

Package: airGRccia (via r-universe)

October 21, 2026

Title Climate Change Impact and Adaptation with airGRiwrn

Version 0.2.0.9006

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

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

BugReports <https://forge.inrae.fr/airgriwrn/airgrccia/-/issues>

License AGPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends airGRiwrn

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, ggplot2, httr, hubeau, logger, methods,
mirai, mori, 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/roxygen2/version 8.1.0

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 2026-10-21 07:46:20 UTC

RemoteUrl <https://forge.inrae.fr/airgriwrn/airgrccia>

```
RemoteRef main
RemoteSha 0c371c0933e83cb05981dd3dfefc529a806756cc
```

Contents

calc_hypso	2
colorize_polygons	3
CreateBasinsObs	3
extract_reaches	4
extract_sub_basins	6
get_climate_safran	9
get_future_workers	10
get_hubeau_flows	11
get_ign_dem	12
get_ign_streams	14
get_ram	15
get_safran_var_units	16
getCachePath	17
hydro_get_station	18
is_projected_crs	19
not.is.na	20
predict_uncertainty_calibration	20
sb_to_outlets	24
Index	25

calc_hypso	<i>Calculate hypsometric curves for sub-basins</i>
------------	--

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 [airGRiwrn::CreateInputsModel.GRiwrn](#) for the `HypsoData` argument.

colorize_polygons	<i>Greedy Colorization of sf Polygons</i>
-------------------	---

Description

Assigns color classes to an sf polygon object using a greedy graph coloring algorithm. Two adjacent polygons (sharing a boundary) will never receive the same color class.

Usage

```
colorize_polygons(sb)
```

Arguments

sb An sf polygon object.

Value

The same sf object with an additional column `col_class` (factor) indicating the color class.

CreateBasinsObs	<i>Create a BasinsObs object (Hydroclimatic time series)</i>
-----------------	--

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

<code>s_sb</code>	Climate data (See get_climate_safran)
<code>ids</code>	Vector of sub-basin ids
<code>Qobs</code>	Observed flows (See get_hubeau_flows)
<code>Qinf</code>	(optional) matrix or data.frame of numeric containing observed flows. It must be provided only for nodes of type "Direct injection" and "Diversiion". See CreateGRiwrn for details about these node types. Unit is [mm per time step] for nodes with an area, and [m ³ per time step] for nodes with <code>area=NA</code> . Column names correspond to node IDs. Negative flows are abstracted from the model and positive flows are injected to the model
<code>Qrelease</code>	(optional) matrix or data.frame of numeric containing release flows by nodes using the model <code>RunModel_Reservoir</code> [m3 per time step]
<code>measures</code>	character vector of climate measures to include in the output object.

Value

A [list](#) containing an `itemDatesR`, vector of dates corresponding to the time dimension of the data, following by [data.frames](#) for each measure including climate data ("Precip", "PotE-vap", "TempMean"), and observed flows ("Q").

<code>extract_reaches</code>	<i>Extract River Reaches by Sub-basin from a DEM</i>
------------------------------	--

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 = 2,
  max_buffer = 3,
  dTolerance = 2 * mean(terra::res(d8fd)),
  workers = get_future_workers(),
  filename = getCachePath(sb, "reaches", "gpkg"),
  overwrite = FALSE
)
```

Arguments

<code>sb</code>	An <code>sf</code> object representing the perimeter of sub-basins. Typically the output of <code>extract_sub_basins()</code> .
<code>d8fd</code>	A <code>SpatRaster</code> or <code>character</code> path to the file containing D8 flow directions. Can also be obtained from <code>extract_sub_basins()</code> .
<code>streams</code>	An <code>sf</code> object or <code>character</code> path to the file containing the stream network (e.g., output of <code>get_ign_streams()</code>).
<code>max_tries</code>	<code>integer</code> . Maximum cells tested for retrieving a reach.
<code>eps</code>	<code>numeric</code> . Minimum acceptable distance between the downstream reach and the downstream outlet in cells.
<code>max_step_back</code>	<code>integer</code> Maximum contiguous back steps in stream cell research.
<code>max_azimut</code>	<code>integer</code> Maximum allowed change in flow direction when searching for the next stream cell downstream (see details).
<code>max_buffer</code>	<code>integer</code> Maximum number of cells matching around stream cells.
<code>dTolerance</code>	<code>numeric</code> . The tolerance for simplifying the reach geometry (Used in <code>sf::st_simplify</code>).
<code>workers</code>	<code>integer</code> Number of workers to use for parallel processing.
<code>filename</code>	<code>character</code> If not NULL, the streams geometry will be saved to this file and can be used as cache if <code>overwrite = FALSE</code> (See details for available formats).
<code>overwrite</code>	<code>logical</code> Whether to overwrite the cached file if it exists.

Details

max_azimut: During the tracing process, the algorithm explores neighboring cells allowing gradual deviations from the current D8 flow direction. A higher value permits larger directional changes, which can help follow the stream network when it does not perfectly align with the D8 pointers (e.g., due to gaps or digitization artifacts).

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*_

Value

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

extract_sub_basins	<i>Extract sub-basins from DEM</i>
--------------------	------------------------------------

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,  
  colorize = TRUE  
)  
  
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 optionally "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); OR an sf object with a column "id" and optionally "area", where coordinates are extracted from the POINT geometry. When providing an sf object, its CRS will be automatically reprojected to match the DEM CRS if different.
dem	character containing the filename of the DEM raster, or a SpatRaster object
streams	A sf object containing the streams.
crs	Either a crs object, a character string, or a numeric EPSG code. The CRS used for outlets coordinates. If NULL, the CRS of <code>dem</code> is used.
threshold	numeric Minimum threshold of contributing area to extract streams (unit: square meters)
snap_dist	numeric Maximum distance between outlet and a stream vector for relocating outlet in the stream (in meters)
area_tolerance	numeric Maximum relative error between the a priori area and the computed area of the sub-basin
max_radius	numeric 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 <code>overwrite = FALSE</code> . If TRUE (default), a filename will be automatically generated with getCachePath based on the input parameters. Only RDS files are supported for caching.
overwrite	logical Whether to overwrite the cached file if it exists.
verbose_mode	Sets verbose mode. If verbose mode is <code>FALSE</code> , tools will not print output messages. Default: NULL will use the value in WhiteboxTools settings, see wbt_verbose() for details.
colorize	Logical, default TRUE. If TRUE, applies greedy graph coloring to the sub-basin polygons using colorize_polygons() , adding a <code>col_class</code> column for thematic mapping of adjacent sub-basins.
i	integer row number to process in <code>outlets</code>
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 an `sf` object with class "SubBasins" containing polygon geometries (in the CRS defined by the `crs` parameter) and the following columns:

- `id`: Sub-basin identifier (character), matching the outlet ID.
- `tot_area`: Total watershed area delineated from the outlet before removing overlapping sub-basin areas (in square meters).
- `area`: Actual sub-basin area after removing upstream overlapping basin extents (in square meters).
- `apriori_area`: A priori sub-basin area provided by the user (in square meters); `NA` if not supplied in the `outlets` input.
- `mvd_outlet_x`, `mvd_outlet_y`: X and Y coordinates of the outlet after snapping to the stream network (in map units of the CRS).
- `outlet_x`, `outlet_y`: Original X and Y coordinates of the outlet before snapping (in map units of the CRS).
- `mvd_dist`: Euclidean distance between the original outlet and the snapped outlet (in meters).
- `down`: ID of the downstream sub-basin into which each sub-basin drains; `NA` for the most downstream sub-basin(s).

When `colorize = TRUE`, an additional column `col_class` is added indicating the greedy graph coloring class for thematic mapping.

Additionally, the returned object carries an attribute `files` (named character vector) with paths to all intermediate cached raster and vector files generated during processing.

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

<code>x</code>	A bounding box bbox , a simple feature collection sf , or a simple feature geometry sfc , defining the area of interest.
<code>fields</code>	A named list where each element corresponds to a GR-compatible variable name (e.g., <code>Precip</code> , <code>PotEvap</code> , <code>TempMean</code>) and contains one or more SIM2 field names to be retrieved and optionally summed.
<code>date_start</code> , <code>date_end</code>	A Date object or a character in format <code>"%Y-%m-%d"</code> specifying start and end dates of the extraction period.
<code>overlay_mode</code>	character parameter passed to <code>RADIS::spatial_combination()</code> , controlling how climate data are overlaid with the input geometry. Valid values are <code>"within"</code> , <code>"intersecting"</code> , <code>"extent"</code> , <code>"intersect_geometry"</code> and <code>"aggregate"</code> .
<code>filename</code>	character If not NULL, path of cached data (See details).
<code>overwrite</code>	logical Whether to overwrite the cached file if it exists.
<code>...</code>	Additional arguments passed to <code>RADIS::get_sim2_daily()</code> .

Details

Default value for `filename` depends on the parameter `overlay_mode`: data are cached as a NetCDF file when `overlay_mode` is "intersect_geometry" or "aggregate", and in RDS format for other modes (e.g., "extent", "within", "intersecting").

The return value depends on the `overlay_mode` argument passed internally to `RADIS::spatial_combination()`, see the documentation of that function for details on the different modes.

Value

An object containing the extracted and aggregated climate variables, named using the keys of `fields` (e.g., "Precip", "PotEvap", etc.). The type depends on the `overlay_mode`:

- For "extent", "aggregate", or "within": a `stars` object where each element is a multi-dimensional array (spatial dimensions and time).
- For "intersect_geometry": an `sf` data frame with clipped polygon geometries, one row per intersecting cell.
- For "intersecting": a list of integer matrices (not a `stars` object), where each matrix indicates which raster cells intersect each mask polygon.

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)
```

<code>get_future_workers</code>	<i>Compute number of workers given available memory and CPUs</i>
---------------------------------	--

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()
```

<code>get_hubeau_flows</code>	<i>Retrieve river flow observation from the Hydrometrie Hub'Eau API</i>
-------------------------------	---

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

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

<code>get_ign_dem</code>	<i>Extract DEM from IGN API</i>
--------------------------	---------------------------------

Description

This function downloads DEM from IGN API and performs 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 = 2154,
    neg_outlier_threshold = 0,
    max_tries = 3,
    filename = getCachePath(list(x, layer, res, crs, neg_outlier_threshold), prefix =
        "dem", fileext = "rds"),
    overwrite = FALSE,
    ...
)

```

Arguments

x	A bounding box bbox , a simple feature collection sf , or a simple feature geometry sfc , defining the area of interest.
...	Parameters passed to S3 methods.
layer	character ; name of the WFS layer. Must correspond to a layer available on the IGN WFS service (see get_layers_metadata()).
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	The CRS to use for the output (the CRS of x is used by default).
neg_outlier_threshold	Threshold under which data is rejected as outlier and replaced by the mean of neighbour cells
max_tries	Maximum number of attempts to retrieve the DEM raster from the IGN WMS server. If a request fails due to a transient network or GDAL error, the function will retry up to this number of times before giving up. Default is 3.
filename	character If not <code>NULL</code> , the streams geometry will be saved to this file and can be used as cache if <code>overwrite = FALSE</code> (See details for available formats).
overwrite	logical Whether to overwrite the cached file if it exists.

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")`.

The **filename** parameter allows you to cache the retrieved streams data to a file. If the file already exists and **overwrite** is set to `FALSE`, the function will load the streams data from the cached file instead of downloading it again. The supported file formats for caching are either RDS format or any format supported by the **terra** package (e.g., GeoTIFF, NetCDF).

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	<i>Get streams from IGN</i>
-----------------	-----------------------------

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 = getCachePath(list(x, layer, crs, filters), prefix = "streams", fileext =
    "gpkg"),
  overwrite = FALSE,
  ...
)
```

Arguments

x	A bounding box bbox , a simple feature collection sf , or a simple feature geometry sfc , defining the area of interest.
...	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 <code>get_layers_metadata()</code>).
crs	The CRS to use for the output (the CRS of <code>x</code> is used by default).
filters	A list of filters to apply to the streams data (See details)
filename	character If not NULL, the streams geometry will be saved to this file and can be used as cache if <code>overwrite = FALSE</code> (See details for available formats).
overwrite	logical Whether to overwrite the cached file if it exists.

Details

The `filename` parameter allows you to cache the retrieved streams data to a file. If the file already exists and `overwrite` is set to `FALSE`, the function will load the streams data from the cached file instead of downloading it again. The supported file formats for caching are either RDS format or any format supported by the `sf` package (e.g., GeoPackage, Shapefile).

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)
```

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	<i>Get cache path</i>
--------------	-----------------------

Description

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

Usage

```
getCacheDir(
  ...,
  path = Sys.getenv("CACHE_DIR_AIRGRCCIA"),
  basepath = rappdirs::user_cache_dir("airGRccia")
)

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

clearCache(cache_dir = getCacheDir(), erase = TRUE)
```

Arguments

...	Arguments passed to <code>RADIS::radis_cache_dir()</code> .
path	Path to the cache directory (default from environment variable <code>CACHE_DIR_AIRGRCCIA</code>). If set, RADIS is bypassed and the directory is created directly.
basepath	Folder in which to add package cache directory (by default OS dependent cache folder via <code>rappdirs::user_cache_dir()</code>).
x	Object to save (or property of object that can be used as a signature)
prefix	character Optional prefix to add to the file name (default variable name)
fileext	character Optional file extension to use for the file (e.g., "rds", "csv", etc.)
cache_dir	character Path to the cache directory (default from <code>getCacheDir()</code>)
erase	If TRUE (default), delete all cached files in <code>cache_dir</code> . If FALSE, remove only the oldest cached files until the total cache size is within the limit configured by <code>options(airgrccia.cache.max.size)</code> .

Details

Cache directory management is handled through the [RADIS](#) package, which provides automatic size-based cleanup of cached files.

On each call to `clearCache()` with `erase = FALSE`, airGRccia automatically removes the oldest cached files until the total cache size falls below a configurable threshold controlled

by the R option `airgrccia.cache.max.size`. This option is set to "10GB" by default when the package loads.

You can change this limit at any time:

```
options(airgrccia.cache.max.size = "2GB") # reduce cache max size
options(airgrccia.cache.max.size = "50GB") # increase cache max size
```

Valid values follow the same format as `rlang::parse_bytes()`, e.g. "100MB", "1GB", "2GiB", or a raw number of bytes (e.g., 1e9).

Value

The cache path.

Functions

- `getCacheDir()`: Returns cache path root and creates the folder if it does not exist. Delegates to `RADIS::radis_cache_dir()` for automatic size-based cleanup.
- `getCachePath()`: Returns cache path for a specific file.
- `clearCache()`: Clear the cache directory. When `erase = FALSE`, only removes oldest files until the total cache size falls below the threshold set by R option `airgrccia.cache.max.size`.

Examples

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

## End(Not run)
```

<code>hydro_get_station</code>	<i>Get info on hydrometric station</i>
--------------------------------	--

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")
```

<code>is_projected_crs</code>	<i>Check if the CRS is projected</i>
-------------------------------	--------------------------------------

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)
```

<code>not.is.na</code>	<i>Shortcut for <code>!is.na</code></i>
------------------------	---

Description

Shortcut for `!is.na`

Usage

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

Arguments

`x` an R object to be tested

Value

logical See [is.na](#).

<code>predict_uncertainty_calibration</code>	<i>Predictive Uncertainty functions</i>
--	---

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), must be ≥ 0 .
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 <code>c(0.05, 0.5, 0.95)</code>) for a 90% prediction interval.
uncertainty_table	Data frame returned by <code>predict_uncertainty_calibration</code> .
Qsim_uncertain	Data frame returned by <code>predict_uncertainty</code> , containing simulated discharge and uncertainty quantiles.
DatesR	Dates corresponding to the discharge values.
bounds	Names of the columns in <code>QSim_uncertain</code> representing the uncertainty bounds (default is the first and last error quantile).
rubbon_fill	Fill color for the uncertainty ribbon.

<code>ribbon_alpha</code>	Alpha transparency for the uncertainty ribbon.
<code>line_colors</code>	Named vector specifying colors for the simulated and observed discharge lines.
<code>labels</code>	List of labels for the plot (title, x, y, ...) used in <code>ggplot2::labs()</code> .
<code>outliers_quantile</code>	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. Moreover, 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 (Q_{sim} / Q_{obs}) for the specified probabilities.

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

A data frame with a `Qsim` column and one column per error quantile, containing the predicted quantiles ($Q_{sim} * \text{quantile of relative error}$).

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

airGRiwrn::CreateInputsModel.GRiwrn, [3](#)

bbox, [9](#), [13](#), [14](#), [19](#)

calc_hypso, [2](#)

character, [4](#), [5](#), [7](#), [9](#), [12](#), [13](#), [15–17](#)

clearCache (*getCachePath*), [17](#)

clearCache(), [17](#)

colorize_polygons, [3](#)

colorize_polygons(), [7](#)

CreateBasinsObs, [3](#)

CreateGRiwrn, [4](#)

crs, [7](#)

data.frame, [4](#), [7](#), [12](#), [19](#)

Date, [9](#)

extract_reaches, [4](#)

extract_sub_basins, [2](#), [6](#), [24](#)

extract_sub_basins(), [5](#)

find_near_target
(*extract_sub_basins*), [6](#)

get_climate_safran, [4](#), [9](#)

get_future_workers, [10](#)

get_hubeau_flows, [4](#), [11](#)

get_ign_dem, [2](#), [12](#)

get_ign_streams, [14](#)

get_ign_streams(), [5](#)

get_layers_metadata(), [13](#), [15](#)

get_ram, [15](#)

get_safran_var_units, [16](#)

getCacheDir (*getCachePath*), [17](#)

getCacheDir(), [17](#)

getCachePath, [7](#), [17](#)

ggplot2::labs(), [22](#)

happign::get_wfs, [14](#)

hubeau::get_hydrometrie_obs_elab, [11](#), [12](#)

hubeau::get_hydrometrie_sites, [19](#)

hydro_get_station, [18](#)

hydro_station_as_sf
(*hydro_get_station*), [18](#)

integer, [5](#), [7](#), [11](#)

is.na, [20](#)

is_projected_crs, [19](#)

list, [4](#), [9](#)

logical, [5](#), [7](#), [9](#), [12](#), [13](#), [15](#), [20](#)

matrix, [3](#), [4](#)

not.is.na, [20](#)

numeric, [4](#), [5](#), [7](#), [11](#)

plot_uncertainty_calibration
(*predict_uncertainty_calibration*), [20](#)

plot_uncertainty_time_series
(*predict_uncertainty_calibration*), [20](#)

predict_uncertainty
(*predict_uncertainty_calibration*), [20](#)

predict_uncertainty_bind_size
(*predict_uncertainty_calibration*), [20](#)

predict_uncertainty_calibration, [20](#)

RADIS, [17](#)

RADIS::get_sim2_daily(), [10](#)

RADIS::radis_cache_dir(), [17](#), [18](#)

RADIS::spatial_combination(), [9](#), [10](#)

rappdirs::user_cache_dir(), [17](#)

rlang::parse_bytes(), [18](#)

sb_to_outlets, [24](#)

`sf`, [5](#), [7–10](#), [13–15](#), [19](#)
`sf::st_simplify`, [5](#)
`sfc`, [9](#), [13](#), [14](#)
`SpatRaster`, [5](#), [7](#), [14](#)
`stars`, [10](#)
`SubBasins`, [12](#)

`transfer_uncertainty`
 (*predict_uncertainty_calibration*),
 [20](#)

`vector`, [16](#)

`watershed_get_from_outlet`
 (*extract_sub_basins*), [6](#)