, *31*, *34*, *45*

environment, *17*
extractParam, 18

function, *6, 7*, *16*
getAllNodesProperties *(*getNodeProperties*)*, 18
getNodeProperties, 18
getNodeRanking, *12*, 20, *46*
getNoSD_Ids, 21
getSD_Ids, 21
graphics::legend, *31*
GRiwrm *(*CreateGRiwrm*)*, 8

isNodeDownstream, 22
isNodeUpstream *(*isNodeDownstream*)*, 22

legend, *31*
list, *4*, *6*, *8*, *12*, *16–19*, *25*, *28*, *32*, *34*, *39*, *45*
logical, *5*, *8*, *14, 15*, *19*, *21–23*, *25*, *32*, *47*

matplot, *31*
matrix, *5*, *7*, *12*, *14, 15*, *27*
mermaid, 22, *23*, *25*
mermaid_gen_link, *23*
mermaid_gen_link *(*mermaid*)*, 22

NA, *9*
NULL, *31*
numeric, *5–9*, *12*, *14–18*, *28*, *31*, *33, 34*, *42*, *47*

par, *31*
plot, *27*
plot.default, *31*
plot.GRiwrm, 24
plot.GRiwrmOutputsModel, 27
plot.mermaid *(*mermaid*)*, 22
plot.OutputsModelReservoir, 28
plot.Qm3s, *28*, 30, *34*
POSIXct, *17*, *45*
POSIXt, *14*, *31*

reduceGRiwrm, 31
RunModel, 32
RunModel.GR, 33
RunModel.GRiwrmInputsModel, *15, 16*, *18*, *27, 28*, *31*, 33
RunModel.GRiwrmInputsModel(), *15*, *17*
RunModel.InputsModel, *34*, 38
RunModel.Supervisor, *7*, *15–18*, *28*, *31*, 39
RunModel.Supervisor(), *7*, *17*
RunModel_CemaNeigeGR4J, *33*, *39*
RunModel_GR4J, *33*, *39*
RunModel_Reservoir, *9*, *15, 16*, *18*, *28*, *31*, 42

Severn, 45
sort.GRiwrm, 46

transferGRparams, 46

vector, *6*, *8*, *12*, *14, 15*, *18*, *20*, *34*, *47*