Package 'simulariatools'

November 8, 2023

Type Package

Title Simularia Tools for the Analysis of Air Pollution Data

Version 2.5.1

Maintainer Giuseppe Carlino <g.carlino@simularia.it>

Description A set of tools developed at Simularia for Simularia, to help preprocessing and post-processing of meteorological and air quality data.

Depends R (>= 3.3)

Imports ggplot2 (>= 3.3), lubridate, reshape2, reticulate, scales, terra

Suggests magick, contoureR, testthat (>= 3.0.0), raster, openair

License GPL (>= 2)

URL https://www.simularia.it/simulariatools/, https://github.com/Simularia/simulariatools

BugReports https://github.com/Simularia/simulariatools/issues

LazyLoad yes

LazyData yes

Encoding UTF-8

RoxygenNote 7.2.3

Config/testthat/edition 3

NeedsCompilation no

Author Giuseppe Carlino [aut, cre], Matteo Paolo Costa [ctb], Simularia [cph, fnd]

Repository CRAN

Date/Publication 2023-11-08 14:10:02 UTC

R topics documented:

`contourPlot` *Contour plot of pollutant concentration*

Description

`contourPlot` plots a contour map of pollutants. This function has been deprecated since version 2.0.0 and will be removed very soon. Use `contourPlot2`.

Usage

```
contourPlot(
  data,
  domain = NULL,
  background = NULL,
  underlayer = NULL,
  overlayer = NULL,
  legend = NULL,
  levels = NULL,
  size = 0,
  cover = TRUE,
  transparency = 0.66,
  smoothness = 1,
  colors = NULL,
  bare = FALSE
)
```

Arguments

data	A dataframe containing data to be plotted in the form of X, Y and Z (levels).
domain	An array with min X, max X, min Y, max Y, number of ticks on X axis, number of ticks on Y axis (optional).
background	String containing the path to the png file to be plotted as a basemap (optional).
underlayer	Array of strings containing layers to be plotted between basemap and contour plot (optional).
overlayer	Array of strings containing layers to be plotted on top of the contour plot (optional).
legend	(string) Legend title (optional).
levels	Array of levels for contour plot. If not set, automatic levels are plotted.
size	float with the thickness of the contour line.
cover	boolean (default TRUE) to specify whether the contour plot should be filled or not.
transparency	float (between 0 and 1, default=0.66). Transparency level of the contour plot.
smoothness	integer factor to improve the horizontal resolution (smaller cells) by bilinear interpolation.
colors	Color palette for contour plot
bare	Boolean (default FALSE) parameter to completely remove axis, legend, titles and any other graphical element from the plot.

Details

This is a convenience function to plot contour levels of a pollutant matrix with ggplot2.

Value

A ggplot2 plot.

Examples

```
## Not run:
# Load example data in long format
data(volcano)
volcano3d <- reshape2::melt(volcano)
names(volcano3d) <- c("x", "y", "z")
# Contour plot with default options
contourPlot(volcano3d)

# Import variable CONCAN from inpufile, convert km to m (k = 1000):
data <- importRaster(paste0(dir, inputfile),
                     k = 1000,
                     variable = "CONCAN")

# Simple contour plot
contourPlot(data)
```

```
# Specifiy (sub)domain to be plotted; background image; legend title and
# pollutant levels.
contourPlot(data,
            domain(500000, 510000, 6000000, 6010000, 7, 7),
            background = "img/background.png",
            legend = "no2 [ug/m3]",
            levels = c(10, 20, 30, 40))

# Add underlayer (same for overlayer)
library(ggplot2)
library(maptools)
perimetro <- readShapeLines("path_to/perimetro.shp")
perimetro <- fortify(perimetro)
strada <- readShapeLines("path_to/strada.shp")
strada <- fortify(strada)
myUnderlayer <- vector(mode = "list", length = 2)
myUnderlayer[[1]] <- geom_polygon(data = perimetro,
                                  aes(long, lat, group = group),
                                  colour = "black",
                                  fill = NA,
                                  size = 0.1,
                                  alpha = 0.5)
myUnderlayer[[2]] <- geom_path(data = strada,
                               aes(long, lat, group = group),
                               colour = "grey",
                               size = 0.1,
                               alpha = 0.5)
contourPlot(data = test,
            background = "path_to/basemap.png",
            underlayer = myUnderlayer)

# Change default colour palette
contourPlot(data = test,
            colors = RColorBrewer::brewer.pal(3, name = "PiYG"))

## End(Not run)
```

`contourPlot2` *New contour plot of pollutant concentration field*

Description

`contourPlot2` plots a contour map of a given quantity, such as the ground concentration of an airborne pollutant or odour, defined on a regular grid.

Usage

```
contourPlot2(
```

```
  data,
  x = "x",
  y = "y",
  z = "z",
  domain = NULL,
  background = NULL,
  underlayer = NULL,
  overlayer = NULL,
  legend = NULL,
  levels = NULL,
  size = 0,
  fill = TRUE,
  tile = FALSE,
  transparency = 0.75,
  colors = NULL,
  bare = FALSE
)
```

Arguments

`data`	dataframe in long format, with three columns for Easting, Northing and values to be plotted.
`x`	name of the column with Easting data (default "x").
`y`	name of the column with Northing data (default "y").
`z`	name of the column with the values to be plotted (default "z").
`domain`	optional list with six numeric values defining the boundaries of the domain to be plotted: minimum X, maximum X, minimum Y, maximum Y, number of ticks on X axis, number of ticks on Y axis.
`background`	optional path to a png file to be plotted as the base map.
`underlayer`	optional list of layers to be plotted between base map and contour plot.
`overlayer`	optional list of layers to be plotted on top of the contour plot.
`legend`	optional title of the legend.
`levels`	numeric vector of levels for contour plot. If not set, automatic pretty levels are computed. If `-Inf` and `Inf` are used as the lowest and highest limits of the array, the lowest and highest bands are unbounded and the legend shows < and >= symbols.
`size`	thickness of the contour line.
`fill`	boolean (default TRUE). If TRUE the contour plot is filled with colour.
`tile`	boolean (default FALSE). If TRUE rectangular tiles are plotted.
`transparency`	transparency level of the contour plot between 0.0 (fully transparent) and 1.0 (fully opaque). Default = 0.75).
`colors`	colour palette for contour plot, as an array of colours.
`bare`	boolean (default FALSE). If TRUE only the bare plot is shown: axis, legend, titles and any other graphical element of the plot are removed.

Details

This is a convenience function to plot contour levels of a scalar quantity such as pollutants computed by a dispersion model, with ggplot2 version >= 3.3.0.

Data are required to be on a regular grid, typically (but not necessarily) in UTM coordinates. The input dataframe has to be in long format, i.e. one line per value to be plotted. The names of the columns corresponding to x, y and z can be specified in the input parameters.

If tile = TRUE data are shown as they are, without any graphical interpolation required for contour plots. This is helpful when you want to visualise the raw data. Since version 2.4.0, when tile = TRUE the intervals include the lowest bound and exclude the highest bound: [min, max). Note: In previous version it was the opposite.

underlayer and overlayer layers are ggplot2 objects to be shown at different levels of the vertical stack of the plot. These are useful to show topographical information related to the plot, such as sources or receptors locations.

Value

A ggplot2 object.

Examples

```
# Load example data in long format
data(volcano)
volcano3d <- reshape2::melt(volcano)
names(volcano3d) <- c("x", "y", "z")
# Contour plot with default options
v <- contourPlot2(volcano3d)
v

# Set levels, and properly format the legend title:
contourPlot2(volcano3d,
             levels = c(-Inf, seq(100, 200, 20), Inf),
             legend = expression(PM[10]~"["~mu*g~m^-3~"]"))

# Sometimes, instead of a contour plot it is better to plot the original
# raster data, without any interpolation:
contourPlot2(volcano3d,
             levels = c(-Inf, seq(100, 200, 20), Inf),
             tile = TRUE)

# Since contourPlot2 returns a `ggplot2` object, you can add instructions as:
library(ggplot2)
v + ggtitle("Example volcano data") +
    labs(x = NULL, y = NULL)
```

createBaseMap *Create base map (OBSOLETE)*

Description

Create base map. This is meant to be the deepest layer of contour plot map. Axes coordinates are supposed to be in meters.

Usage

```
createBaseMap(
  imageFile,
  domain = c(0, 0, 1000, 1000, 5, 5),
  font_size = 10,
  font_family = "sans"
)
```

Arguments

imageFile	(string) Path to the background 'png' file.
domain	Six components vector with the domain SW corner coordinates, the X and Y extensions, and the number of breaks along the to axis (X, Y, DX, DY, NX, NY)
font_size	This is the font size for axis labels
font_family	This is the font family for labels

Value

A ggplot2 plot.

Examples

```
## Not run:
# Import image 'img'. Divide the axis with 9 ticks.
v <- createBaseMap(img, c(minx, miny, extent, extent, 9, 9), font_size=10)

## End(Not run)
```

`downloadBasemap` *Download basemap from Italian National Geoportal*

Description

This function tries to download the aerial orthophoto of the requested domain from the Italian National Geoportal. The output is given in *png* format at the path given in the `file` parameter.

Usage

```
downloadBasemap(
  file = file,
  xSW = 410000,
  ySW = 5000500,
  xExt = 5000,
  yExt = 5000,
  crs = 32,
  width = 1024,
  height = 1024,
  units = "px",
  res = 72
)
```

Arguments

`file`	Path to output file.
`xSW`	South West Easting UTM coordinate of the basemap (in metres).
`ySW`	South West Northing UTM coordinate of the basemap (in metres).
`xExt`	Easting extension in metres.
`yExt`	Northing extension in metres.
`crs`	UTM Coordinate Reference System: either 32 or 33.
`width`	The basemap width.
`height`	The basemap height.
`units`	The unit of measure of width and height. It can be `px` (pixels, the default), `in` (inches), `cm` or `mm`
`res`	The resolution in dpi.

Value

No value is returned.

Examples

```
## Not run:
# Download a basemap of a domain with SW coordinates (410000, 5000500)
# in the UTM32 CRS and extension 5000m in both directions.

downloadBasemap(file = "./basemap.png",
                xSW = 410000, ySW = 5000500, xExt = 5000, yExt = 5000)

# Download a basemap of a domain with SW coordinates (410000, 5000500)
# in the UTM32 CRS and extension 5000m in both directions.
# The file has to be 2048 x 2048 pixels.

downloadBasemap(file = "./basemap.png",
                xSW = 410000, ySW = 5000500, xExt = 5000, yExt = 5000,
                width = 2048, height = 2048)

# Download a basemap of a domain with SW coordinates (410000, 5000500)
# in the UTM32 CRS and extension 5000m in both directions.
# The file has to be 10cm x 10cm with a resolution of 150 dpi.

downloadBasemap(file = "./basemap.png",
                xSW = 410000, ySW = 5000500, xExt = 5000, yExt = 5000,
                width = 10, height = 10, units = "cm", res = 150)

## End(Not run)
```

---

importADSOBIN *ADSO/BIN data import function*

Description

Import data from ADSO/BIN binary file. It requires an active Python installation with the `arinfopy` library.

Usage

```
importADSOBIN(
  file = file.choose(),
  variable = NULL,
  slice = 1,
  deadline = 1,
  k = 1,
  kz = 1,
  dx = 0,
  dy = 0,
  destaggering = FALSE,
  raster.object = FALSE,
  verbose = FALSE
)
```

Arguments

file	The ADSO/BIN file to be imported.
variable	A string with the name of the variable to be imported.
slice	An integer corresponding to the horizontal slice (vertical level) of 3D variables (default = 1). In the case of a 2D variable, it is ignored.
deadline	An integer representing the temporal deadline (default = 1). It can optionally be a string with date time (see examples).
k	A numeric factor to be applied to x and y coordinates (default = 1).
kz	A numeric factor to be applied to z values to rescale them (default = 1).
dx	A number to shift x coordinates by dx (default = 0).
dy	A number to shift y coordinates by dy (default = 0).
destaggering	Use TRUE to apply destaggering to X and Y coordinates (default = FALSE).
raster.object	Use TRUE to return a raster object instead of a dataframe with (X, Y, Z) columns (default = FALSE).
verbose	Use TRUE to print out basic statistics (default = FALSE).

Details

The importADSIOBIN() function was developed to import data from an ADSO/BIN binary file. It relies on the 'arinfopy' (version >= 2.2.0) python library. For more information on the library see the GitHub repository.

For more information on the active python installation, check the documentation of **reticulate**.

Value

In standard use, importADSOBIN() return a data frame with (X, Y, Z) columns. Column Z contains the values of the requested variable. If the raster.object option is set, it returns a RasterLayer object.

See Also

importRaster(), importSurferGrd()

Examples

```
## Not run:
# Read ground level (slice = 1) value of variable M001S001.
pm10 <- importADSOBIN(file = "average_2018.bin",
                      variable = "M001S001",
                      slice = 1)

# Read deadline 12 of the second vertical level of temperature:
temperature <- importADSOBIN(file = "swift_surfpro_01-10_01_2018",
                             variable = "TEMPK",
                             slice = 2,
```

```
                              deadline = 12)

# Read varibale M001S001 at ground level, at given date and time,
# and print basic information:
nox <- importADSOBIN(file = "conc_01-10_07_2018",
                     variable = "M001S001",
                     slice = 1,
                     deadline = "2018/07/02 12:00",
                     verbose = TRUE)

## End(Not run)
```

`importRaster` *Import generic raster file*

Description

The function import the first layer of a generic raster file. Data are imported as an array of x, y, z columns.

Usage

```
importRaster(
  file = file.choose(),
  k = 1,
  kz = 1,
  dx = 0,
  dy = 0,
  destaggering = FALSE,
  variable = NULL,
  verbose = FALSE
)
```

Arguments

`file`	The raster file to be imported.
`k`	A numerical factor to be applied to x and y coordinates (default = 1).
`kz`	A numerical factor to be applied to z values (default = 1).
`dx`	Shifts x coordinates by dx (default = 0).
`dy`	float. Shift y coordinates by dy (default = 0).
`destaggering`	Use `TRUE` to apply destaggering to X and Y coordinates (default = FALSE).
`variable`	The name of the variable to be imported.
`verbose`	If `TRUE`, prints out basic statistics (default = FALSE).

Details

Supported files include those managed by the **raster** package (as netcdf),

Destaggering is useful for importing data from the SPRAY model and it is not applied by default.

An optional summary output can be printed by setting the `verbose` parameter.

This function is based on the `terra` package and it can import any format managed by it.

Value

It returns a dataframe with x, y and z columns.

See Also

`importADSOBIN()`, `importSurferGrd()`

Examples

```
## Not run:
# Import binary (netcdf) file and convert coordinates from km to m,
# without destaggering:
mydata <- importRaster(file = "/path_to_file/filename.nc",
                       k = 1000,
                       destaggering = FALSE)

# Import binary (netcdf) file and convert coordinates from km to m,
# with shift of 100 m in both directions:
mydata <- importRaster(file = "/path_to_file/filename.nc",
                       k = 1000,
                       dx = 100,
                       dy = 100)

## End(Not run)
```

---

importSurferGrd *Import Grid file*

Description

A function to import data from Surfer text grid file.

Usage

```
importSurferGrd(fname, k = 1000, destaggering = FALSE)
```

Arguments

`fname`	Surfer grd file to be imported
`k`	Factor to apply to x and y coordinates
`destaggering`	Boolean variable to apply or not destaggering.

Details

Surfer grd file is imported and an array of x, y, z columns is returned X and y coordinates can be converted from km to m (default k=1000) and vice versa. Destaggering is applied by default.

Value

A dataset with x, y and z columns is returned.

See Also

`importRaster()`, `importADSOBIN()`

Examples

```
## Not run:
# Import Surfer Grd file and convert coordinates from km to m,
# with destaggering
mydata <- importSurferGrd("/path_to_file/filename.grd", k = 1000)

# Import Surfer Grd file and do not convert coordinates, without
#  destaggering
mydata <- importSurferGrd("path_to_file/filename.grd", k = 1,
                          destaggering = FALSE)

## End(Not run)
```

---

plotAvgRad *Plot hourly average radiation*

Description

Plot a histogram with hourly average of solar radiation, together with hourly maxima for June and December.

Usage

```
plotAvgRad(mydata, date = "date", rad = "radg")
```

Arguments

mydata	A data frame containing fields with solar radiation time series.
date	Name of the column representing date and time.
rad	Name of the column representing radiation.

Value

A ggplot2 plot.

See Also

plotStabilityClass(), plotAvgTemp()

Examples

```
data(stMeteo)
plotAvgRad(stMeteo, date = "date", rad = "radg")
```

---

plotAvgTemp *Plot average temperature*

Description

plotAvgTemp builds a bar plot of time average temperature and two line plots with maximum and minimum temperature.

Usage

```
plotAvgTemp(
  mydata,
  temp = "temp",
  avg.time = "1 month",
  ylabel = "Temperatura [C]",
  title = ""
)
```

Arguments

mydata	A data frame containing fields date and temp
temp	Name of the column representing temperature
avg.time	This defines the time period to average to (see openair::timeAverage). Default is "1 month".
ylabel	The label to be plot along y axis
title	Option plot title

Value

A plot with average, min and max temperature in a given range of time.

Note

plotAvgTemp uses openair::timeAvearge to compute average.

See Also

plotStabilityClass(), plotAvgRad()

Examples

```
# Plot histogram with monthly averages together with maxima and minima
# curves
data(stMeteo)
plotAvgTemp(stMeteo)
plotAvgTemp(stMeteo, temp = "temperature",
            avg.time = "1 month", ylabel = "Temperatura [C]")
```

`plotStabilityClass` *Plot stability class*

Description

Plot histogram of stability class on season or hour base.

Usage

```
plotStabilityClass(mydata, sc = "sc", type = "season")
```

Arguments

`mydata`	A data frame containing `date` and `stability class` fields.
`sc`	The name of the stability class field.
`type`	type determines how the data are split and then plotted. Accepted values are "season" (default) and "hour".

Details

Numerical values of stability classes are mapped as: 1 = A, 2 = B, ..., 6 = F.

Value

A `ggplot2` plot.

See Also

`stabilityClass()`, `plotAvgRad()`, `plotAvgTemp()`

Examples

```
data(stMeteo)

# Season plot of stability class pgt
plotStabilityClass(stMeteo, sc = "pgt", type = "season")

# Hourly plot of stability class pgt
plotStabilityClass(stMeteo, sc = "pgt", type = "hour")
```

removeOutliers *Remove data outliers*

Description

Remove data outliers based on the interquartile range.

Usage

```
removeOutliers(x, k = 1.5)
```

Arguments

x	vector of data.
k	factor to applied to the interquartile range (default = 1.5).

Details

The interquartile range IQR is computed from input dataset as IQR = Q3 - Q1, where Q1 is 25th percentile and Q3 is the 75th percentile. Values larger than Q3 + k * IQR and smaller than Q1 - k * IQR are deemed as outliers and substituted with NA's.

The default value of k is 1.5.

Value

A numeric vector with the same length as input vector.

Examples

```
mydata <- c(-10 * runif(10), runif(10))
removeOutliers(mydata)
```

rollingMax *Compute rolling max*

Description

The rolling maximum value along a series of data is computed.

Usage

```
rollingMax(mydata, length = 24)
```

Arguments

mydata	A vector of data
length	The length of data subset where the maximum values has to be picked. The value must be greater or equal than 3.

Details

It computes the maximum value centred along a subset of data.

Value

A numeric vector of the same length as mydata.

Examples

```
# Compute rolling max along 24 hours on hourly time series
data(airquality)
solar.R.24 <- rollingMax(mydata = airquality$Solar.R, length = 24)
```

---

stabilityClass *Stability class.*

Description

Computes stability class given net radiation, total cloud cover and wind speed.

Usage

```
stabilityClass(rad, tcc, ws, option = "impact")
```

Arguments

rad	The net radiation in W/m^2
tcc	The total cloud cover in a range from 1 to 8
ws	wind speed in m/s
option	This is to determine which specific categories to use to determine the stability class. It can be impact to comply with ARIA Impact(tm), pasquill or custom.

Details

stabilityClass() computes stability class according to IAEA method based on net radiation, total cloud cover tcc and wind speed. Net radiation and wind are used by day; tcc and wind are used by night.

Three different alogorithms are implemented; see source code for details.

Value

`stabilityClass` returns a numeric vector with Pasquill stability classes coded as: A = 1, B = 2, ... , F = 6.

See Also

`plotStabilityClass()`

Examples

```
# Compute stability class with custom algorithm
stMeteo$cst <- stabilityClass(rad = stMeteo$rad,
                              tcc = stMeteo$tcc,
                              ws = stMeteo$ws,
                              option = "custom")
```

---

stMeteo *Meteorological dataset with hourly values*

Description

A dataset containing 8760 hourly values of some meteorological variables corresponding to a full solar year.

Usage

```
stMeteo
```

Format

A data frame with 8760 rows and 7 variables:

date date time in yyyy-mm-hh HH:MM:SS

ws wind speed in m/s

wd wind direction in deg.

temp air temperature in C

radg Global solar radiation in W/m^2

tcc Total cloud cover in integers ranging from 0 to 8

pgt Pasquill-Gifford-Turner stability class

Source

Self derived dataset.

`vectorField` *Vector field plot*

Description

Simple function to plot a **velocities** vector field.

Usage

```
vectorField(data, scale = 1, everyx = 1, everyy = 1, size = 0.25)
```

Arguments

`data`	A dataframe containing data to be plotted in the form of: *(x, y, u, v)*.
`scale`	length factor of vector components
`everyx`	keep one out of every *everyx* values, along *x* direction.
`everyy`	keep one out of every *everyy* values, along *y* direction.
`size`	arrow size.

Details

This function plots a vector field given a data.frame with coordinates (x, y) and corresponding velocity components (u, v). Vectors are coloured by magnitude (speed). The coordinates are assumed to be on a regular rectangular domain in UTM reference system.

This function is heavily inspired by snippets of code in *R Graphics Cookbook* by Winston Chang (https://r-graphics.org/index.html).

Value

A `ggplot2` plot.

Examples

```
## Not run:
metU <- importADSOBIN('/path/to/meteofile',
                      variable = 'U',
                      slice=2,
                      k = 1000,
                      verbose = TRUE)
metU <- as.data.frame(metU)
metU <- metU %>%
        mutate(u = z, z = NULL)

metV <- importADSOBIN('/path/to/meteofile',
                      variable = 'V',
                      slice=2,
                      k = 1000,
```

```
                      verbose = TRUE)
metV <- as.data.frame(metV)
metV <- metV %>%
        mutate(v = z, z = NULL)

met <- merge(metU, metV, by = c("x", "y"))

vectorField(met, everyx = 2, everyy = 2, scalex = 10, scaley = 10) +
    coord_fixed(ratio = 1, xlim = c(0, 1000), ylim = c(0, 1000)) +
    scale_color_viridis_c()


## End(Not run)
```

Index