Package ‘meteoland’

October 17, 2024

Type Package

Title Landscape Meteorology Tools

Version 2.2.2

Date 2024-10-17

Description Functions to estimate weather variables at any position of a landscape [De Caceres et al. (2018) <doi:10.1016/j.envsoft.2018.08.003>].

License GPL (>= 2)

URL https://emf-creaf.github.io/meteoland/, https://github.com/emf-creaf/meteoland

BugReports https://github.com/emf-creaf/meteoland/issues

LazyData TRUE

Depends R (>= 3.5.0)

Imports methods, sf, stars, stats, Rcpp, units, lifecycle, cli, dplyr, tidyr, rlang, assertthat, purrr, ncdfgeom, ncmeta, lubridate, cubelyr

Suggests knitr, rmarkdown, testthat (>= 3.0.0), meteospain, worldmet, tibble

LinkingTo Rcpp

Encoding UTF-8

NeedsCompilation yes

VignetteBuilder knitr

RoxygenNote 7.3.2

Config/testthat/edition 3

Config/testthat/parallel true

Author Miquel De Cáceres [aut, cre] (<https://orcid.org/0000-0001-7132-2080>), Víctor Granda [aut] (<https://orcid.org/0000-0002-0469-1991>), Nicolas Martin [aut] (<https://orcid.org/0000-0001-7574-0108>), Antoine Cabon [aut] (<https://orcid.org/0000-0001-6426-1726>)

Maintainer Miquel De Cáceres <miquelcaceres@gmail.com>

Repository CRAN

Date/Publication 2024-10-17 17:30:02 UTC

Contents

`add_topo` *Add topography data to meteo object*

Description

Add topography data to meteo object

Usage

```
add_topo(meteo, topo, verbose = getOption("meteoland_verbosity", TRUE))
```

Arguments

`meteo`	meteo object
`topo`	topo object
`verbose`	Logical indicating if the function must show messages and info. Default value checks `"meteoland_verbosity"` option and if not set, defaults to TRUE. It can be turned off for the function with FALSE, or session wide with `options(meteoland_verbosity = FALSE)`

Details

When using meteo data without topography info to create an interpolator, topography must be added

Value

meteo with the topography info added

See Also

Other interpolator functions: `create_meteo_interpolator()`, `get_interpolation_params()`, `read_interpolator()`, `set_interpolation_params()`, `with_meteo()`, `write_interpolator()`

Examples

```
# example meteo
data(meteoland_meteo_no_topo_example)
# example topo
data(meteoland_topo_example)
# add topo
with_meteo(meteoland_meteo_no_topo_example) |>
  add_topo(meteoland_topo_example)
```

`complete_meteo` *Complete missing meteo variables*

Description

Calculates missing values of relative humidity, radiation and potential evapotranspiration from a data frame with daily values of minimum/maximum/mean temperature and precipitation. The function takes a meteo object (with meteoland names) and complete any missing variable if it is possible

Usage

```
complete_meteo(meteo, verbose = getOption("meteoland_verbosity", TRUE))
```

Arguments

`meteo` meteoland weather data

`verbose` Logical indicating if the function must show messages and info. Default value checks "meteoland_verbosity" option and if not set, defaults to TRUE. It can be turned off for the function with FALSE, or session wide with options(meteoland_verbosity = FALSE)

Details

#' The function fills values for humidity, radiation and PET only if they are missing in the input data frame. If a column 'SpecificHumidity' is present in the input data, relative humidity is calculated from it. Otherwise, relative humidity is calculated assuming that dew point temperature equals the minimum temperature. Potential solar radiation is calculated from latitude, slope and aspect. Incoming solar radiation is then corrected following Thornton & Running (1999) and potential evapotranspiration following Penman (1948).

Value

the same meteo data provided with the the variables completed

Author(s)

Miquel De Cáceres Ainsa, EMF-CREAF

Victor Granda García, EMF-CREAF

References

Thornton, P.E., Running, S.W., 1999. An improved algorithm for estimating incident daily solar radiation from measurements of temperature, humidity, and precipitation. Agric. For. Meteorol. 93, 211-228.

Penman, H. L. 1948. Natural evaporation from open water, bare soil and grass. Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences, 193, 120-145.

Examples

```
# example data
data("meteoland_meteo_example")

# remove MinRelativeHumidity
meteoland_meteo_example$MinRelativeHumidity <- NULL
# complete vars
completed_meteo <- complete_meteo(meteoland_meteo_example)
# check MinRelativeHumidity
completed_meteo$MinRelativeHumidity
```

create_meteo_interpolator

Meteoland interpolator creation

Description

Function to create the meteoland interpolator

Usage

```
create_meteo_interpolator(
  meteo_with_topo,
  params = NULL,
  verbose = getOption("meteoland_verbosity", TRUE)
)
```

Arguments

`meteo_with_topo`	Meteo object, as returned by `with_meteo`
`params`	Interpolation parameters as a list. Typically the result of `defaultInterpolationParams`.
`verbose`	Logical indicating if the function must show messages and info. Default value checks `"meteoland_verbosity"` option and if not set, defaults to TRUE. It can be turned off for the function with FALSE, or session wide with `options(meteoland_verbosity = FALSE)`

Details

This function takes meteorology information and a list of interpolation parameters and creates the interpolator object to be ready to use.

Value

an interpolator object (stars)

Author(s)

Victor Granda García, EMF-CREAF

Miquel De Cáceres Ainsa, EMF-CREAF

See Also

Other interpolator functions: `add_topo()`, `get_interpolation_params()`, `read_interpolator()`, `set_interpolation_params()`, `with_meteo()`, `write_interpolator()`

Examples

```
# example meteo data
data(meteoland_meteo_example)

# create the interpolator with default params
with_meteo(meteoland_meteo_example) |>
  create_meteo_interpolator()

# create the interpolator with some params changed
with_meteo(meteoland_meteo_example) |>
  create_meteo_interpolator(params = list(debug = TRUE))
```

---

defaultInterpolationParams

Default interpolation parameters

Description

Returns a list with the default parameterization for interpolation. Most parameter values are set according to Thornton et al. (1997).

Usage

```
defaultInterpolationParams()
```

Value

A list with the following items (default values in brackets):

- `initial_Rp` [= 140000]: Initial truncation radius.
- `iterations` [= 3]: Number of station density iterations.
- `alpha_MinTemperature` [= 3.0]: Gaussian shape parameter for minimum temperature.
- `alpha_MaxTemperature` [= 3.0]: Gaussian shape parameter for maximum temperature.
- `alpha_DewTemperature` [= 3.0]: Gaussian shape parameter for dew-point temperature.
- `alpha_PrecipitationEvent` [= 5.0]: Gaussian shape parameter for precipitation events.
- `alpha_PrecipitationAmount` [= 5.0]: Gaussian shape parameter for the regression of precipitation amounts.
- `alpha_Wind` [= 3.0]: Gaussian shape parameter for wind.
- `N_MinTemperature` [= 30]: Average number of stations with non-zero weights for minimum temperature.
- `N_MaxTemperature` [= 30]: Average number of stations with non-zero weights for maximum temperature.
- `N_DewTemperature` [= 30]: Average number of stations with non-zero weights for dew-point temperature.

- N_PrecipitationEvent [= 5]: Average number of stations with non-zero weights for precipitation events.
- N_PrecipitationAmount [= 20]: Average number of stations with non-zero weights for the regression of precipitation amounts.
- N_Wind [= 2]: Average number of stations with non-zero weights for wind.
- St_Precipitation [= 5]: Number of days for the temporal smoothing of precipitation.
- St_TemperatureRange [= 15]: Number of days for the temporal smoothing of temperature range.
- pop_crit [= 0.50]: Critical precipitation occurrence parameter.
- f_max [= 0.6]: Maximum value for precipitation regression extrapolations (0.6 equals to a maximum of 4 times extrapolation).
- wind_height [= 10]: Wind measurement height (in m).
- wind_roughness_height [= 0.001]: Wind roughness height (in m), for PET calculations.
- penman_albedo [= 0.25]: Albedo for PET calculations.
- penman_windfun [= "1956"]: Wind speed function version, either "1948" or "1956", for PET calculation.
- debug [= FALSE]: Boolean flag to show extra console output.

Author(s)

Miquel De Cáceres Ainsa, CREAF

References

Thornton, P.E., Running, S.W., White, M. A., 1997. Generating surfaces of daily meteorological variables over large regions of complex terrain. J. Hydrol. 190, 214–251. doi:10.1016/S0022-1694(96)03128-9.

De Caceres M, Martin-StPaul N, Turco M, Cabon A, Granda V (2018) Estimating daily meteorological data and downscaling climate models over landscapes. Environmental Modelling and Software 108: 186-196.

See Also

interpolate_data

---

get_interpolation_params

Retrieving interpolation parameters from interpolator object

Description

Retrieve the parameter list from and interpolator object

Usage

```
get_interpolation_params(interpolator)
```

Arguments

`interpolator`	interpolator object as returned by `create_meteo_interpolator`

Value

The complete parameter list from the interpolator object

Author(s)

Victor Granda García, EMF-CREAF

See Also

Other interpolator functions: `add_topo()`, `create_meteo_interpolator()`, `read_interpolator()`, `set_interpolation_params()`, `with_meteo()`, `write_interpolator()`

Examples

```
# example interpolator
data(meteoland_interpolator_example)
# get the params from the interpolator
get_interpolation_params(meteoland_interpolator_example)
```

---

humidity_relative2dewtemperature

Humidity conversion tools

Description

Functions to transform relative humidity to specific humidity or dew point temperature and viceversa.

Usage

```
humidity_relative2dewtemperature(Tc, HR)

humidity_dewtemperature2relative(Tc, Td, allowSaturated = FALSE)

humidity_specific2relative(Tc, HS, allowSaturated = FALSE)

humidity_relative2specific(Tc, HR)
```

Arguments

Tc	A numeric vector of temperature in degrees Celsius.
HR	A numeric vector of relative humidity (in %).
Td	A numeric vector of dew temperature in degrees Celsius.
allowSaturated	Logical flag to allow values over 100%
HS	A numeric vector of specific humidity (unitless).

Value

A numeric vector with specific or relative humidity.

Author(s)

Nicholas Martin-StPaul, INRA

Miquel De Cáceres Ainsa, CREAF

See Also

complete_meteo

---

interpolate_data *Interpolation process for spatial data*

Description

Interpolate spatial data to obtain downscaled meteorologic variables

Usage

```
interpolate_data(
  spatial_data,
  interpolator,
  dates = NULL,
  variables = NULL,
  ignore_convex_hull_check = FALSE,
  verbose = getOption("meteoland_verbosity", TRUE)
)
```

Arguments

spatial_data	An sf or stars raster object to interpolate
interpolator	A meteoland interpolator object, as created by create_meteo_interpolator
dates	vector with dates to interpolate (must be within the interpolator date range). Default to NULL (all dates present in the interpolator object)

variables
: vector with variable names to be interpolated. NULL (default), will interpolate all variables. Accepted names are "Temperature", "Precipitation", "RelativeHumidity", "Radiation" and "Wind"

ignore_convex_hull_check
: Logical indicating whether errors in convex hull checks should be ignored. Checking for points to be inside the convex hull will normally raise an error if >10% of points are outside. Setting `ignore_convex_hull_check = TRUE` means that a warning is raised but interpolation is performed, which can be useful to users interpolating on a few points close but outside of the convex hull.

verbose
: Logical indicating if the function must show messages and info. Default value checks `"meteoland_verbosity"` option and if not set, defaults to TRUE. It can be turned off for the function with FALSE, or session wide with `options(meteoland_verbosity = FALSE)`

Details

This function takes a spatial data object (sf or stars raster), an interpolator object (`create_meteo_interpolator`) and a vector of dates to perform the interpolation of the meteorologic variables for the spatial locations present in the `spatial_data` object.

Value

an object with the same class and structure as the provided spatial data with the results of the interpolation joined. In the case of spatial data being an sf, the results are added as a list-type column that can be unnested with `unnest`. In the case of a stars raster object, interpolation results are added as attributes (variables)

Spatial data

The spatial data provided must be of two types. (I) A sf object containing POINT for each location to interpolate or (II) a stars raster object for which the interpolation should be done. Independently of the class of `spatial_data` it has to have some mandatory variables, namely `elevation`. It should also contain `aspect` and `slope` for a better interpolation process, though this two variables are not mandatory.

Author(s)

Victor Granda García, EMF-CREAF

Miquel De Cáceres Ainsa, EMF-CREAF

Examples

```
# example of data to interpolate and example interpolator
data("points_to_interpolate_example")
data("meteoland_interpolator_example")

# interpolate data
res <- interpolate_data(points_to_interpolate_example, meteoland_interpolator_example)
```

```
# check result
# same class as input data
class(res)
# data
res
# results for the first location
res[["interpolated_data"]][1]
# unnest results
tidyr::unnest(res, cols = "interpolated_data")
```

`interpolation_cross_validation`
Calibration and validation of interpolation procedures

Description

Calibration and validation of interpolation procedures

Usage

```
interpolation_cross_validation(
  interpolator,
  stations = NULL,
  verbose = getOption("meteoland_verbosity", TRUE)
)

interpolator_calibration(
  interpolator,
  stations = NULL,
  update_interpolation_params = FALSE,
  variable = "MinTemperature",
  N_seq = seq(5, 30, by = 5),
  alpha_seq = seq(0.25, 10, by = 0.25),
  verbose = getOption("meteoland_verbosity", TRUE)
)
```

Arguments

`interpolator`	A meteoland interpolator object, as created by `create_meteo_interpolator`
`stations`	A vector with the stations (numeric for station indexes or character for stations id) to be used to calculate `"MAE"`. All stations with data are included in the training set but predictive `"MAE"` are calculated for the stations subset indicated in `stations` param only. If `NULL` all stations are used in the predictive `"MAE"` calculation.

verbose: Logical indicating if the function must show messages and info. Default value checks "meteoland_verbosity" option and if not set, defaults to TRUE. It can be turned off for the function with FALSE, or session wide with options(meteoland_verbosity = FALSE)

update_interpolation_params: Logical indicating if the interpolator object must be updated with the calculated parameters. Default to FALSE

variable: A string indicating the meteorological variable for which interpolation parameters "N" and "alpha" will be calibrated. Accepted values are:

- MinTemperature (kernel for minimum temperature)
- MaxTemperature (kernel for maximum temperature)
- DewTemperature (kernel for dew-temperature (i.e. relative humidity))
- Precipitation (to calibrate the same kernel for both precipitation events and regression of precipitation amounts; not recommended)
- PrecipitationAmount (kernel for regression of precipitation amounts)
- PrecipitationEvent (kernel for precipitation events)

N_seq: Numeric vector with "N" values to be tested

alpha_seq: Numeric vector with "alpha"

Details

Function interpolator_calibration determines optimal interpolation parameters "N" and "alpha" for a given meteorological variable. Optimization is done by minimizing mean absolute error ("MAE") (Thornton *et al.* 1997). Function interpolation_cross_validation calculates average mean absolute errors ("MAE") for the prediction period of the interpolator object. In both calibration and cross validation procedures, predictions for each meteorological station are made using a *leave-one-out* procedure (i.e. after excluding the station from the predictive set).

Value

interpolation_cross_validation returns a list with the following items

- errors: Data frame with each combination of station and date with observed variables, predicated variables and the total error (predicted - observed) calculated for each variable
- station_stats: Data frame with error and bias statistics aggregated by station
- dates_stats: Data frame with error and bias statistics aggregated by date
- r2: correlation indexes between observed and predicted values for each meteorological variable

If update_interpolation_params is FALSE (default), interpolator_calibration returns a list with the following items

- MAE: A numeric matrix with the mean absolute error values, averaged across stations, for each combination of parameters "N" and "alpha"
- minMAE: Minimum MAE value
- N: Value of parameter "N" corresponding to the minimum MAE

- alpha: Value of parameter "alpha" corresponding the the minimum MAE
- observed: matrix with observed values (meteorological measured values)
- predicted: matrix with interpolated values for the optimum parameter combination

If `update_interpolation_params` is FALSE, `interpolator_calibration` returns the interpolator provided with the parameters updated

Functions

- `interpolation_cross_validation()`:

Author(s)

Miquel De Cáceres Ainsa, EMF-CREAF

Victor Granda García, EMF-CREAF

References

Thornton, P.E., Running, S.W., 1999. An improved algorithm for estimating incident daily solar radiation from measurements of temperature, humidity, and precipitation. Agric. For. Meteorol. 93, 211–228. doi:10.1016/S0168-1923(98)00126-9.

De Caceres M, Martin-StPaul N, Turco M, Cabon A, Granda V (2018) Estimating daily meteorological data and downscaling climate models over landscapes. Environmental Modelling and Software 108: 186-196.

Examples

```
# example interpolator
data("meteoland_interpolator_example")

# As the cross validation for all stations can be time consuming, we are
# gonna use only for the first 5 stations of the 198
cv <- interpolation_cross_validation(meteoland_interpolator_example, stations = 1:5)

# Inspect the results
cv$errors
cv$station_stats
cv$dates_stats
cv$r2




# example interpolator
data("meteoland_interpolator_example")

# As the calibration for all stations can be time consuming, we are gonna
# interpolate only for the first 5 stations of the 198 and only a handful
# of parameter combinations
```

```
calibration <- interpolator_calibration(
  meteoland_interpolator_example,
  stations = 1:5,
  variable = "MaxTemperature",
  N_seq = seq(10, 20, by = 5),
  alpha_seq = seq(8, 9, by = 0.25)
)

# we can update the interpolator params directly:
updated_interpolator <- interpolator_calibration(
  meteoland_interpolator_example,
  stations = 1:5,
  update_interpolation_params = TRUE,
  variable = "MaxTemperature",
  N_seq = seq(10, 20, by = 5),
  alpha_seq = seq(8, 9, by = 0.25)
)


# check the new interpolator have the parameters updated
get_interpolation_params(updated_interpolator)$N_MaxTemperature
get_interpolation_params(updated_interpolator)$alpha_MaxTemperature
```

`interpolation_precipitation` *Low-level interpolation functions*

Description

Low-level functions to interpolate meteorology (one day) on a set of points.

Usage

```
interpolation_precipitation(
  Xp,
  Yp,
  Zp,
  X,
  Y,
  Z,
  P,
  Psmooth,
  iniRp = 140000,
  alpha_event = 6.25,
  alpha_amount = 6.25,
  N_event = 20L,
  N_amount = 20L,
```

```
  iterations = 3L,
  popcrit = 0.5,
  fmax = 0.95,
  debug = FALSE
)

interpolation_dewtemperature(
  Xp,
  Yp,
  Zp,
  X,
  Y,
  Z,
  T,
  iniRp = 140000,
  alpha = 3,
  N = 30L,
  iterations = 3L,
  debug = FALSE
)

interpolation_temperature(
  Xp,
  Yp,
  Zp,
  X,
  Y,
  Z,
  T,
  iniRp = 140000,
  alpha = 3,
  N = 30L,
  iterations = 3L,
  debug = FALSE
)

interpolation_wind(
  Xp,
  Yp,
  WS,
  WD,
  X,
  Y,
  iniRp = 140000,
  alpha = 2,
  N = 1L,
  iterations = 3L,
  directionsAvailable = TRUE
```

```
)
```

Arguments

- `Xp, Yp, Zp` Spatial coordinates and elevation (Zp; in m.a.s.l) of target points.
- `X, Y, Z` Spatial coordinates and elevation (Zp; in m.a.s.l) of reference locations (e.g. meteorological stations).
- `P` Precipitation at the reference locations (in mm).
- `Psmooth` Temporally-smoothed precipitation at the reference locations (in mm).
- `iniRp` Initial truncation radius.
- `iterations` Number of station density iterations.
- `popcrit` Critical precipitation occurrence parameter.
- `fmax` Maximum value for precipitation regression extrapolations (0.6 equals to a maximum of 4 times extrapolation).
- `debug` Boolean flag to show extra console output.
- `T` Temperature (e.g., minimum, maximum or dew temperature) at the reference locations (in degrees).
- `alpha, alpha_amount, alpha_event` Gaussian shape parameter.
- `N, N_event, N_amount` Average number of stations with non-zero weights.
- `WS, WD` Wind speed (in m/s) and wind direction (in degrees from north clock-wise) at the reference locations.
- `directionsAvailable` A flag to indicate that wind directions are available (i.e. non-missing) at the reference locations.

Details

This functions exposes internal low-level interpolation functions written in C++ not intended to be used directly in any script or function. The are maintained for compatibility with older versions of the package and future versions of meteoland will remove this functions (they will be still accessible through the triple colon notation (`:::`), but their use is not recommended)

Value

All functions return a vector with interpolated values for the target points.

Functions

- `interpolation_precipitation()`: Precipitation
- `interpolation_dewtemperature()`: Dew temperature
- `interpolation_wind()`: Wind

Author(s)

Miquel De Cáceres Ainsa, CREAF

References

Thornton, P.E., Running, S.W., White, M. A., 1997. Generating surfaces of daily meteorological variables over large regions of complex terrain. J. Hydrol. 190, 214–251. doi:10.1016/S0022-1694(96)03128-9.

De Caceres M, Martin-StPaul N, Turco M, Cabon A, Granda V (2018) Estimating daily meteorological data and downscaling climate models over landscapes. Environmental Modelling and Software 108: 186-196.

See Also

defaultInterpolationParams

Examples

```
Xp <- as.numeric(sf::st_coordinates(points_to_interpolate_example)[,1])
Yp <- as.numeric(sf::st_coordinates(points_to_interpolate_example)[,2])
Zp <- points_to_interpolate_example$elevation
X <- as.numeric(
 sf::st_coordinates(stars::st_get_dimension_values(meteoland_interpolator_example, "station"))[,1]
)
Y <- as.numeric(
 sf::st_coordinates(stars::st_get_dimension_values(meteoland_interpolator_example, "station"))[,2]
)
Z <- as.numeric(meteoland_interpolator_example[["elevation"]][1,])
Temp <- as.numeric(meteoland_interpolator_example[["MinTemperature"]][1,])
P <- as.numeric(meteoland_interpolator_example[["Precipitation"]][1,])
Psmooth <- as.numeric(meteoland_interpolator_example[["SmoothedPrecipitation"]][1,])
WS <- as.numeric(meteoland_interpolator_example[["WindSpeed"]][1,])
WD <- as.numeric(meteoland_interpolator_example[["WindDirection"]][1,])
iniRp <- get_interpolation_params(meteoland_interpolator_example)$initial_Rp
alpha <- get_interpolation_params(meteoland_interpolator_example)$alpha_MinTemperature
N <- get_interpolation_params(meteoland_interpolator_example)$N_MinTemperature
alpha_event <- get_interpolation_params(meteoland_interpolator_example)$alpha_PrecipitationEvent
N_event <- get_interpolation_params(meteoland_interpolator_example)$N_PrecipitationEvent
alpha_amount <- get_interpolation_params(meteoland_interpolator_example)$alpha_PrecipitationAmount
N_amount <- get_interpolation_params(meteoland_interpolator_example)$N_PrecipitationAmount
alpha_wind <- get_interpolation_params(meteoland_interpolator_example)$alpha_Wind
N_wind <- get_interpolation_params(meteoland_interpolator_example)$N_Wind
iterations <- get_interpolation_params(meteoland_interpolator_example)$iterations
popcrit <- get_interpolation_params(meteoland_interpolator_example)$pop_crit
fmax <- get_interpolation_params(meteoland_interpolator_example)$f_max
debug <- get_interpolation_params(meteoland_interpolator_example)$debug

interpolation_temperature(
  Xp, Yp, Zp,
  X[!is.na(Temp)], Y[!is.na(Temp)], Z[!is.na(Temp)],
  Temp[!is.na(Temp)],
```

```
  iniRp, alpha, N, iterations, debug
)

interpolation_wind(
  Xp, Yp,
  WS[!is.na(WD)], WD[!is.na(WD)],
  X[!is.na(WD)], Y[!is.na(WD)],
  iniRp, alpha_wind, N_wind, iterations, directionsAvailable = FALSE
)

interpolation_precipitation(
  Xp, Yp, Zp,
  X[!is.na(P)], Y[!is.na(P)], Z[!is.na(P)],
  P[!is.na(P)], Psmooth[!is.na(P)],
  iniRp, alpha_event, alpha_amount, N_event, N_amount,
  iterations, popcrit, fmax, debug
)
```

meteoland_interpolator_example

Example interpolator object

Description

[Experimental]

Example interpolator with daily meteorological records from 189 weather stations in Catalonia (NE Spain) corresponding to April 2022.

Format

stars data cube object

Source

Spanish National Forest Inventory

Examples

```
data(meteoland_interpolator_example)
```

meteoland_meteo_example

Example data set for meteo data from weather stations

Description

[Experimental]

Example data set of spatial location, topography and daily meteorological records from 189 weather stations in Catalonia (NE Spain) corresponding to April 2022.

Format

sf object

Source

'Servei Meteorològic de Catalunya' (SMC)

Examples

```
data(meteoland_meteo_example)
```

meteoland_meteo_no_topo_example

Example data set for meteo data from weather stations, without topography

Description

[Experimental]

Example data set of spatial location and daily meteorological records from 189 weather stations in Catalonia (NE Spain) corresponding to April 2022.

Format

sf object

Source

'Servei Meteorològic de Catalunya' (SMC)

Examples

```
data(meteoland_meteo_no_topo_example)
```

meteoland_topo_example *Example data set for topography data from weather stations, without meteo*

Description

[Experimental]

Example data set of spatial location and topography records from 189 weather stations in Catalonia (NE Spain).

Format

sf object

Source

'Servei Meteorològic de Catalunya' (SMC)

Examples

```
data(meteoland_topo_example)
```

meteospain2meteoland *From meteospain to meteoland meteo objects*

Description

Adapting meteospain meteo objects to meteoland meteo objects

Usage

```
meteospain2meteoland(meteo, complete = FALSE)
```

Arguments

`meteo`	meteospain meteo object.
`complete`	logical indicating if the meteo data missing variables should be calculated (if possible). Default to FALSE.

Details

This function converts `meteospain` R package meteo objects to compatible meteoland meteo objects by selecting the needed variables and adapting the names to comply with meteoland requirements.

Value

a compatible meteo object to use with meteoland.

Examples

```
if (interactive()) {
  # meteospain data
  library(meteospain)
  mg_april_2022_data <- get_meteo_from(
    "meteogalicia",
    meteogalicia_options("daily", as.Date("2022-04-01"), as.Date("2022-04-30"))
  )

  # just convert
  meteospain2meteoland(mg_april_2022_data)
  # convert and complete
  meteospain2meteoland(mg_april_2022_data, complete = TRUE)

}
```

---

penman *Potential evapotranspiration*

Description

Functions to calculate potential evapotranspiration using Penman or Penman-Monteith.

Usage

```
penman(
  latrad,
  elevation,
  slorad,
  asprad,
  J,
  Tmin,
  Tmax,
  RHmin,
  RHmax,
  R_s,
  u,
  z = 10,
  z0 = 0.001,
  alpha = 0.25,
  windfun = "1956"
)

penmanmonteith(rc, elevation, Tmin, Tmax, RHmin, RHmax, Rn, u = NA_real_)
```

Arguments

`latrad`	Latitude in radians.
`elevation`	Elevation (in m).
`slorad`	Slope (in radians).
`asprad`	Aspect (in radians from North).
`J`	Julian day, number of days since January 1, 4713 BCE at noon UTC.
`Tmin`	Minimum temperature (degrees Celsius).
`Tmax`	Maximum temperature (degrees Celsius).
`RHmin`	Minimum relative humidity (percent).
`RHmax`	Maximum relative humidity (percent).
`R_s`	Solar radiation (MJ/m2).
`u`	With wind speed (m/s).
`z`	Wind measuring height (m).
`z0`	Roughness height (m).
`alpha`	Albedo.
`windfun`	Wind speed function version, either "1948" or "1956".
`rc`	Canopy vapour flux (stomatal) resistance (s·m-1).
`Rn`	Daily net radiation (MJ·m-2·day-1).

Details

The code was adapted from package ‘Evapotranspiration’, which follows McMahon et al. (2013). If wind speed is not available, an alternative formulation for potential evapotranspiration is used as an approximation (Valiantzas 2006)

Value

Potential evapotranspiration (in mm of water).

Functions

- `penmanmonteith()`: Penman Monteith method

Author(s)

Miquel De Cáceres Ainsa, CREAF

References

Penman, H. L. 1948. Natural evaporation from open water, bare soil and grass. Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences, 193, 120-145.

Penman, H. L. 1956. Evaporation: An introductory survey. Netherlands Journal of Agricultural Science, 4, 9-29.

McMahon, T.A., Peel, M.C., Lowe, L., Srikanthan, R., McVicar, T.R. 2013. Estimating actual, potential, reference crop and pan evaporation using standard meteorological data: a pragmatic synthesis. Hydrology & Earth System Sciences 17, 1331–1363. doi:10.5194/hess-17-1331-2013.

See Also

`interpolate_data`

---

points_to_interpolate_example

Example data set of points for interpolation of weather variables

---

Description

[Experimental]

Example data set of spatial location and topography records from 15 experimental plots in Catalonia (NE Spain).

Format

sf object

Source

Spanish National Forest Inventory

Examples

```
data(points_to_interpolate_example)
```

---

precipitation_concentration

Precipitation daily concentration

Description

Function `precipitation_concentration()` calculates daily precipitation concentration (Martin-Vide et al. 2004).

Usage

```
precipitation_concentration(p)
```

Arguments

`p`	A numeric vector with daily precipitation values.

Value

Function `precipitation_concentration()` returns a value between 0 (equal distribution of rainfall) and 1 (one day concentrates all rainfall).

Author(s)

Miquel De Cáceres Ainsa, CREAF.

References

Martin-Vide J (2004) Spatial distribution of a daily precipitation concentration index in peninsular Spain. International Journal of Climatology 24, 959–971. doi:10.1002/joc.1030.

---

precipitation_rainfall_erosivity

Precipitation rainfall erosivity

Description

[Experimental]

Function `precipitation_rainfall_erosivity()` calculates a multi-year average of monthly rainfall erosivity using the MedREM model proposed by Diodato and Bellochi (2010) for the Mediterranean area (see also Guerra et al. 2016).

Usage

```
precipitation_rainfall_erosivity(
  meteo_data,
  longitude,
  scale = c("month", "year"),
  average = TRUE
)
```

Arguments

`meteo_data`	A meteo tibble as with the dates and meteorological variables as returned by `interpolate_data` in the "interpolated_data" column.
`longitude`	Longitude in degrees.
`scale`	Character, either 'month' or 'year'. Default to 'month'
`average`	Boolean flag to calculate multi-year averages before applying MedREM's formula.

Details

MedREM model is: Rm = b0·P·sqrt(d)·(alpha + b1*longitude), where P is accumulated precipitation and d is maximum daily precipitation. Parameters used for the MedREM model are b0 = 0.117, b1 = -0.015, alpha = 2. Note that there is a mistake in Guerra et al. (2016) regarding parameters b1 and a.

Value

A vector of values for each month (in MJ·mm·ha-1·h-1·month-1) or each year (in MJ·mm·ha-1·h-1·yr-1), depending on the scale

Author(s)

Miquel De Cáceres Ainsa, CREAF.

Víctor Granda García, CREAF.

References

Diodato, N., Bellocchi, G., 2010. MedREM, a rainfall erosivity model for the Mediterranean region. J. Hydrol. 387, 119–127, doi:10.1016/j.jhydrol.2010.04.003.

Guerra CA, Maes J, Geijzendorffer I, Metzger MJ (2016) An assessment of soil erosion prevention by vegetation in Mediterranean Europe: Current trends of ecosystem service provision. Ecol Indic 60:213–222. doi: 10.1016/j.ecolind.2015.06.043.

Examples

```
interpolated_example <-
  interpolate_data(points_to_interpolate_example, meteoland_interpolator_example)

precipitation_rainfall_erosivity(
  meteo_data = interpolated_example$interpolated_data[[1]],
  longitude = 2.32,
  scale = "month",
  average = TRUE
)
```

---

radiation_julianDay *Solar radiation utility functions*

Description

Set of functions used in the calculation of incoming solar radiation and net radiation.

Usage

```
radiation_julianDay(year, month, day)

radiation_dateStringToJulianDays(dateStrings)

radiation_solarDeclination(J)

radiation_solarConstant(J)
```

```
radiation_sunRiseSet(latrad, slorad, asprad, delta)

radiation_solarElevation(latrad, delta, hrad)

radiation_daylength(latrad, slorad, asprad, delta)

radiation_daylengthseconds(latrad, slorad, asprad, delta)

radiation_potentialRadiation(solarConstant, latrad, slorad, asprad, delta)

radiation_solarRadiation(
  solarConstant,
  latrad,
  elevation,
  slorad,
  asprad,
  delta,
  diffTemp,
  diffTempMonth,
  vpa,
  precipitation
)

radiation_directDiffuseInstant(
  solarConstant,
  latrad,
  slorad,
  asprad,
  delta,
  hrad,
  R_s,
  clearday
)

radiation_directDiffuseDay(
  solarConstant,
  latrad,
  slorad,
  asprad,
  delta,
  R_s,
  clearday,
  nsteps = 24L
)

radiation_skyLongwaveRadiation(Tair, vpa, c = 0)
```

```
radiation_outgoingLongwaveRadiation(
  solarConstant,
  latrad,
  elevation,
  slorad,
  asprad,
  delta,
  vpa,
  tmin,
  tmax,
  R_s
)

radiation_netRadiation(
  solarConstant,
  latrad,
  elevation,
  slorad,
  asprad,
  delta,
  vpa,
  tmin,
  tmax,
  R_s,
  alpha = 0.08
)
```

Arguments

year, month, day	Year, month and day as integers.
dateStrings	A character vector with dates in format "YYYY-MM-DD".
J	Julian day (integer), number of days since January 1, 4713 BCE at noon UTC.
latrad	Latitude (in radians North).
slorad	Slope (in radians).
asprad	Aspect (in radians from North).
delta	Solar declination (in radians).
hrad	Solar hour (in radians).
solarConstant	Solar constant (in kW·m-2).
elevation	Elevation above sea level (in m).
diffTemp	Difference between maximum and minimum temperature (ºC).
diffTempMonth	Difference between maximum and minimum temperature, averaged over 30 days (ºC).
vpa	Average daily vapor pressure (kPa).
precipitation	Precipitation (in mm).
R_s	Daily incident solar radiation (MJ·m-2).

clearday	Boolean flag to indicate a clearsky day (vs. overcast).
nsteps	Number of daily substeps.
Tair	Air temperature (in degrees Celsius).
c	Proportion of sky covered by clouds (0-1).
tmin, tmax	Minimum and maximum daily temperature (ºC).
alpha	Surface albedo (from 0 to 1).

Value

Values returned for each function are:

- radiation_dateStringToJulianDays: A vector of Julian days (i.e. number of days since January 1, 4713 BCE at noon UTC).
- radiation_daylength: Day length (in hours).
- radiation_daylengthseconds: Day length (in seconds).
- radiation_directDiffuseInstant: A vector with instantaneous direct and diffusive radiation rates (for both SWR and PAR).
- radiation_directDiffuseDay: A data frame with instantaneous direct and diffusive radiation rates (for both SWR and PAR) for each subdaily time step.
- radiation_potentialRadiation: Daily (potential) solar radiation (in MJ·m-2).
- radiation_julianDay: Number of days since January 1, 4713 BCE at noon UTC.
- radiation_skyLongwaveRadiation: Instantaneous incoming (sky) longwave radiation (W·m-2).
- radiation_outgoingLongwaveRadiation: Daily outgoing longwave radiation (MJ·m-2·day-1).
- radiation_netRadiation: Daily net solar radiation (MJ·m-2·day-1).
- radiation_solarConstant: Solar constant (in kW·m-2).
- radiation_solarDeclination: Solar declination (in radians).
- radiation_solarElevation: Angle of elevation of the sun with respect to the horizon (in radians).
- radiation_solarRadiation: Daily incident solar radiation (MJ·m-2·day-1).
- radiation_sunRiseSet: Sunrise and sunset hours in hour angle (radians).

Functions

- radiation_dateStringToJulianDays(): Date string to julian days
- radiation_solarDeclination(): solar declination
- radiation_solarConstant(): solar constant
- radiation_sunRiseSet(): sun rise and set
- radiation_solarElevation(): solar elevation
- radiation_daylength(): Day length
- radiation_daylengthseconds(): Day length seconds

- `radiation_potentialRadiation()`: Potential radiation
- `radiation_solarRadiation()`: solar Radiation
- `radiation_directDiffuseInstant()`: Direct diffuse instant
- `radiation_directDiffuseDay()`: Direct diffuse day
- `radiation_skyLongwaveRadiation()`: Sky longwave radiation
- `radiation_outgoingLongwaveRadiation()`: Outgoing longwave radiation
- `radiation_netRadiation()`: Net radiation

Note

Code for `radiation_julianDay()`, `radiation_solarConstant()` and `radiation_solarDeclination()` was translated to C++ from R code in package 'insol' (by J. G. Corripio).

Author(s)

Miquel De Cáceres Ainsa, CREAF

References

Danby, J. M. Eqn. 6.16.4 in Fundamentals of Celestial Mechanics, 2nd ed. Richmond, VA: Willmann-Bell, p. 207, 1988.

Garnier, B.J., Ohmura, A., 1968. A method of calculating the direct shortwave radiation income of slopes. J. Appl. Meteorol. 7: 796-800

McMahon, T. A., M. C. Peel, L. Lowe, R. Srikanthan, and T. R. McVicar. 2013. Estimating actual, potential, reference crop and pan evaporation using standard meteorological data: a pragmatic synthesis. Hydrology & Earth System Sciences 17:1331–1363. See also: http://www.fao.org/docrep/x0490e/x0490e06.htm.

Reda, I. and Andreas, A. 2003. Solar Position Algorithm for Solar Radiation Applications. 55 pp.; NREL Report No. TP-560-34302, Revised January 2008. http://www.nrel.gov/docs/fy08osti/34302.pdf

Spitters, C.J.T., Toussaint, H.A.J.M. and Goudriaan, J. (1986). Separating the diffuse and direct components of global radiation and its implications for modeling canopy photosynthesis. I. Components of incoming radiation. Agricultural and Forest Meteorology, 38, 231–242.

See Also

`interpolate_data`

`raster_to_interpolate_example` *Example raster data set for interpolation of weather variables*

Description

[Experimental]

Example raster data set of spatial location and topography records from Catalonia (NE Spain). Cell size is 1km x 1km and raster size is 10x10 cells.

Format

stars object

Source

ICGC

Examples

```
data(raster_to_interpolate_example)
```

---

read_interpolator *Read interpolator files*

Description

Read interpolator files created with `write_interpolator`

Usage

```
read_interpolator(filename)
```

Arguments

`filename`	interpolator file name

Details

This function takes the file name of the nc file storing an interpolator object and load it into the work environment

Value

an interpolator (stars) object

Author(s)

Victor Granda García, EMF-CREAF

See Also

Other interpolator functions: `add_topo()`, `create_meteo_interpolator()`, `get_interpolation_params()`, `set_interpolation_params()`, `with_meteo()`, `write_interpolator()`

Examples

```
# example interpolator
data(meteoland_interpolator_example)

# temporal folder
tmp_dir <- tempdir()

# write interpolator
write_interpolator(
  meteoland_interpolator_example,
  file.path(tmp_dir, "meteoland_interpolator_example.nc")
)

# check file exists
file.exists(file.path(tmp_dir, "meteoland_interpolator_example.nc"))

# read it again
read_interpolator(file.path(tmp_dir, "meteoland_interpolator_example.nc"))
```

---

set_interpolation_params

Setting interpolation parameters in an interpolator object

Description

Changing or updating interpolation parameters in an interpolator object

Usage

```
set_interpolation_params(
  interpolator,
  params = NULL,
  verbose = getOption("meteoland_verbosity", TRUE)
)
```

Arguments

interpolator	interpolator object to update
params	list with the parameters provided by the user
verbose	Logical indicating if the function must show messages and info. Default value checks "meteoland_verbosity" option and if not set, defaults to TRUE. It can be turned off for the function with FALSE, or session wide with options(meteoland_verbosity = FALSE)

Details

This function ensures that if no parameters are provided, the default ones are used (see `defaultInterpolationParams`). Also, if params are partially provided, this function ensures that the rest of the parameters are not changed.

Value

The same interpolator object provided, with the updated interpolation parameters

Author(s)

Victor Granda García, EMF-CREAF

See Also

Other interpolator functions: `add_topo()`, `create_meteo_interpolator()`, `get_interpolation_params()`, `read_interpolator()`, `with_meteo()`, `write_interpolator()`

Examples

```
# example interpolator
data(meteoland_interpolator_example)
# store the actual parameters
old_parameters <- get_interpolation_params(meteoland_interpolator_example)
# we can provide only the parameter we want to change
meteoland_interpolator_example <- set_interpolation_params(
  meteoland_interpolator_example,
  list(debug = TRUE)
)
# check
get_interpolation_params(meteoland_interpolator_example)$debug
# compare with old
old_parameters$debug
# the rest should be the same
setdiff(old_parameters, get_interpolation_params(meteoland_interpolator_example))
```

---

summarise_interpolated_data

Summarise interpolated data by temporal dimension

Description

[Experimental]

Summarises the interpolated meteorology in one or more locations by the desired temporal scale

Usage

```
summarise_interpolated_data(
  interpolated_data,
  fun = "mean",
  frequency = NULL,
  vars_to_summary = c("MeanTemperature", "MinTemperature", "MaxTemperature",
    "Precipitation", "MeanRelativeHumidity", "MinRelativeHumidity",
    "MaxRelativeHumidity", "Radiation", "WindSpeed", "WindDirection", "PET"),
  dates_to_summary = NULL,
  months_to_summary = 1:12,
  verbose = getOption("meteoland_verbosity", TRUE),
  ...
)
```

Arguments

`interpolated_data`
: An interpolated data object as returned by `interpolate_data`.

`fun`
: The function to use for summarising the data.

`frequency`
: A string indicating the interval specification (allowed ones are "week", "month", "quarter" and "year"). If NULL (default), aggregation is done in one interval for all the dates present.

`vars_to_summary`
: A character vector with one or more variable names to summarise. By default, all interpolated variables are summarised.

`dates_to_summary`
: A Date object to define the dates to be summarised. If NULL (default), all dates in the interpolated data are processed.

`months_to_summary`
: A numeric vector with the month numbers to subset the data before summarising. (e.g. `c(7,8)` for July and August). This parameter allows studying particular seasons, when combined with `frequency`. For example `frequency = "years"` and `months = 6:8` leads to summarizing summer months of each year.

`verbose`
: Logical indicating if the function must show messages and info. Default value checks `"meteoland_verbosity"` option and if not set, defaults to TRUE. It can be turned off for the function with FALSE, or session wide with `options(meteoland_verbosity = FALSE)`

`...`
: Arguments needed for `fun`

Details

If `interpolated_data` is a nested interpolated data sf object, as returned by `interpolate_data`, temporal summary is done for each location present in the interpolated data. If `interpolated_data` is an unnested interpolated data sf object, temporal summary is done for all locations together. If `interpolated_data` is a single location data.frame containing the dates and the interpolated variables, temporal summary is done for that location. If `interpolated_data` is a stars object as returned by `interpolate_data`, temporal summary is done for all the raster.

Value

For a nested interpolated data, the same sf object with a new column with the temporal summaries. For an unnested interpolated data, a data.frame with the summarised meteo variables. For an interpolated raster (stars object), the same raster with the temporal dimension aggregated as desired.

Author(s)

Víctor Granda García, CREAF

Examples

```
# points interpolation aggregation
points_to_interpolate_example |>
  interpolate_data(meteoland_interpolator_example, verbose = FALSE) |>
  summarise_interpolated_data()

# raster interpolation aggregation
raster_to_interpolate_example |>
  interpolate_data(meteoland_interpolator_example, verbose = FALSE) |>
  summarise_interpolated_data()
```

---

summarise_interpolator

Summarise interpolator objects by temporal dimension

Description

[Experimental]

Summarises an interpolator object by the desired temporal scale.

Usage

```
summarise_interpolator(
  interpolator,
  fun = "mean",
  frequency = NULL,
  vars_to_summary = c("Temperature", "MinTemperature", "MaxTemperature", "Precipitation",
    "RelativeHumidity", "MinRelativeHumidity", "MaxRelativeHumidity", "Radiation",
    "WindSpeed", "WindDirection", "PET", "SmoothedPrecipitation",
    "SmoothedTemperatureRange", "elevation", "slope", "aspect"),
  dates_to_summary = NULL,
  months_to_summary = 1:12,
  verbose = getOption("meteoland_verbosity", TRUE),
  ...
)
```

Arguments

- `interpolator` — An interpolator object as created by `create_meteo_interpolator`.
- `fun` — The function to use for summarising the data.
- `frequency` — A string indicating the interval specification (allowed ones are "week", "month", "quarter" and "year"). If NULL (default), aggregation is done in one interval for all the dates present.
- `vars_to_summary` — A character vector with one or more variable names to summarise. By default, all interpolated variables are summarised.
- `dates_to_summary` — A Date object to define the dates to be summarised. If NULL (default), all dates in the interpolated data are processed.
- `months_to_summary` — A numeric vector with the month numbers to subset the data before summarising. (e.g. `c(7,8)` for July and August). This parameter allows studying particular seasons, when combined with `frequency`. For example `frequency = "years"` and `months = 6:8` leads to summarizing summer months of each year.
- `verbose` — Logical indicating if the function must show messages and info. Default value checks `"meteoland_verbosity"` option and if not set, defaults to TRUE. It can be turned off for the function with FALSE, or session wide with `options(meteoland_verbosity = FALSE)`
- `...` — Arguments needed for `fun`

Value

`summarise_interpolator` function returns the same interpolator object provided with the temporal dimension aggregated to desired frequency.

Author(s)

Víctor Granda García, CREAF

Examples

```
# example interpolator
meteoland_interpolator_example

# aggregate all dates in the interpolator, calculating the maximum values
summarise_interpolator(meteoland_interpolator_example, fun = "max")

# aggregate weekly, calculating mean values
summarise_interpolator(meteoland_interpolator_example, frequency = "week")
```

utils_saturationVP *Physical utility functions*

Description

Set of functions used in the calculation of physical variables.

Usage

```
utils_saturationVP(temperature)

utils_averageDailyVP(Tmin, Tmax, RHmin, RHmax)

utils_atmosphericPressure(elevation)

utils_airDensity(temperature, Patm)

utils_averageDaylightTemperature(Tmin, Tmax)

utils_latentHeatVaporisation(temperature)

utils_latentHeatVaporisationMol(temperature)

utils_psychrometricConstant(temperature, Patm)

utils_saturationVaporPressureCurveSlope(temperature)
```

Arguments

`temperature`	Air temperature (ºC).
`Tmin`, `Tmax`	Minimum and maximum daily temperature (ºC).
`RHmin`, `RHmax`	Minimum and maximum relative humidity (%).
`elevation`	Elevation above sea level (in m).
`Patm`	Atmospheric air pressure (in kPa).

Value

Values returned for each function are:

- `utils_airDensity`: air density (in kg·m-3).
- `utils_atmosphericPressure`: Air atmospheric pressure (in kPa).
- `utils_averageDailyVP`: average (actual) vapour pressure (in kPa).
- `utils_averageDaylightTemperature`: average daylight air temperature (in ºC). `utils_latentHeatVaporisation`: Latent heat of vaporisation (MJ·kg-1). `utils_latentHeatVaporisationMol`: Latent heat of vaporisation (J·mol-1).

- `utils_psychrometricConstant`: Psychrometric constant (kPa·°C-1).
- `utils_saturationVP`: saturation vapour pressure (in kPa).
- `utils_saturationVaporPressureCurveSlope`: Slope of the saturation vapor pressure curve (kPa·°C-1).

Functions

- `utils_averageDailyVP()`: Average daily VP
- `utils_atmosphericPressure()`: Atmospheric pressure
- `utils_airDensity()`: Air density
- `utils_averageDaylightTemperature()`: Daylight temperature
- `utils_latentHeatVaporisation()`: latent heat vaporisation
- `utils_latentHeatVaporisationMol()`: Heat vaporisation mol
- `utils_psychrometricConstant()`: psychrometric constant
- `utils_saturationVaporPressureCurveSlope()`: Saturation VP curve slope

Author(s)

Miquel De Cáceres Ainsa, CREAF

References

McMurtrie, R. E., D. A. Rook, and F. M. Kelliher. 1990. Modelling the yield of Pinus radiata on a site limited by water and nitrogen. Forest Ecology and Management 30:381–413.

McMahon, T. A., M. C. Peel, L. Lowe, R. Srikanthan, and T. R. McVicar. 2013. Estimating actual, potential, reference crop and pan evaporation using standard meteorological data: a pragmatic synthesis. Hydrology & Earth System Sciences 17:1331–1363. See also: http://www.fao.org/docrep/x0490e/x0490e06.htm

---

`with_meteo` *Ensure meteo object is ready to create an interpolator object*

Description

Check integrity of meteo objects

Usage

```
with_meteo(meteo, verbose = getOption("meteoland_verbosity", TRUE))
```

Arguments

`meteo`	meteo object
`verbose`	Logical indicating if the function must show messages and info. Default value checks `"meteoland_verbosity"` option and if not set, defaults to TRUE. It can be turned off for the function with FALSE, or session wide with `options(meteoland_verbosity = FALSE)`

Details

This function is the first step in the creation of a meteoland interpolator, ensuring the meteo provided contains all the required elements

Value

invisible meteo object ready to pipe in the interpolator creation

See Also

Other interpolator functions: `add_topo()`, `create_meteo_interpolator()`, `get_interpolation_params()`, `read_interpolator()`, `set_interpolation_params()`, `write_interpolator()`

Examples

```
# example meteo
data(meteoland_meteo_example)
with_meteo(meteoland_meteo_example)
```

---

worldmet2meteoland *From worldmet to meteoland meteo objects*

Description

Adapting `importNOAA` meteo objects to meteoland meteo objects

Usage

```
worldmet2meteoland(meteo, complete = FALSE)
```

Arguments

`meteo`	worldmet meteo object.
`complete`	logical indicating if the meteo data missing variables should be calculated (if possible). Default to FALSE.

Details

This function converts `importNOAA` meteo objects to compatible meteoland meteo objects by selecting the needed variables and adapting the names to comply with meteoland requirements. Also it aggregates subdaily data as well as complete missing variables if possible (setting `complete = TRUE`)

Value

a compatible meteo object to use with meteoland.

Examples

```
if (interactive()) {
  # worldmet data
  library(worldmet)
  worldmet_stations <- worldmet::getMeta(lat = 42, lon = 0, n = 2, plot = FALSE)
  worldmet_subdaily_2022 <-
    worldmet::importNOAA(worldmet_stations$code, year = 2022, hourly = TRUE)

  # just convert
  worldmet2meteoland(worldmet_subdaily_2022)
  # convert and complete
  worldmet2meteoland(worldmet_subdaily_2022, complete = TRUE)

}
```

`write_interpolator` *Write the interpolator object*

Description

Write the interpolator object to a file

Usage

```
write_interpolator(interpolator, filename, .overwrite = FALSE)
```

Arguments

`interpolator`	meteoland interpolator object, as created by `create_meteo_interpolator`
`filename`	file name for the interpolator nc file
`.overwrite`	logical indicating if the file should be overwritten if it already exists

Details

This function writes the interpolator object created with `create_meteo_interpolator` in a NetCDF-CF standard compliant format, as specified in https://cfconventions.org/cf-conventions/cf-conventions.html

Value

invisible interpolator object, to allow using this function as a step in a pipe

Author(s)

Victor Granda García, EMF-CREAF

See Also

Other interpolator functions: add_topo(), create_meteo_interpolator(), get_interpolation_params(), read_interpolator(), set_interpolation_params(), with_meteo()

Examples

```
# example interpolator
data(meteoland_interpolator_example)

# temporal folder
tmp_dir <- tempdir()

# write interpolator
write_interpolator(
  meteoland_interpolator_example,
  file.path(tmp_dir, "meteoland_interpolator_example.nc"),
  .overwrite = TRUE
)

# check file exists
file.exists(file.path(tmp_dir, "meteoland_interpolator_example.nc"))

# read it again
read_interpolator(file.path(tmp_dir, "meteoland_interpolator_example.nc"))
```

Index