Package: airGRccia (via r-universe)

May 17, 2026

Title Climate Change Impact and Adaptation with airGRiwrm

Version 0.1.0.9000

Description This package provides tools for studying climate Change Impact and Adaptation on a watershed modeled with airGRiwrm.

URL https://inrae.r-universe.dev/airGRccia, https://airgriwrm.pages.forge.inrae.fr/airgrccia

BugReports https://forge.inrae.fr/airgriwrm/airgrccia/-/issues

License AGPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends airGRiwrm

Suggests airGR, happign, knitr, lubridate, mapview, openxlsx2, pkgload, rmarkdown, rprojroot, scales, testthat (>= 3.0.0), tidyr, tidyverse, tmap, TSstudio

VignetteBuilder knitr

Imports cli, config, dplyr, future, future.apply, ggplot2, httr, hubeau, logger, methods, parallelly, lobstr, purrr, RADIS, rappdirs, readr, rvest, sf, stars, stats, stringr, terra, units, urltools, utils, whitebox, xfun

Config/testthat/edition 3

Additional_repositories https://inrae.r-universe.dev

Config/pak/sysreqs libabsl-dev cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev libgdal-dev gdal-bin libgeos-dev make libharfbuzz-dev libicu-dev libpng-dev libuv1-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev libx11-dev zlib1g-dev

Repository https://inrae.r-universe.dev

Date/Publication 2026-03-18 16:16:19 UTC

RemoteUrl https://forge.inrae.fr/airgriwrm/airgrccia

RemoteRef main

RemoteSha d9b4134f46abd76107e95c82850fab8a155c3fb7

Contents

`calc_hypso` *Calculate hypsometric curves for sub-basins*

Description

This function computes the hypsometric curve for each sub-basin based on the provided DEM. The hypsometric curve represents the distribution of elevation within a basin and is mandatory for using the CemaNeige module in airGR.

Usage

```
calc_hypso(sb, dem)
```

Arguments

`sb`	An `sf` object representing the sub-basins with an "id" field (See output of extract_sub_basins).
`dem`	A raster object (e.g., `SpatRaster`, `RasterLayer`, or file path) representing the Digital Elevation Model (DEM) (See output of get_ign_dem).

Value

A matrix where each column corresponds to a sub-basin and each row represents the elevation quantiles (from 0% to 100%). This corresponds to the format required by airGRiwrm::CreateInputsModel.GRiwrm for the `HypsoData` argument.

`CreateBasinsObs` *Create a* `BasinsObs` *object (Hydroclimatic time series)*

Description

Create a `BasinsObs` object (Hydroclimatic time series)

Usage

```
CreateBasinsObs(
  s_sb,
  ids,
  Qobs = NULL,
  Qinf = NULL,
  Qrelease = NULL,
  measures = c("Precip", "PotEvap", "TempMean")
)
```

Arguments

`s_sb`	Climate data (See get_climate_safran)
`ids`	Vector of sub-basin ids
`Qobs`	Observed flows (See get_hubeau_flows)
`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
`Qrelease`	(optional) matrix or data.frame of numeric containing release flows by nodes using the model `RunModel_Reservoir` [m3 per time step]
`measures`	

Value

A list containing an item`DatesR`, vector of dates corresponding to the time dimension of the data, following by data.frames for each measure including climate data ("Precip", "PotEvap", "TempMean"), and observed flows ("Q").

`extract_reaches` *Extract River Reaches by Sub-basin from a DEM*

Description

This function extracts individual river reaches by tracing flow paths from each sub-basin outlet to the next downstream outlet using a D8 flow direction raster and a stream network.

Usage

```
extract_reaches(
  sb,
  d8fd = attr(sb, "files")["d8"],
  streams = attr(sb, "files")["streams"],
  max_tries = 1e+05,
  eps = 1,
  max_step_back = 10,
  max_azimut = 4,
  max_buffer = 3,
  dTolerance = 2 * terra::res(d8fd),
  future_plan = future::multisession,
  future_workers = get_future_workers(),
  filename = getCachePath(sb, "reaches", "gpkg"),
  overwrite = FALSE
)
```

Arguments

`sb`	An `sf` object representing the perimeter of sub-basins. Typically the output of `extract_sub_basins()`.
`d8fd`	A `SpatRaster` or character path to the file containing D8 flow directions. Can also be obtained from `extract_sub_basins()`.
`streams`	An `sf` object or character path to the file containing the stream network (e.g., output of `get_ign_streams()`).
`max_tries`	integer. Maximum cells tested for retrieving a reach.
`eps`	numeric. Minimum acceptable distance between the downstream reach and the downstream outlet in cells.
`max_step_back`	integer Maximum contiguous back steps in stream cell research
`max_azimut`	integer Maximum change of orientation in cell research (See details)
`max_buffer`	integer Maximum number of cells matching around stream cells
`dTolerance`	numeric. The tolerance for simplifying the reach geometry (Used in sf::st_simplify).
`future_plan`	function One of available future plan: future::sequential, future::multisession, future::multicore, future::cluster

`future_workers`
: integer Number of workers to use for parallel processing.

`filename`
: character Optional. If provided, the streams geometry will be saved to this file and can be used as cache if `overwrite = FALSE`.

`overwrite`
: logical Whether to overwrite the cached file if it exists.

Details

Type `logger::log_level(logger::DEBUG)` or `logger::log_level(logger::TRACE)` before running the function for getting details on stream cells search. In this mode, maps zooming on issue zones are recorded in the working directory (names "`debug_extract_reaches_bvi*_step*_row*_`

`max_azimut` correspond to the maximum change of orientation when searching the next stream cell (0, for no change; 2 for 45°; 4 for 90°, 6 for 135° and 7 for 90°).

Value

An `sf` object containing one river reach per sub-basin, from the sub-basin outlet to the next downstream outlet.

`extract_sub_basins` *Extract sub-basins from DEM*

Description

Extract sub-basins from DEM

Usage

```
extract_sub_basins(
  outlets,
  dem = NULL,
  streams,
  crs = NULL,
  threshold = 1000,
  snap_dist = 100,
  area_tolerance = 0.05,
  max_radius = 50,
  filename = TRUE,
  overwrite = FALSE,
  verbose_mode = NULL
)

watershed_get_from_outlet(
  i,
  outlets,
  file_d8,
  r_flow_accum,
```

```
  crs,
  area_tolerance = 0.05,
  max_radius = 50,
  verbose_mode = NULL,
  overwrite = FALSE
)

find_near_target(
  r_flow_accum,
  row0,
  col0,
  target,
  area_tolerance,
  max_radius = 5
)
```

Arguments

outlets	data.frame with columns "id", "x", and "y" and optionnaly "area" containing respectively the outlet ID, X and Y coordinates (in the CRS of the DEM) and the a priori area of the sub-basin (in square meters) (See Details)
dem	character containing the filename of the DEM raster, or a SpatRaster object
streams	A sf object containing the streams.
crs	The CRS used for outlets coordinates. If NULL, the CRS of dem is used
threshold	Minimum threshold of contributing area to extract streams (unit: square meters)
snap_dist	Maximum distance between outlet and a stream vector for relocating outlet in the stream (in meters)
area_tolerance	Maximum relative error between the a priori area and the computed area of the sub-basin
max_radius	Maximum search radius (in number of cells) to find a closer point
filename	character Optional. If provided, the sub-basins geometry will be saved to this file and can be used as cache if overwrite = FALSE. If TRUE (default), a filename will be automatically generated with getCachePath based on the input parameters.
overwrite	logical Whether to overwrite the cached file if it exists.
verbose_mode	Sets verbose mode. If verbose mode is FALSE, tools will not print output messages. Default: NULL will use the value in WhiteboxTools settings, see wbt_verbose() for details.
i	integer row number to process in outlets
file_d8	Path of raster D8
r_flow_accum	Raster of flow accumulation

`row0`, `col0`	respectively row and column number of the current coordinates of the outlet in the raster
`target`	Target area to reach (in square meters)

Details

`extract_sub_basins` delineate the watershed from the DEM for each outlet, then it computes the sub-basins contour by remove overlapping basin areas and compute which sub-basin is downstream of which other sub-basin.

`watershed_get_from_outlet` delineate one basin given outlet coordinates. One can provide an a priori area in `outlets$area` which is used to control the area of the resulting polygon. If the resulting area is outside the tolerance (`area_tolerance`), the algorithm search for a closer point (at a maximum distance of `max_radius` cells of the DEM raster) on the stream network and delineate again the basin. This is repeated until the area is within the tolerance or `max_tries` is reached.

Value

`extract_sub_basins` returns a `sf` object with the polygons representing the sub-basins and various attributes detailling the sub-basin up-down order, total basin and sub-basin areas, coordinates of outlets after move.

Functions

- `extract_sub_basins()`: extracts sub-basins from a DEM using whitebox tools.
- `watershed_get_from_outlet()`: Extract one watershed for one outlet
- `find_near_target()`: Find a cell near (row0, col0) in the flow accumulation raster cells with a value close to a target area.

`get_climate_safran` *Extract and Aggregate SIM2 Daily Climate Data from RADIS*

Description

This function retrieves selected daily climate variables from the SIM2 dataset via the `RADIS::get_sim2_daily()` function, renames the fields for compatibility with GR models (e.g., `airGR`), and optionally aggregates multiple related source fields (e.g., solid and liquid precipitation).

Usage

```
get_climate_safran(
  x,
  fields = list(Precip = c("PRENEI_Q", "PRELIQ_Q"), PotEvap = "ETP_Q", TempMean = "T_Q"),
  date_start,
  date_end,
  overlay_mode = "aggregate",
```

```
  filename = getCachePath(list(x, fields, date_start, date_end, overlay_mode),
    "climate_safran", ifelse(overlay_mode %in% c("intersect_geometry", "aggregate"),
    "nc", "RDS")),
  overwrite = FALSE,
  ...
)
```

Arguments

`x`	An `sf` object or geometry supported by `RADIS::get_sim2_daily()`, typically representing a spatial feature (e.g., watershed or point).
`fields`	A named list where each element corresponds to a GR-compatible variable name (e.g., `Precip`, `PotEvap`, `TempMean`) and contains one or more SIM2 field names to be retrieved and optionally summed.
`date_start`, `date_end`	A Date object or a character in format `"%Y-%m-%d"` specifying start and end dates of the extraction period.
`overlay_mode`	A character string passed to `RADIS::get_sim2_daily()`, typically `"aggregate"` for spatial mean over polygons or `"raw"` for full raster.
`filename`	If not `NULL`, path of cached data (See details).
`overwrite`	logical Whether to overwrite the cached file if it exists.
`...`	Additional arguments passed to `RADIS::get_sim2_daily()`.

Details

Default value for `filename` depends on the parameter `overlay_mode` which cache the data as a netCDF file for `overlay_mode = "intersect_geometry"` and in RDS format for `overlay_mode = "intersect_geometry"`.

Value

A named list where each element is a 3D array (or similar structure) containing the extracted and aggregated climate variables, named using the keys of `fields` (e.g., `"Precip"`, `"PotEvap"`, etc.).

See Also

`RADIS::get_sim2_daily()`

Examples

```
## Not run:
library(sf)
# Example with a watershed polygon
basin <- sf::st_read("my_watershed.gpkg")
data <- get_climate_safran(
  x = basin,
  date_start = as.Date("2000-01-01"),
  date_end = as.Date("2005-12-31")
```

```
)

## End(Not run)
```

`get_future_workers` *Compute number of workers given available memory and CPUs*

Description

This function computes the number of workers to use for parallel processing taking into account both constraints that are available RAM and CPUs.

Usage

```
get_future_workers(mem_share = 0.8, cpu_share = 0.8)
```

Arguments

`mem_share`	numeric Share of RAM to use given total RAM and currently used RAM by this R session.
`cpu_share`	numeric Share of CPUs to use given total available CPUs.

Value

integer Number of workers to use.

Examples

```
get_future_workers()
```

`get_hubeau_flows` *Retrieve river flow observation from the Hydrometrie Hub'Eau API*

Description

Flow observations are retrieved with hubeau::get_hydrometrie_obs_elab and formatted for **airGRiwrm** usage. Station identifiers are used as `code_entite` filter parameter on the Hydrometrie Hub"Eau API and can be either a site (8 characters-length) or a measuring station (10 characters-length) code.

Usage

```
get_hubeau_flows(x, ...)

## Default S3 method:
get_hubeau_flows(x, ...)

## S3 method for class 'SubBasins'
get_hubeau_flows(x, ...)

## S3 method for class 'character'
get_hubeau_flows(
  x,
  date_start,
  date_end,
  grandeur_hydro_elab = "QmnJ",
  max_days = 10000,
  max_retries = 3,
  wait_seconds = 2,
  path_cache = getCachePath(list(date_start, date_end, grandeur_hydro_elab),
    "hubeau_flows"),
  overwrite = FALSE,
  ...
)
```

Arguments

`x`	Either a character vector of station codes or a `SubBasins` object.
`...`	Parameters passed to methods or to hubeau::get_hydrometrie_obs_elab
`date_start`	Date of starting period
`date_end`	Date of ending period
`grandeur_hydro_elab`	Processed hydrometric variable type ("HIXM", "HIXnJ", "QINM", "QINnJ", "QixM", "QIXnJ", "QmM", or "QmnJ")
`max_days`	Maximum number of days in one API call
`max_retries`	Maximum number of retries for failed API calls
`wait_seconds`	Initial waiting time between retries (will be doubled at each retry)
`path_cache`	If not `NULL`, folder path of cached data
`overwrite`	logical Whether to overwrite the cached file if it exists.

Details

Hub'Eau APIs are currently limited to 20000 rows so this function cut the time period in `max_days` chunks for the API calls.

Value

A data.frame with a first column `DatesR` containing the dates and one column by station containing observed flows in m3/s. The returned object contains an attribute "codes_site_station" listing the correspondance between `code_entite` and station codes.

`get_ign_dem` *Extract DEM from IGN API*

Description

This function download DEM from IGN API and perform correction on outliers.

Usage

```
get_ign_dem(x, ...)

## S3 method for class 'sf'
get_ign_dem(x, ...)

## S3 method for class 'bbox'
get_ign_dem(x, ...)

## S3 method for class 'sfc'
get_ign_dem(
  x,
  layer = "ELEVATION.ELEVATIONGRIDCOVERAGE",
  res = 25,
  crs = sf::st_crs(x),
  neg_outlier_threshold = 0,
  ...
)
```

Arguments

`x`	A bounding box, a simple feature collection or a simple feature geometry.
`...`	Further parameters passed to methods and to happign::get_wms_raster
`layer`	`character`; layer name obtained from `get_layers_metadata("wms-r")` or the IGN website.
`res`	`numeric`; resolution specified in the units of the coordinate system (e.g., meters for EPSG:2154, degrees for EPSG:4326). See details for more information.
`crs`	`numeric`, `character`, or object of class `sf` or `sfc`; defaults to EPSG:2154. See `sf::st_crs()` for more details.
`neg_outlier_threshold`	Threshold under which data is rejected as outlier and replaced by the mean of neighbour cells

Details

Default value of `layer` correspond to the BD Alti with 25 m of resolution. Available DEM on IGN API can be explored with the instruction: `happign::get_layers_metadata("wms-r", "altimetrie")`.

Value

A `SpatRaster` object containing the DEM data.

Examples

```
# Basic usage with default DEM
bbox <- c(xmin = 325522.8, ymin = 6250949.2, xmax = 347137.1, ymax = 6270285.1)
dem <- get_ign_dem(sf::st_bbox(bbox, crs = 2154))
dem
terra::plot(dem)
```

---

get_ign_streams *Get streams from IGN*

Description

This function retrieves hydrographic streams from the IGN WFS service and filter them based on specified criteria.

Usage

```
get_ign_streams(x, ...)

## S3 method for class 'sf'
get_ign_streams(x, ...)

## S3 method for class 'bbox'
get_ign_streams(x, ...)

## S3 method for class 'sfc'
get_ign_streams(
  x,
  layer = "BDCARTO_V5:troncon_hydrographique",
  crs = sf::st_crs(x),
  filters = NULL,
  filename = NULL,
  overwrite = FALSE,
  ...
)
```

Arguments

x	A bounding box, a simple feature collection or a simple feature geometry.
...	Further parameters passed to methods and to happign::get_wfs
layer	character; name of the WFS layer. Must correspond to a layer available on the IGN WFS service (see get_layers_metadata()).
crs	The CRS to use for the output streams (the CRS of x is used by default).
filters	A list of filters to apply to the streams data (See details)
filename	character Optional. If provided, the streams geometry will be saved to this file and can be used as cache if overwrite = FALSE.
overwrite	logical Whether to overwrite the cached file if it exists.

Details

List of filters is structured as follow: name correspond to the column names in the streams data, and values can be:

- A character vector to filter by specific values
- A function that takes a vector and returns a logical vector (e.g., not.is.na) not.is.na is equivalent of !is.na().

Value

A sf object containing the streams data.

Examples

```
bbox <- c(xmin = 325522.8, ymin = 6250949.2, xmax = 347137.1, ymax = 6270285.1)
# Retrieve only river streams that have a name and of specific nature
streams <- get_ign_streams(
  sf::st_bbox(bbox, crs = 2154),
  filters = list(
    nature = c("Ecoulement naturel", "Ecoulement canalis\u00E9"),
    cpx_toponyme_de_cours_d_eau = not.is.na
  )
)
streams
mapview::mapview(streams)
```

---

get_ram *Get the amount of RAM*

Description

Attempt to extract the amount of RAM on the current machine. This is OS specific:

- Linux: `proc/meminfo`
- Apple: `system_profiler -detailLevel mini`
- Windows: First tries `grep MemTotal /proc/meminfo` then falls back to `wmic MemoryChip get Capacity`
- Solaris: `prtconf`

A value of `NA` is return if it isn't possible to determine the amount of RAM.

Usage

```
get_ram()
```

References

This function and associated codes were taken from the **benchmarkme** package.

Examples

```
# Return (and pretty print) the amount of RAM
get_ram()
# In raw format
str(get_ram)
```

---

get_safran_var_units *Get units of the variables available in SAFRAN database*

Description

Get units of the variables available in SAFRAN database

Usage

```
get_safran_var_units()
```

Value

named character vector containing units for each climatic variable

Examples

```
get_safran_var_units()
```

`getCachePath` *Get cache path*

Description

Build a path in a cache directory and a file name build from MD5 hash.

Usage

```
getCacheDir(
  path = Sys.getenv("CACHE_DIR_AIRGRCCIA", file.path(basepath, pkg)),
  pkg = utils::packageName(),
  basepath = rappdirs::user_cache_dir()
)

getCachePath(
  x,
  prefix = as.character(substitute(x)),
  fileext = NULL,
  cache_dir = getCacheDir()
)

clearCache(cache_dir = getCacheDir())
```

Arguments

`path`	Path to the cache directory (default from environment variable `CACHE_DIR_AIRGRCCIA` or package cache directory).
`pkg`	Package name (default current package)
`basepath`	Folder in which to add package cache directory (by default OS dependant cache folder)
`x`	Object to save (or property of object that can be used as a signature)
`prefix`	Optional prefix to add to the file name (default variable name)
`fileext`	Optional file extension to use for the file (e.g., "rds", "csv", etc.)
`cache_dir`	Path to the cache directory (default from `getCacheDir()`)

Value

The cache path.

Functions

- `getCacheDir()`: Returns cache path root and create the folder if it does not exist.
- `getCachePath()`: Returns cache path for a specific file.
- `clearCache()`: Clear the cache directory

Examples

```
getCacheDir()
path_cars <- getCachePath(cars, fileext = "rds")
path_cars
## Not run:
saveRDS(cars, path_cars)

## End(Not run)
```

---

hydro_get_station *Get info on hydrometric station*

Description

Get info on hydrometric station

Usage

```
hydro_get_station(x)

hydro_station_as_sf(x)
```

Arguments

x	code site

Value

A data.frame or a SpatVector. See hubeau::get_hydrometrie_sites.

Examples

```
hydro_get_station("Y2300020")
hydro_station_as_sf("Y2300020")
```

---

is_projected_crs *Check if the CRS is projected*

Description

Check if the CRS is projected

Usage

```
is_projected_crs(x)
```

Arguments

x	sf, sf geometry, or bbox.

Value

TRUE if the CRS is projected, FALSE otherwise.

Examples

```
# Define the bounding box coordinates in WGS84
bounding_box <- c(xmin = 3.211086, ymin = 43.24153, xmax = 3.904615, ymax = 44.28818)

# Create the bounding box using st_bbox
bbox <- sf::st_bbox(bounding_box, crs = 4326)

# Is it in projected CRS?
is_projected_crs(bbox)

# Convert to projected CRS
bbox_projected <- sf::st_transform(bbox, 2154)
is_projected_crs(bbox_projected)
```

---

not.is.na *Shortcut for* !is.na

Description

Shortcut for !is.na

Usage

```
not.is.na(x)
```

Arguments

x	an R object to be tested

Value

logical See is.na.

predict_uncertainty_calibration
Predictive Uncertainty functions

Description

Predictive Uncertainty functions

Usage

```
predict_uncertainty_calibration(
  Qobs,
  Qsim,
  bind_size = predict_uncertainty_bind_size(Qobs, obs_by_bind, min_bind_size),
  bind_step = 0.1 * bind_size,
  min_bind_size = 0.1,
  obs_by_bind = 1000,
  confidence_interval = 0.9,
  probs = c((1 - confidence_interval)/2, 1 - (1 - confidence_interval)/2)
)

predict_uncertainty(Qsim, uncertainty_table)

plot_uncertainty_time_series(
  Qsim_uncertain,
  DatesR,
  Qobs = NULL,
  bounds = names(Qsim_uncertain)[c(2, ncol(Qsim_uncertain))],
  rubbon_fill = "darkblue",
  rubbon_alpha = 0.2,
  line_colors = c(Simulated = "orangered", Observed = "black"),
  labels = list(title = "Predicted Discharge with Uncertainty Intervals", x = "Date", y =
    "Discharge (mm)", color = "Flow type")
)

plot_uncertainty_calibration(
  Qsim_uncertain,
  Qobs,
  bounds = names(Qsim_uncertain)[c(2, ncol(Qsim_uncertain))],
  rubbon_fill = "darkblue",
  rubbon_alpha = 0.2,
  outliers_quantile = 0.999,
  labels = list(title =
    "Relative Error vs. Simulated Discharge with Uncertainty Intervals", x =
    "Simulated Discharge (mm)", y = "Relative Error Qobs/Qsim (-)")
)
```

```
predict_uncertainty_bind_size(Qobs, obs_by_bind = 1000, min_bind_size = 0.1)

transfer_uncertainty(Qsim, uncertainty_table)
```

Arguments

`Qobs`	Observed discharge values (numeric vector).
`Qsim`	Simulated discharge values (numeric vector).
`bind_size`	Size of the moving bin as a fraction of the data range (default is 0.1).
`bind_step`	Step size for moving the bin as a fraction of the data range.
`min_bind_size`	Minimum bind size (default is 0.1).
`obs_by_bind`	Number of observations per bind (default is 1000).
`confidence_interval`	Confidence interval level (default is 0.90).
`probs`	Probabilities for quantile calculation (default is c(0.05, 0.5, 0.95)) for a 90% prediction interval.
`uncertainty_table`	Data frame returned by `predict_uncertainty_calibration`.
`Qsim_uncertain`	Data frame returned by `predict_uncertainty`, containing simulated discharge and uncertainty quantiles.
`DatesR`	Dates corresponding to the discharge values.
`bounds`	Names of the columns in `QSim_uncertain` representing the uncertainty bounds (default is the first and last error quantile).
`rubbon_fill`	Fill color for the uncertainty ribbon.
`rubbon_alpha`	Alpha transparency for the uncertainty ribbon.
`line_colors`	Named vector specifying colors for the simulated and observed discharge lines.
`labels`	List of labels for the plot (title, x, y, ...) used in `ggplot2::labs()`.
`outliers_quantile`	Quantile threshold to filter out extreme relative errors (default is 0.999).

Details

This method is based on the approach described in Bourgin (2014) pages 201-203 with a variant consisting in moving a bin along the distribution of simulated discharge instead of using fixed discharge classes. Morever, the size of the bin is defined with respect to the number of observations which allows to refine the bin size when a large dataset is used.

References:

Bourgin, François. 2014. « Comment quantifier l'incertitude prédictive en modélisation hydrologique ? : Travail exploratoire sur un grand échantillon de bassins versants ». Phdthesis, AgroParisTech. https://pastel.archives-ouvertes.fr/tel-01130084.

Value

A data frame with columns:

- `Qmin`: Minimum discharge value of the bin.
- `Qmax`: Maximum discharge value of the bin.
- Quantiles of the relative error (Qsim / Qobs) for the specified probabilities.

Each row corresponds to a bin defined by `bind_size` and `bind_step`.

Functions

- `predict_uncertainty_calibration()`: Predictive Uncertainty Calibration
  Model of predictive uncertainty based on quantiles of the empirical distribution of relative errors for different bins of simulated discharge values using a moving bin approach.
- `predict_uncertainty()`: Predictive Uncertainty Prediction Using the uncertainty table generated by `predict_uncertainty_calibration`, this function predicts the relative uncertainty for new simulated discharge values.
- `plot_uncertainty_time_series()`: Plot Predicted Uncertainty Time Series. This function visualizes the predicted discharge values along with their uncertainty intervals.
- `plot_uncertainty_calibration()`: Plot Predicted Uncertainty Calibration. This function visualizes the relative error between observed and simulated discharge values, along with the predicted uncertainty intervals.
- `predict_uncertainty_bind_size()`: Predict Bind Size for Uncertainty Calibration. This function suggests an appropriate bind size for the uncertainty calibration based on the number of observations.
- `transfer_uncertainty()`: Transfer Uncertainty Quantiles.

Examples

```
library(ggplot2)
library(dplyr)
library(tidyr)

data(L0123001, package = "airGR")
InputsModel <- CreateInputsModel(
  RunModel_GR4J,
  DatesR = BasinObs$DatesR,
  Precip = BasinObs$P,
  PotEvap = BasinObs$E
)
Ind_Run <- seq(
  which(format(BasinObs$DatesR, format = "%Y-%m-%d") == "1985-01-01"),
  which(format(BasinObs$DatesR, format = "%Y-%m-%d") == "2012-12-31")
)
RunOptions <- CreateRunOptions(
  RunModel_GR4J,
  InputsModel = InputsModel,
  IndPeriod_Run = Ind_Run,
```

```
  IndPeriod_WarmUp = (Ind_Run[1] - 365):(Ind_Run[1] - 1)
)
Param <- c(257.237556, 1.012237, 88.234673, 2.207958)
OutputsModel <- RunModel_GR4J(
  InputsModel = InputsModel,
  RunOptions = RunOptions,
  Param = Param
)
Qsim <- OutputsModel$Qsim
Qobs <- BasinObs$Qmm[Ind_Run]

uncertainty_table <- predict_uncertainty_calibration(Qobs, Qsim)

Qsim_uncertain <- predict_uncertainty(Qsim, uncertainty_table)

# Plot the result of uncertainty estimation
plot_uncertainty_calibration(Qsim_uncertain, Qobs)

# Plot the time series with uncertainty intervals
ts_selection <- 1:365
plot_uncertainty_time_series(
  Qsim_uncertain[ts_selection, ],
  BasinObs$DatesR[Ind_Run[ts_selection]]
)

# Plot the time series with uncertainty intervals compared with observations
plot_uncertainty_time_series(
  Qsim_uncertain[ts_selection, ],
  BasinObs$DatesR[Ind_Run[ts_selection]],
  Qobs[ts_selection]
)

# Transfer uncertainty estimation to another simulated station
# Here we artificially modify Qsim to simulate another station
Qsim_new <- Qsim * 1.2 + 0.5
uncertainty_table_new <- transfer_uncertainty(Qsim_new, uncertainty_table)

Qsim_uncertain_new <- predict_uncertainty(Qsim_new, uncertainty_table_new)

plot_uncertainty_time_series(
  Qsim_uncertain[ts_selection, ],
  BasinObs$DatesR[Ind_Run[ts_selection]]
)
```

`sb_to_outlets` *Title*

Description

Title

Usage

```
sb_to_outlets(sb)
```

Arguments

sb	Output of extract_sub_basins.

Value

A simple feature with points representing sub-basins outlets.

Index