Package: openairmaps (via r-universe)

November 20, 2026

Type Package

Title Create Maps of Air Pollution Data

Version 0.11.0

Description Combine the air quality data analysis methods of 'openair' with the JavaScript 'Leaflet' (<https://leafletjs.com/>) library. Functionality includes plotting site maps, ``directional analysis'' figures such as polar plots, and air mass trajectories.

License MIT + file LICENSE

URL https://openair-project.github.io/openairmaps/, https://github.com/openair-project/openairmaps

BugReports https://github.com/openair-project/openairmaps/issues

Depends R (>= 4.1.0)

Imports cli, dplyr, ggplot2, leaflet (>= 2.2.0), lifecycle, lubridate, openair (>= 3.2.0), purrr (>= 1.1.0), rlang, sf, stringr, tidyr

Suggests carrier, curl, ggspatial (>= 1.1.11), ggtext, gstat, jsonlite, knitr, mirai, png, prettymapr, rmarkdown, rnaturalearth, rnaturalearthdata, shiny (>= 1.9.0), stars, terra, testthat (>= 3.0.0), withr

Config/Needs/website openair-project/openairpkgdown, rnaturalearth, rnaturalearthdata

Config/roxygen2/version 8.1.1

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

LazyData true

NeedsCompilation no

Author Jack Davison [cre, aut] (ORCID: <https://orcid.org/0000-0003-2653-6615>), David Carslaw [aut] (ORCID: <https://orcid.org/0000-0003-0991-950X>)

Maintainer Jack Davison <jack.davison@wsp.com>

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

Date/Publication 2026-11-20 14:40:07 UTC

RemoteUrl https://github.com/cran/openairmaps

RemoteRef HEAD

RemoteSha 3eaeae5400aab2b4662ff24ac1495fda70bf6fb0

Contents

addPolarMarkers 2
addTrajLevel 6
addTrajPaths 9
annulusMap 11
buildPopup 20
convertPostcode 22
diffMap 23
freqMap 33
krigingMap 41
networkMap 49
percentileMap 52
pointMap 59
polar_data 66
polarMap 67
pollroseMap 78
quickTextHTML 84
traj_data 85
trajLevelMap 86
trajMap 90
windroseMap 94

Index **102**

`addPolarMarkers` *Add polar markers to leaflet map*

Description

This function is similar (but not identical to) the `leaflet::addMarkers()` and `leaflet::addCircleMarkers()` functions in `leaflet`, which allows users to add `openair` directional analysis plots to any leaflet map and have more control over groups and layerIds than in "all-in-one" functions like `polarMap()`.

Usage

```
addPolarMarkers(
  map,
  pollutant,
  fun = openair::polarPlot,
  lng = NULL,
  lat = NULL,
  layerId = NULL,
  group = NULL,
  popup = NULL,
  popupOptions = NULL,
  label = NULL,
  labelOptions = NULL,
  options = leaflet::markerOptions(),
  clusterOptions = NULL,
  clusterId = NULL,
  key.position = "none",
  d.icon = 200,
  d.fig = 3.5,
  alpha = 1,
  progress = TRUE,
  data = leaflet::getMapData(map),
  ...
)

addPolarDiffMarkers(
  map,
  pollutant,
  before = leaflet::getMapData(map),
  after = leaflet::getMapData(map),
  lng = NULL,
  lat = NULL,
  layerId = NULL,
  group = NULL,
  popup = NULL,
  popupOptions = NULL,
  label = NULL,
  labelOptions = NULL,
  options = leaflet::markerOptions(),
  clusterOptions = NULL,
  clusterId = NULL,
  key.position = "none",
  d.icon = 200,
  d.fig = 3.5,
  alpha = 1,
  progress = TRUE,
  ...
)
```

Arguments

map	a map widget object created from leaflet()
pollutant	The name of the pollutant to be plot. Note that, if fun = openair::windRose, you must set pollutant = "ws".
fun	An openair directional analysis plotting function. Supported functions include openair::polarPlot() (the default), openair::polarAnnulus(), openair::polarFreq(), openair::percentileRose(), openair::pollutionRose() and openair::windRose(). For openair::polarDiff(), use addPolarDiffMarkers().
lng	The decimal longitude.
lat	The decimal latitude.
layerId	the layer id
group	the name of the group the newly created layers should belong to (for clearGroup() and addLayersControl() purposes). Human-friendly group names are permitted–they need not be short, identifier-style names. Any number of layers and even different types of layers (e.g., markers and polygons) can share the same group name.
popup	A column of data to be used as a popup.
popupOptions	A Vector of popupOptions() to provide popups
label	A column of data to be used as a label.
labelOptions	A Vector of labelOptions() to provide label options for each label. Default NULL
options	a list of extra options for tile layers, popups, paths (circles, rectangles, polygons, ...), or other map elements
clusterOptions	if not NULL, markers will be clustered using Leaflet.markercluster; you can use markerClusterOptions() to specify marker cluster options
clusterId	the id for the marker cluster layer
key.position	Passed to key.position for the relevant fun.
d.icon	The diameter of the plot on the map in pixels. This will affect the size of the individual polar markers. Alternatively, a vector in the form c(width, height) can be provided if a non-circular marker is desired.
d.fig	The diameter of the plots to be produced using openair in inches. This will affect the resolution of the markers on the map. Alternatively, a vector in the form c(width, height) can be provided if a non-circular marker is desired.
alpha	The desired opacity of the polar markers. Can also be set via options but is provided here for convenience.
progress	Show a progress bar while the markers are created?
data	A data frame. The data frame must contain the data to plot your choice of openair directional analysis plot, which includes wind speed (ws), wind direction (wd), and the column representing the concentration of a pollutant. In addition, data must include a decimal latitude and longitude. By default, it is the data object provided to leaflet::leaflet() initially, but can be overridden.
...	Other arguments for the plotting function (e.g. period for openair::polarAnnulus()).

`before`, `after` A data frame that represents the before/after case. See `openair::polarPlot()` for details of different input requirements. By default, both `before` and `after` are the data object provided to `leaflet::leaflet()` initially, but at least one should be overridden.

Value

A leaflet object.

Functions

- `addPolarMarkers()`: Add any one-table polar marker (e.g., `openair::polarPlot()`)
- `addPolarDiffMarkers()`: Add the two-table `openair::polarDiff()` marker.

See Also

`shiny::runExample(package = "openairmaps")` to see examples of this function used in a `shiny::shinyApp()`

Examples

```
## Not run:
library(leaflet)
library(openair)

# different types of polar plot on one map
leaflet(data = polar_data) |>
  addTiles() |>
  addPolarMarkers(
    "ws",
    fun = openair::windRose,
    annotate = FALSE,
    group = "Wind Rose"
  ) |>
  addPolarMarkers("nox", fun = openair::polarPlot, group = "Polar Plot") |>
  addLayersControl(
    baseGroups = c("Wind Rose", "Polar Plot")
  )

# use of polar diff (NB: both 'before' and 'after' inherit from `leaflet()`,
# so at least one should be overridden - in this case 'after')
leaflet(data = polar_data) |>
  addTiles() |>
  addPolarDiffMarkers("nox",
    after = dplyr::mutate(polar_data, nox = jitter(nox, 5))
  )

## End(Not run)
```

addTrajLevel *Add gridded trajectory statistics to leaflet map*

Description

This function is similar (but not identical to) the `leaflet::addRectangles()` function in `leaflet`, which allows users to add gridded trajectory statistics (calculated using `openair::trajLevel()`) to any leaflet map and have more control over groups and layerIds than in "all-in-one" functions like `trajLevelMap()`.

Usage

```
addTrajLevel(
  map,
  data = leaflet::getMapData(map),
  lng = "lon",
  lat = "lat",
  pollutant = NULL,
  statistic = "frequency",
  percentile = 90,
  lon.inc = 1,
  lat.inc = 1,
  breaks = NULL,
  min.bin = 1,
  .combine = NULL,
  sigma = 1.5,
  smooth = FALSE,
  type = NULL,
  cols = "turbo",
  alpha = 0.5,
  tile.border = NA,
  layerId = NULL,
  group = NULL,
  legend = TRUE,
  legend.position = "topright",
  legend.title = NULL,
  legend.title.autotext = TRUE,
  ...
)
```

Arguments

map	a map widget object created from `leaflet::leaflet()`.
data	Data frame, the result of importing a trajectory file using `openair::importTraj()`. By default, it is the data object provided to `leaflet::leaflet()` initially, but can be overridden.

`lng` The decimal longitude.

`lat` The decimal latitude.

`pollutant` The pollutant of interest. Not required when `statistic = "frequency"`.

`statistic` One of:

- `"frequency"` (the default) shows trajectory frequencies.
- `"hexbin"`, which is similar to `"frequency"` but shows a hexagonal grid of counts.
- `"difference"` - in this case trajectories where the associated concentration is greater than `percentile` are compared with the the full set of trajectories to understand the differences in frequencies of the origin of air masses. The comparison is made by comparing the percentage change in gridded frequencies. For example, such a plot could show that the top 10\ to the east.
- `"pscf"` for a Potential Source Contribution Function map. This statistic method interacts with `percentile`.
- `"cwt"` for concentration weighted trajectories.
- `"sqtba"` to undertake Simplified Quantitative Transport Bias Analysis. This statistic method interacts with `.combine` and `sigma`.

`percentile` The percentile concentration of `pollutant` against which the all trajectories are compared.

`lon.inc, lat.inc`

The longitude and latitude intervals to be used for binning data. If `statistic = "hexbin"`, the minimum value out of of `lon.inc` and `lat.inc` is passed to the `binwidth` argument of `ggplot2::geom_hex()`.

`breaks` `breaks` bins a continuous axis into discrete bins. It can either take a single number (e.g., `breaks = 5`) to split the scale into quantiles, a vector of numbers (e.g., `breaks = c(0, 50, 100, 200, 500)`) to define specific break-points, or a named list. See `breakOpts()` for more details. Note that, in `trajLevel()`, each `statistic` provides a different default break strategy; to override those that bin the limits by default (e.g., `"frequency"`), pass `breaks = NULL` explicitly.

`min.bin` The minimum number of unique points in a grid cell. Counts below `min.bin` are set as missing.

`.combine` When statistic is "SQTBA" it is possible to combine lots of receptor locations to derive a single map. `.combine` identifies the column that differentiates different sites (commonly a column named `"site"`). Note that individual site maps are normalised first by dividing by their mean value.

`sigma` For the SQTBA approach `sigma` determines the amount of back trajectory spread based on the Gaussian plume equation. Values in the literature suggest 5.4 km after one hour. However, testing suggests lower values reveal source regions more effectively while not introducing too much noise.

`smooth` Should the trajectory surface be smoothed? Defaults to `FALSE`. Note that, when `smooth = TRUE`, no popup information will be available.

`type` Optionally, a method to condition the `data`, passed to `openair::cutData()`. Each level of `type` is added to the map as its own `group` (sharing a single colour scale), so can be toggled between using `leaflet::addLayersControl()`. If `type` is provided, `group` is ignored.

cols	The colours used for plotting, passed to openair::openColours(). The default, "turbo", is a rainbow palette with relatively perceptually uniform colours.
alpha	Opacity of the tiles. Must be between 0 and 1.
tile.border	Colour to use for the border of binned tiles. Defaults to NA, which draws no border.
layerId	The base string for the layer id. The actual layer IDs will be in the format "layerId-tilenum". For example, the first tile will be "layerId-1".
group	the name of the group the newly created layers should belong to (for leaflet::clearGroup() and leaflet::addLayersControl() purposes). Human-friendly group names are permitted–they need not be short, identifier-style names. Any number of layers and even different types of layers (e.g. markers and polygons) can share the same group name.
legend	Should a legend be added to the map? Defaults to TRUE. When type is not used, the legend is given the same group as the tiles, so it is shown and hidden alongside them.
legend.position	Where should the legend be placed? One of "topright", "topleft", "bottomleft" or "bottomright". Passed to the position argument of leaflet::addLegend().
legend.title	Title of the legend. By default, when legend.title = NULL, the function will attempt to provide a sensible legend title based on statistic. legend.title allows users to overwrite this - for example, to include units or other contextual information. Users may wish to use HTML tags to format the title.
legend.title.autotext	When legend.title.autotext = TRUE, legend.title will be first run through quickTextHTML().
...	Other arguments to pass to leaflet::addRectangles(). These will override any defaults set by addTrajLevel() (e.g., weight).

Value

A leaflet object.

See Also

openair::trajLevel()

Examples

```
## Not run:
library(leaflet)

leaflet() |>
  addTiles() |>
  addTrajLevel(
    data = traj_data,
    pollutant = "pm2.5",
    statistic = "pscf",
```

```
    min.bin = 10,
    group = "PSCF"
  ) |>
  addLayersControl(overlayGroups = "PSCF")

## End(Not run)
```

---

addTrajPaths *Add trajectory paths to leaflet map*

Description

This function is similar (but not identical to) the `leaflet::addMarkers()` function in `leaflet`, which allows users to add trajectory paths to any leaflet map and have more control over groups and layerIds than in "all-in-one" functions like `trajMap()`.

Usage

```
addTrajPaths(
  map,
  lng = "lon",
  lat = "lat",
  layerId = NULL,
  group = NULL,
  data = leaflet::getMapData(map),
  npoints = 12,
  markerColor = NULL,
  ...
)
```

Arguments

`map`	a map widget object created from `leaflet::leaflet()`.
`lng`	The decimal longitude.
`lat`	The decimal latitude.
`layerId`	The base string for the layer id. The actual layer IDs will be in the format "layerId-linenum" for lines and "layerId_linenum-pointnum" for points. For example, the first point of the first trajectory path will be "layerId-1-1".
`group`	the name of the group the newly created layers should belong to (for `leaflet::clearGroup()` and `leaflet::addLayersControl()` purposes). Human-friendly group names are permitted–they need not be short, identifier-style names. Any number of layers and even different types of layers (e.g. markers and polygons) can share the same group name.
`data`	Data frame, the result of importing a trajectory file using `openair::importTraj()`. By default, it is the data object provided to `leaflet::leaflet()` initially, but can be overridden.

npoints	A dot is placed every npoints along each full trajectory. For hourly back trajectories points are plotted every npoints hours. This helps to understand where the air masses were at particular times and get a feel for the speed of the air (points closer together correspond to slower moving air masses). Defaults to 12.
markerColor	An optional colour (or vector of colours, one per row of data) to use for the point markers, overriding whatever color is passed via .... Unlike color/fillColor, this is permitted to vary per point - useful for colouring the markers along a single trajectory by a continuous variable (e.g. a pollutant) while keeping the trajectory line itself a single colour, as required by leaflet::addPolylines().
...	Other arguments o pass to both leaflet::addCircleMarkers() and leaflet::addPolylines(). If you use the color argument, it is important to ensure the vector you supply is of length one to avoid issues with leaflet::addPolylines() (i.e., use color = ~ pal(nox)[1]), unless you use markerColor to separately control the colour of the point markers. Note that opacity controls the opacity of the lines and fillOpacity the opacity of the markers.

Details

addTrajPaths() can be a powerful way of quickly plotting trajectories on a leaflet map, but users should take some care due to any additional arguments being passed to both leaflet::addCircleMarkers() and leaflet::addPolylines(). In particular, users should be weary of the use of the color argument. Specifically, if color is passed a vector of length greater than one, multiple polylines will be drawn on top of one another. At best this will affect opacity, but at worst this will significantly impact the performance of R and the final leaflet map.

To mitigate this, please ensure that any vector passed to color is of length one. This is simple if you want the whole path to be the same colour, but more difficult if you want to colour by a pollutant, for example. The easiest way to achieve this is to write a for loop or use another iterative approach (e.g. the purrr package) to add one path per arrival date. An example of this is provided in the Examples.

If you only need to vary the colour of the *markers* rather than the lines - for example, colouring the points by a pollutant that changes over time - use markerColor instead, which is not subject to the same length-one restriction as color and can vary across every trajectory in data, not just within one. This often avoids the need for a loop entirely; see the Examples.

Value

A leaflet object.

See Also

shiny::runExample(package = "openairmaps") to see examples of this function used in a shiny::shinyApp()

Examples

```
## Not run:
library(leaflet)

pal <- colorNumeric(palette = "viridis", domain = traj_data$height)
```

```
map <- leaflet() |>
  addTiles()

for (i in seq_along(unique(traj_data$date))) {
  data <- dplyr::filter(traj_data, date == unique(traj_data$date)[i])

  map <- map |>
    addTrajPaths(
      data = data,
      color = pal(data$height)[1],
      markerColor = pal(data$height),
      opacity = 0.5,
      fillOpacity = 0.8
    )
}

map

## End(Not run)
```

---

annulusMap *Polar annulus plots on dynamic and static maps*

Description

The annulusMap() function creates a map using polar annulus plots as markers. Any number of pollutants can be specified using the `pollutant` argument, and multiple layers of markers can be created using `type`. By default, these maps are dynamic and can be panned, zoomed, and otherwise interacted with. Using the `static` argument allows for static images to be produced instead.

Usage

```
annulusMap(
  data,
  pollutant = NULL,
  period = "hour",
  limits = "free",
  breaks = NULL,
  latitude = NULL,
  longitude = NULL,
  crs = 4326,
  type = NULL,
  popup = NULL,
  label = NULL,
  provider = "OpenStreetMap",
  cols = "turbo",
  alpha = 1,
  theme = "default",
```

```
  key.position = "none",
  legend = TRUE,
  legend.position = NULL,
  legend.title = NULL,
  legend.title.autotext = TRUE,
  control.collapsed = FALSE,
  control.position = "topright",
  control.autotext = TRUE,
  d.icon = 200,
  d.fig = 3.5,
  static = FALSE,
  static.nrow = NULL,
  api_key = NULL,
  progress = TRUE,
  ...,
  control = NULL
)
```

Arguments

`data` *Input data table with pollutant, wind, and geo-spatial information.*

required | *scope:* dynamic & static

A data frame. The data frame must contain the data to plot the directional analysis marker, which includes wind speed (`ws`), wind direction (`wd`), and the column representing the concentration of a pollutant. In addition, `data` must include a decimal latitude and longitude (or X/Y coordinate used in conjunction with `crs`).

`pollutant` *Pollutant name(s).*

required | *scope:* dynamic & static

The column name(s) of the pollutant(s) to plot. If multiple pollutants are specified and a non-pairwise statistic is supplied, the `type` argument will no longer be able to be used and:

- *Dynamic*: The pollutants can be toggled between using a "layer control" menu.
- *Static*: The pollutants will each appear in a different panel.

Multiple `pollutants` prohibit the use of the `type` argument for non-pairwise statistics.

`period` *Temporal period for radial axis.*

default: `"hour"` | *scope:* dynamic & static

Options are "hour" (the default, to plot diurnal variations), "season" to plot variation throughout the year, "weekday" to plot day of the week variation and "trend" to plot the trend by wind direction.

`limits` *Specifier for the plot colour scale bounds.*

default: `"free"` | *scope:* dynamic & static

One of:

- `"fixed"` which ensures all of the markers use the same colour scale.

- `"free"` (the default) which allows all of the markers to use different colour scales.
- A numeric vector in the form `c(lower, upper)` used to define the colour scale. For example, `limits = c(0, 100)` would force the plot limits to span 0-100.

`limits` is ignored if `breaks` is also provided.

`breaks` *Specifier for a binned colour scale.*

default: `NULL` | *scope:* dynamic & static

By default, markers use a continuous colour scale (see `limits`). If `breaks` is provided, the colour scale is instead binned into discrete intervals. One of:

- A numeric vector of breakpoints. For example, `breaks = c(0, 50, 100, 1000)` would bin the colour scale into three intervals.
- A single number, which is the number of intervals to bin the colour scale into.
- A list created using `openair::breakOpts()`, which provides finer control over the binning method and allows custom `labels` to be defined.

In all cases, a single set of breakpoints is calculated across *all* of the markers, so that they share the same intervals and a shared legend can be drawn. If any of the markers' data fall outside of the given breakpoints, the lowest and/or highest breakpoints are extended to cover them. If `breaks` is provided, `limits` is ignored.

`latitude, longitude`

The decimal latitude(Y)/longitude(X).

default: `NULL` | *scope:* dynamic & static

Column names representing the decimal latitude and longitude (or other Y/X coordinate if using a different `crs`). If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).

`crs` *The coordinate reference system (CRS).*

default: `4326` | *scope:* dynamic & static

The coordinate reference system (CRS) of the data, passed to `sf::st_crs()`. By default this is EPSG:4326, the CRS associated with the commonly used latitude and longitude coordinates. Different coordinate systems can be specified using `crs` (e.g., `crs = 27700` for the British National Grid). Note that non-lat/lng coordinate systems will be re-projected to EPSG:4326 for plotting on the map.

`type` *A method to condition the* `data` *for separate plotting.*

default: `NULL` | *scope:* dynamic & static

Used for splitting the input data into different groups, passed to the `type` argument of `openair::cutData()`. When `type` is specified:

- *Dynamic*: The different data splits can be toggled between using a "layer control" menu.
- *Static*: The data splits will each appear in a different panel.

`type` cannot be used if multiple `pollutant` columns have been provided.

popup

Content for marker popups on dynamic maps.

default: NULL | *scope:* dynamic

Columns to be used as the HTML content for marker popups on dynamic maps. Popups may be useful to show information about the individual sites (e.g., site names, codes, types, etc.). If a vector of column names are provided they are passed to buildPopup() using its default values.

Numeric columns (e.g., pollutant concentrations) have many values at each marker, so are averaged and labelled as such (e.g., "NO2 (mean)"). Names given to popup (e.g., c("Mean NOx" = "nox")) are used instead of these labels.

label

Content for marker hover-over on dynamic maps.

default: NULL | *scope:* dynamic

Column to be used as the HTML content for hover-over labels. Labels are useful for the same reasons as popups, though are typically shorter. As with popup, numeric columns are averaged at each marker.

provider

The basemap(s) to be used.

default: "OpenStreetMap" | *scope:* dynamic & static

The base map(s) to be used for the map. If not provided, will default to "OpenStreetMap"/"osm" for both dynamic and static maps.

- *Dynamic*: Any number of leaflet::providers. See http://leaflet-extras.github.io/leaflet-providers/preview/ for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery"))
- *Static*: One of the options listed in rosm::osm.types() (for example, "osm", "cartodark", "cartolight", etc.).

There is some overlap in static and dynamic providers. For example, {ggspatial} uses "osm" to specify "OpenStreetMap". When static providers are provided to dynamic maps or vice versa, {openairmaps} will attempt to substitute the correct provider string.

cols

Colours to use for plotting.

default: "turbo" | *scope:* dynamic & static

The colours used for plotting, passed to openair::openColours(). The default, "turbo", is a rainbow palette with relatively perceptually uniform colours.

alpha

Transparency value for polar markers.

default: 1 | *scope:* dynamic & static

A value between 0 (fully transparent) and 1 (fully opaque).

theme

Custom ggplot2 theme for the polar markers.

default: "default" | *scope:* dynamic & static

A custom ggplot2 theme to add to the polar markers. This should ideally be a partial theme (i.e., ggplot2::theme()) over a complete theme (e.g., ggplot2::theme_bw()) as other arguments like key interact with the plot theme *before* custom themes are set, so would be overridden by a complete theme.

`key.position` *Legend position for individual marker legends.*

default: `"none"` | *scope:* dynamic & static

When `key.position` is not `"none"`, a key will be drawn for each individual marker. Potentially useful when `limits = "free"`, but of limited use otherwise.

`legend` *Draw a shared legend?*

default: `TRUE` | *scope:* dynamic & static

When all markers share the same colour scale, should a shared legend be created at the side of the map? This is the case for:

- *Continuous* colour scales, when all markers share the same `limits` (e.g., when `limits != "free"` in `polarMap()`), which are shown as a colour bar. This legend is not drawn if the colour scale has been transformed using `trans` (e.g., `trans = "log10"`), as it would be misleading.
- *Discrete* colour scales, when all markers are binned into the same intervals (e.g., when `breaks` are passed to `polarMap()`, or in `windroseMap()`), which are shown as a set of labelled colours.

`legend.position` *Position of the shared legend.*

default: `NULL` | *scope:* dynamic & static

When `legend = TRUE`, where should the legend be placed?

- *Dynamic*: One of "topright", "topleft", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLegend()`.
- *Static*: One of "top", "right", "bottom" or "left". Passed to the `legend.position` argument of `ggplot2::theme()`.

`legend.title` *Title of the legend.*

default: `NULL` | *scope:* dynamic & static

By default, when `legend.title = NULL`, the function will attempt to provide a sensible legend title. `legend.title` allows users to overwrite this - for example, to include units or other contextual information. For *dynamic* maps, users may wish to use HTML tags to format the title.

`legend.title.autotext` *Automatically format the title of the legend?*

default: `TRUE` | *scope:* dynamic & static

When `legend.title.autotext = TRUE`, `legend.title` will be first run through `quickTextHTML()` (*dynamic*) or `openair::quickText()` (*static*).

`control.collapsed` *Show the layer control as a collapsed?*

default: `FALSE` | *scope:* dynamic

For *dynamic* maps, should the "layer control" interface be collapsed? If `TRUE`, users will have to hover over an icon to view the options.

`control.position` *Position of the layer control menu*

default: `"topright"` | *scope:* dynamic

When `type != NULL`, or multiple pollutants are specified, where should the "layer control" interface be placed? One of "topleft", "topright", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLayersControl()`.

`control.autotext`
: *Automatically format the content of the layer control menu?*

default: `TRUE` | *scope:* dynamic

When `control.autotext = TRUE`, the content of the "layer control" interface will be first run through `quickTextHTML()`.

`d.icon`
: *The diameter of the plot on the map in pixels.*

default: `200` | *scope:* dynamic & static

This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`d.fig`
: *The diameter of the plots to be produced using* `{openair}` *in inches.*

default: `3.5` | *scope:* dynamic & static

This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`static`
: *Produce a static map?*

default: `FALSE`

This controls whether a *dynamic* or *static* map is produced. The former is the default and is broadly more useful, but the latter may be preferable for DOCX or PDF outputs (e.g., academic papers).

`static.nrow`
: *Number of rows in a static map.*

default: `NULL` | *scope:* static

Controls the number of rows of panels on a static map when multiple `pollutants` or `type` are specified; passed to the `nrow` argument of `ggplot2::facet_wrap()`. The default, `NULL`, results in a roughly square grid of panels.

`api_key`
: *API key for certain 'static' base map providers.*

default: `NULL` | *scope:* static

Passed to the `api_key` argument of `ggspatial::annotation_map_tile()`. Needed for certain `providers` (e.g., `"cartolight"`).

`progress`
: *Show a progress bar?*

default: `TRUE` | *scope:* dynamic & static

By default, a progress bar is shown to visualise the function's progress creating individual polar markers. This option allows this to be turned off, if desired.

`...`
: Arguments passed on to `openair::polarAnnulus`

`mydata` A data frame minimally containing `date`, a wind direction and a pollutant.

`wd` The name of the column in `mydata` representing the decimal wind direction, 0 to 360 where 0/360 are North and 180 is South. Defaults to `"wd"`.

`resolution` Two plot resolutions can be set: "normal" and "fine" (the default).

`local.tz` Should the results be calculated in local time that includes a treatment of daylight savings time (DST)? The default is not to consider DST issues, provided the data were imported without a DST offset. Emissions activity tends to occur at local time e.g. rush hour is at 8 am every day. When the clocks go forward in spring, the emissions are effectively released into the atmosphere typically 1 hour earlier during the summertime

i.e. when DST applies. When plotting diurnal profiles, this has the effect of "smearing-out" the concentrations. Sometimes, a useful approach is to express time as local time. This correction tends to produce better-defined diurnal profiles of concentration (or other variables) and allows a better comparison to be made with emissions/activity data. If set to FALSE then GMT is used. Examples of usage include `local.tz = "Europe/London"`, `local.tz = "America/New_York"`. See `cutData` and `import` for more details.

`statistic` The statistic that should be applied to each wind speed/direction bin. Can be "mean" (default), "median", "max" (maximum), "frequency". "stdev" (standard deviation), "weighted.mean" or "cpf" (Conditional Probability Function). Because of the smoothing involved, the colour scale for some of these statistics is only to provide an indication of overall pattern and should not be interpreted in concentration units e.g. for `statistic = "weighted.mean"` where the bin mean is multiplied by the bin frequency and divided by the total frequency. In many cases using `polarFreq` will be better. Setting `statistic = "weighted.mean"` can be useful because it provides an indication of the concentration * frequency of occurrence and will highlight the wind speed/direction conditions that dominate the overall mean.

`percentile` If `statistic = "percentile"` or `statistic = "cpf"` then `percentile` is used, expressed from 0 to 100. Note that the percentile value is calculated in the wind speed, wind direction 'bins'. For this reason it can also be useful to set `min.bin` to ensure there are a sufficient number of points available to estimate a percentile. See `quantile` for more details of how percentiles are calculated.

`trans` Should a transformation be applied to the colour scale? If the distribution of data is skewed, the default scale may be dominated by a few high values, so a log or square-root transform may mean the whole colour scale is better presented on the plot. Can be:

- `FALSE`, which performs no transform.
- `TRUE`, which uses an appropriate transform for the plot type (usually `"log10"`).
- A `scales` 'transform' object (e.g., `scales::transform_log10()`).
- A character string corresponding to a `scales` transform function. Useful options include `"sqrt"`, `"log10"`, `"log2"`, `"log1p"`, `"pseudo_log"` and `"reverse"`.

`col.na` Colour to be used to show missing data.

`offset` `offset` controls the size of the 'hole' in the middle and is expressed on a scale of `0` to `100`, where `0` is no hole and `100` is a hole that takes up the entire plotting area.

`angle.scale` In radial plots (e.g., `polarPlot()`), the radial scale is drawn directly on the plot itself. While suitable defaults have been chosen, sometimes the placement of the scale may interfere with an interesting feature. `angle.scale` can take any value between `0` and `360` to place the scale at a different angle, or `FALSE` to move it to the side of the plots.

`min.bin` The minimum number of points allowed in a wind speed/wind direction bin. The default is 1. A value of two requires at least 2 valid records in each bin an so on; bins with less than 2 valid records are set to NA. Care should be taken when using a value > 1 because of the risk of removing real data points. It is recommended to consider your data with care. Also, the `polarFreq` function can be of use in such circumstances.

`exclude.missing` Setting this option to `TRUE` (the default) removes points from the plot that are too far from the original data. The smoothing routines will produce predictions at points where no data exist i.e. they predict. By removing the points too far from the original data produces a plot where it is clear where the original data lie. If set to `FALSE` missing data will be interpolated.

`date.pad` For `type = "trend"` (default), `date.pad = TRUE` will pad-out missing data to the beginning of the first year and the end of the last year. The purpose is to ensure that the trend plot begins and ends at the beginning or end of year.

`force.positive` The default is `TRUE`. Sometimes if smoothing data with steep gradients it is possible for predicted values to be negative. `force.positive = TRUE` ensures that predictions remain positive. This is useful for several reasons. First, with lots of missing data more interpolation is needed and this can result in artefacts because the predictions are too far from the original data. Second, if it is known beforehand that the data are all positive, then this option carries that assumption through to the prediction. The only likely time where setting `force.positive = FALSE` would be if background concentrations were first subtracted resulting in data that is legitimately negative. For the vast majority of situations it is expected that the user will not need to alter the default option.

`k` The smoothing value supplied to `gam` for the temporal and wind direction components, respectively. In some cases e.g. a trend plot with less than 1-year of data the smoothing with the default values may become too noisy and affected more by outliers. Choosing a lower value of `k` (say 10) may help produce a better plot.

`normalise` If `TRUE` concentrations are normalised by dividing by their mean value. This is done *after* fitting the smooth surface. This option is particularly useful if one is interested in the patterns of concentrations for several pollutants on different scales e.g. NOx and CO. Often useful if more than one `pollutant` is chosen.

`key.title` Used to set the title of the legend. The legend title is passed to `quickText()` if `auto.text = TRUE`.

`key.columns,key.rows` Number of columns or rows to be used in a categorical legend. With many categories a single column or row can make the key fit poorly in the plot. The user can thus choose to use several columns by setting `key.columns` to be less than the number of categories.

`auto.text` Either `TRUE` (default) or `FALSE`. If `TRUE` titles and axis labels will automatically try and format pollutant names and units properly, e.g., by subscripting the "2" in "NO2". Passed to `quickText()`.

`plot` When `openair` plots are created they are automatically printed to the active graphics device. `plot = FALSE` deactivates this behaviour. This may

be useful when the plot *data* is of more interest, or the plot is required to appear later (e.g., later in a Quarto document, or to be saved to a file).

`key` Deprecated; please use `key.position`. If `FALSE`, sets `key.position` to `"none"`.

`control` **Removed.** Please use `type`.

Value

Either:

- *Dynamic:* A leaflet object
- *Static:* A `ggplot2` object using `ggplot2::coord_sf()` coordinates with a `ggspatial` basemap

Parallel processing with mirai

Creating a directional analysis map can take a lot of time; each polar marker needs to be plot individually, and many of these require some expensive computations. `openairmaps` supports parallel processing with `{mirai}` to speed these computations up. Users may create workers by running `mirai::daemons()` in their R session. This requires the `{mirai}` and `{carrier}` packages, which are otherwise not needed; without daemons, markers are plotted one at a time.

```
mirai::daemons(4)
polarMap(polar_data, "no2")
```

Typically, spawning one fewer daemons than your number of available cores is a useful rule of thumb. Parallel processing will be most useful for the most computationally intensive plotting functions - i.e., `polarMap()` and `annulusMap()`.

Customisation of static maps using ggplot2

As all static plots functions are `ggplot2` figures, further customisation is possible using functions such as `ggplot2::theme()`, `ggplot2::guides()` and `ggplot2::labs()`.

Subscripting pollutants (e.g., the "x" in "NOx") is achieved using the ggtext package. Therefore if you choose to override the plot theme, it is recommended to use `[ggplot2::theme()]` and `[ggtext::element_markdown()]` to define the `strip.text` parameter.

Legends can be removed using `ggplot2::theme(legend.position = "none")`, or further customised using `ggplot2::guides()` and either `color = ggplot2::guide_colourbar()` for continuous legends or `color = ggplot2::guide_legend()` for discrete legends.

The extent of a map can be adjusted using the `xlim` and `ylim` arguments of `ggplot2::coord_sf()`.

```
polarMap(polar_data, "no2", static = TRUE) +
    ggplot2::coord_sf(
        xlim = c(-0.3, 0.2),
        ylim = c(51.2, 51.8)
    )
```

See Also

openair::polarAnnulus()

Other directional analysis maps: diffMap(), freqMap(), percentileMap(), polarMap(), pollroseMap(), windroseMap()

Examples

```
## Not run:
annulusMap(polar_data,
  pollutant = "nox",
  period = "hour",
  provider = "Esri.WorldGrayCanvas"
)

## End(Not run)
```

---

buildPopup *Build a Complex Popup for a Leaflet Map*

Description

Group a dataframe together by latitude/longitude columns and create a HTML popup with user-defined columns. By default, the unique values of character columns are collapsed into comma-separated lists, numeric columns are averaged, and date columns are presented as a range. The latitude and longitude columns can also be included, and are shown as-is. Each column's label (but not its value) is formatted using quickTextHTML(). This function returns the input dataframe appended with a "popup" column, which can then be used in the popup argument of a function like polarMap().

Usage

```
buildPopup(
  data,
  columns,
  latitude = NULL,
  longitude = NULL,
  type = NULL,
  fun.character = function(x) {
     paste(unique(x), collapse = ", ")
 },
  fun.numeric = function(x) {
     signif(mean(x, na.rm = TRUE), 3)
 },
  fun.dttm = function(x) {
     paste(lubridate::floor_date(range(x, na.rm = TRUE),
    "day"), collapse = " to ")
 },
```

```
  name = "popup",
  ...
)
```

Arguments

`data`	*Input data table with geo-spatial information.* **required** A data frame containing latitude and longitude information that will go on to be used in a function such as `polarMap()`.
`columns`	*A character vector of column names to include in the popup.* **required** Summaries of the selected columns will appear in the popup. If a named vector is provided, the names of the vector will be used in place of the raw column names. See the Examples for more information.
`latitude, longitude`	*The decimal latitude(Y)/longitude(X).* *default:* `NULL` | *scope:* dynamic & static Column names representing the decimal latitude and longitude (or other Y/X co-ordinate if using a different `crs`). If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).
`type`	*A column to be passed to the* `type` *argument of another function.* *default:* `NULL` Column which will be used for the `type` argument of other mapping functions. This only needs to be used if `type` is going to be used in `polarMap()` or another similar function, and you'd expect different values for the different map layers (for example, if you are calculating a mean pollutant concentration).
`fun.character`	*A function to summarise character, factor and logical columns.* *default:* `function(x) paste(unique(x), collapse = ", ")` The default collapses unique values into a comma-separated list.
`fun.numeric`	*A function to summarise numeric columns.* *default:* `function(x) signif(mean(x, na.rm = TRUE), 3)` The default takes the mean to three significant figures. Other numeric summaries may be of interest, such as the maximum, minimum, standard deviation, and so on.
`fun.dttm`	*A function to summarise date and date-time columns.* *default:* `function(x) paste(lubridate::floor_date(range(x, na.rm = TRUE), "day"), collapse = " to ")` The default presents the date as a range. Other statistics of interest could be the start or end of the dates.
`name`	*The name for the new column.* *default:* `"popup"` A character string for the name for the new column to be appended to `data`.
`...`	**Not currently used.**

Value

a tibble

Examples

```
## Not run:
buildPopup(
  data = polar_data,
  columns = c(
    "Site" = "site",
    "Site Type" = "site_type",
    "Date Range" = "date"
  )
) |>
  polarMap("nox", popup = "popup")

## End(Not run)
```

---

convertPostcode *Convert a UK postcode to a latitude/longitude pair*

Description

This is a much simpler implementation of the tools found in the PostcodesioR R package, intended for use with the networkMap() function.

Usage

```
convertPostcode(postcode)
```

Arguments

postcode	*A valid UK postcode.* **required** A string containing a single valid UK postcode, e.g., "SW1A 1AA".

Value

A list containing the latitude, longitude, and input postcode.

Source

https://postcodes.io/

See Also

The PostcodesioR package at https://github.com/ropensci/PostcodesioR/

Examples

```
rlang::is_installed(c("curl", "jsonlite"))
## Not run:
# convert a UK postcode (needs an internet connection)
convertPostcode("SW1A1AA")

# use with `networkMap()`
palace <- convertPostcode("SW1A1AA")
networkMap("aurn", lat = palace$lat, lng = palace$lng, max_dist = 10)

## End(Not run)
```

---

diffMap *Bivariate polar 'difference' plots on dynamic and static maps*

Description

The `diffMap()` function creates a map using bivariate polar plots as markers. Any number of pollutants can be specified using the `pollutant` argument, and multiple layers of markers can be created using `type`. By default, these maps are dynamic and can be panned, zoomed, and otherwise interacted with. Using the `static` argument allows for static images to be produced instead.

Usage

```
diffMap(
  before,
  after,
  pollutant = NULL,
  x = "ws",
  limits = "free",
  latitude = NULL,
  longitude = NULL,
  crs = 4326,
  type = NULL,
  popup = NULL,
  label = NULL,
  provider = "OpenStreetMap",
  cols = rev(openair::openColours("RdBu", 10)),
  alpha = 1,
  theme = "default",
  key.position = "none",
  legend = TRUE,
  legend.position = NULL,
  legend.title = NULL,
  legend.title.autotext = TRUE,
  control.collapsed = FALSE,
```

```
  control.position = "topright",
  control.autotext = TRUE,
  d.icon = 200,
  d.fig = 3.5,
  static = FALSE,
  static.nrow = NULL,
  api_key = NULL,
  progress = TRUE,
  ...,
  control = NULL
)
```

Arguments

before, after	Data frames representing the "before" and "after" cases. See polarPlot() for details of different input requirements.
pollutant	Mandatory. A pollutant name corresponding to a variable in a data frame should be supplied e.g. pollutant = "nox". There can also be more than one pollutant specified e.g. pollutant = c("nox", "no2"). The main use of using two or more pollutants is for model evaluation where two species would be expected to have similar concentrations. This saves the user stacking the data and it is possible to work with columns of data directly. A typical use would be pollutant = c("obs", "mod") to compare two columns "obs" (the observations) and "mod" (modelled values). When pair-wise statistics such as Pearson correlation and regression techniques are to be plotted, pollutant takes two elements too. For example, pollutant = c("bc", "pm25") where "bc" is a function of "pm25".
x	Name of variable to plot against wind direction in polar coordinates, the default is wind speed, "ws".
limits	*Limits for the plot colour scale.* *default:* "free" | *scope:* dynamic & static One of: • "free" (the default) which allows all of the markers to use different colour scales. • A numeric vector in the form c(lower, upper) used to define the colour scale. For example, limits = c(-10, 10) would force the plot limits to span -10 to 10. It is recommended to use a symmetrical limit scale (along with a "diverging" colour palette) for effective visualisation. Note that the "fixed" option is not supported in diffMap().
latitude, longitude	*The decimal latitude(Y)/longitude(X).* *default:* NULL | *scope:* dynamic & static Column names representing the decimal latitude and longitude (or other Y/X coordinate if using a different crs). If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).

crs

The coordinate reference system (CRS).

default: 4326 | *scope:* dynamic & static

The coordinate reference system (CRS) of the data, passed to sf::st_crs(). By default this is EPSG:4326, the CRS associated with the commonly used latitude and longitude coordinates. Different coordinate systems can be specified using crs (e.g., crs = 27700 for the British National Grid). Note that non-lat/lng coordinate systems will be re-projected to EPSG:4326 for plotting on the map.

type

Character string(s) defining how data should be split/conditioned before plotting. "default" produces a single panel using the entire dataset. Any other options will split the plot into different panels - a roughly square grid of panels if one type is given, or a 2D matrix of panels if two types are given. type is always passed to cutData(), and can therefore be any of:

- A built-in type defined in cutData() (e.g., "season", "year", "weekday", etc.). For example, type = "season" will split the plot into four panels, one for each season.
- The name of a numeric column in mydata, which will be split into n.levels quantiles (defaulting to 4).
- The name of a character or factor column in mydata, which will be used as-is. Commonly this could be a variable like "site" to ensure data from different monitoring sites are handled and presented separately. It could equally be any arbitrary column created by the user (e.g., whether a nearby possible pollutant source is active or not).

Most openair plotting functions can take two type arguments. If two are given, the first is used for the columns and the second for the rows.

popup

Content for marker popups on dynamic maps.

default: NULL | *scope:* dynamic

Columns to be used as the HTML content for marker popups on dynamic maps. Popups may be useful to show information about the individual sites (e.g., site names, codes, types, etc.). If a vector of column names are provided they are passed to buildPopup() using its default values.

Numeric columns (e.g., pollutant concentrations) have many values at each marker, so are averaged and labelled as such (e.g., "NO2 (mean)"). Names given to popup (e.g., c("Mean NOx" = "nox")) are used instead of these labels.

label

Content for marker hover-over on dynamic maps.

default: NULL | *scope:* dynamic

Column to be used as the HTML content for hover-over labels. Labels are useful for the same reasons as popups, though are typically shorter. As with popup, numeric columns are averaged at each marker.

provider

The basemap(s) to be used.

default: "OpenStreetMap" | *scope:* dynamic & static

The base map(s) to be used for the map. If not provided, will default to "OpenStreetMap"/"osm" for both dynamic and static maps.

- *Dynamic*: Any number of leaflet::providers. See http://leaflet-extras.github.io/leaflet-providers/preview/ for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled

between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., `c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery")`)

- *Static*: One of the options listed in `rosm::osm.types()` (for example, `"osm"`, `"cartodark"`, `"cartolight"`, etc.).

There is some overlap in static and dynamic providers. For example, `{ggspatial}` uses "osm" to specify "OpenStreetMap". When static providers are provided to dynamic maps or vice versa, `{openairmaps}` will attempt to substitute the correct provider string.

`cols` *Colours to use for plotting.*

default: `rev(openair::openColours("RdBu", 10))` | *scope:* dynamic & static

The colours used for plotting, passed to `openair::openColours()`. It is recommended to use a "diverging" colour palette (along with a symmetrical `limit` scale) for effective visualisation.

`alpha` *Transparency value for polar markers.*

default: `1` | *scope:* dynamic & static

A value between 0 (fully transparent) and 1 (fully opaque).

`theme` *Custom ggplot2 theme for the polar markers.*

default: `"default"` | *scope:* dynamic & static

A custom `ggplot2` theme to add to the polar markers. This should ideally be a partial theme (i.e., `ggplot2::theme()`) over a complete theme (e.g., `ggplot2::theme_bw()`) as other arguments like `key` interact with the plot theme *before* custom themes are set, so would be overridden by a complete theme.

`key.position` *Legend position for individual marker legends.*

default: `"none"` | *scope:* dynamic & static

When `key.position` is not `"none"`, a key will be drawn for each individual marker. Potentially useful when `limits = "free"`, but of limited use otherwise.

`legend` *Draw a shared legend?*

default: `TRUE` | *scope:* dynamic & static

When all markers share the same colour scale, should a shared legend be created at the side of the map? This is the case for:

- *Continuous* colour scales, when all markers share the same `limits` (e.g., when `limits != "free"` in `polarMap()`), which are shown as a colour bar. This legend is not drawn if the colour scale has been transformed using `trans` (e.g., `trans = "log10"`), as it would be misleading.
- *Discrete* colour scales, when all markers are binned into the same intervals (e.g., when `breaks` are passed to `polarMap()`, or in `windroseMap()`), which are shown as a set of labelled colours.

`legend.position` *Position of the shared legend.*

default: `NULL` | *scope:* dynamic & static

When `legend = TRUE`, where should the legend be placed?

- *Dynamic*: One of "topright", "topleft", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLegend()`.

- *Static*: One of "top", "right", "bottom" or "left". Passed to the `legend.position` argument of `ggplot2::theme()`.

`legend.title` *Title of the legend.*

default: `NULL` | *scope:* dynamic & static

By default, when `legend.title = NULL`, the function will attempt to provide a sensible legend title. `legend.title` allows users to overwrite this - for example, to include units or other contextual information. For *dynamic* maps, users may wish to use HTML tags to format the title.

`legend.title.autotext`

Automatically format the title of the legend?

default: `TRUE` | *scope:* dynamic & static

When `legend.title.autotext = TRUE`, `legend.title` will be first run through `quickTextHTML()` (*dynamic*) or `openair::quickText()` (*static*).

`control.collapsed`

Show the layer control as a collapsed?

default: `FALSE` | *scope:* dynamic

For *dynamic* maps, should the "layer control" interface be collapsed? If `TRUE`, users will have to hover over an icon to view the options.

`control.position`

Position of the layer control menu

default: `"topright"` | *scope:* dynamic

When `type != NULL`, or multiple pollutants are specified, where should the "layer control" interface be placed? One of "topleft", "topright", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLayersControl()`.

`control.autotext`

Automatically format the content of the layer control menu?

default: `TRUE` | *scope:* dynamic

When `control.autotext = TRUE`, the content of the "layer control" interface will be first run through `quickTextHTML()`.

`d.icon` *The diameter of the plot on the map in pixels.*

default: `200` | *scope:* dynamic & static

This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`d.fig` *The diameter of the plots to be produced using* `{openair}` *in inches.*

default: `3.5` | *scope:* dynamic & static

This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`static` *Produce a static map?*

default: `FALSE`

This controls whether a *dynamic* or *static* map is produced. The former is the default and is broadly more useful, but the latter may be preferable for DOCX or PDF outputs (e.g., academic papers).

`static.nrow` *Number of rows in a static map.*

default: `NULL` | *scope:* static

Controls the number of rows of panels on a static map when multiple `pollutants` or `type` are specified; passed to the `nrow` argument of `ggplot2::facet_wrap()`. The default, `NULL`, results in a roughly square grid of panels.

`api_key` *API key for certain 'static' base map providers.*

default: `NULL` | *scope:* static

Passed to the `api_key` argument of `ggspatial::annotation_map_tile()`. Needed for certain `providers` (e.g., `"cartolight"`).

`progress` *Show a progress bar?*

default: `TRUE` | *scope:* dynamic & static

By default, a progress bar is shown to visualise the function's progress creating individual polar markers. This option allows this to be turned off, if desired.

`...` Arguments passed on to `openair::polarPlot`

`mydata` A data frame minimally containing a decimal wind direction, another variable to plot in polar coordinates (the default is a column `"ws"` — wind speed) and a pollutant. Should also contain `date` if plots by time period are required.

`wd` The name of the column in `mydata` representing the decimal wind direction, 0 to 360 where 0/360 are North and 180 is South. Defaults to `"wd"`.

`statistic` The statistic that should be applied to each wind speed/direction bin. Because of the smoothing involved, the colour scale for some of these statistics is only to provide an indication of overall pattern and should not be interpreted in concentration units e.g. for `statistic = "weighted.mean"` where the bin mean is multiplied by the bin frequency and divided by the total frequency. In many cases using `polarFreq` will be better. Setting `statistic = "weighted.mean"` can be useful because it provides an indication of the concentration * frequency of occurrence and will highlight the wind speed/direction conditions that dominate the overall mean.Can be:

- "mean" (default), "median", "max" (maximum), "frequency". "stdev" (standard deviation), "weighted.mean".
- `statistic = "nwr"` Implements the Non-parametric Wind Regression approach of Henry et al. (2009) that uses kernel smoothers. The `openair` implementation is not identical because Gaussian kernels are used for both wind direction and speed. The smoothing is controlled by `ws_spread` and `wd_spread`.
- `statistic = "cpf"` the conditional probability function (CPF) is plotted and a single (usually high) percentile level is supplied. The CPF is defined as CPF = my/ny, where my is the number of samples in the y bin (by default a wind direction, wind speed interval) with mixing ratios greater than the *overall* percentile concentration, and ny is the total number of samples in the same wind sector (see Ashbaugh et al., 1985). Note that percentile intervals can also be considered; see `percentile` for details.
- When `statistic = "r"` or `statistic = "Pearson"`, the Pearson correlation coefficient is calculated for *two* pollutants. The calculation

involves a weighted Pearson correlation coefficient, which is weighted by Gaussian kernels for wind direction an the radial variable (by default wind speed). More weight is assigned to values close to a wind speed-direction interval. Kernel weighting is used to ensure that all data are used rather than relying on the potentially small number of values in a wind speed-direction interval.

- When `statistic = "Spearman"`, the Spearman correlation coefficient is calculated for *two* pollutants. The calculation involves a weighted Spearman correlation coefficient, which is weighted by Gaussian kernels for wind direction an the radial variable (by default wind speed). More weight is assigned to values close to a wind speed-direction interval. Kernel weighting is used to ensure that all data are used rather than relying on the potentially small number of values in a wind speed-direction interval.
- `"robust_slope"` is another option for pair-wise statistics and `"quantile.slope"`, which uses quantile regression to estimate the slope for a particular quantile level (see also `tau` for setting the quantile level).
- `"york_slope"` is another option for pair-wise statistics which uses the *York regression method* to estimate the slope. In this method the uncertainties in `x` and `y` are used in the determination of the slope. The uncertainties are provided by `x_error` and `y_error` — see below.

`breaks` `breaks` bins a continuous axis into discrete bins. It can either take a single number (e.g., `breaks = 5`) to split the scale into quantiles, a vector of numbers (e.g., `breaks = c(0, 50, 100, 200, 500)` to define specific break-points, or a named list. See `breakOpts()` for more details.

`trans` Should a transformation be applied to the colour scale? If the distribution of data is skewed, the default scale may be dominated by a few high values, so a log or square-root transform may mean the whole colour scale is better presented on the plot. Can be:

- `FALSE`, which performs no transform.
- `TRUE`, which uses an appropriate transform for the plot type (usually `"log10"`).
- A `scales` 'transform' object (e.g., `scales::transform_log10()`).
- A character string corresponding to a `scales` transform function. Useful options include `"sqrt"`, `"log10"`, `"log2"`, `"log1p"`, `"pseudo_log"` and `"reverse"`.

`exclude.missing` Setting this option to `TRUE` (the default) removes points from the plot that are too far from the original data. The smoothing routines will produce predictions at points where no data exist i.e. they predict. By removing the points too far from the original data produces a plot where it is clear where the original data lie. If set to `FALSE` missing data will be interpolated.

`uncertainty` Should the uncertainty in the calculated surface be shown? If `TRUE` three plots are produced on the same scale showing the predicted surface together with the estimated lower and upper uncertainties at the 95% confidence interval. Calculating the uncertainties is useful to understand whether features are real or not. For example, at high wind speeds where

there are few data there is greater uncertainty over the predicted values. The uncertainties are calculated using the GAM and weighting is done by the frequency of measurements in each wind speed-direction bin. Note that if uncertainties are calculated then the type is set to "default".

`percentile` If `statistic = "percentile"` then `percentile` is used, expressed from 0 to 100. Note that the percentile value is calculated in the wind speed, wind direction 'bins'. For this reason it can also be useful to set `min.bin` to ensure there are a sufficient number of points available to estimate a percentile. See `quantile` for more details of how percentiles are calculated. `percentile` is also used for the Conditional Probability Function (CPF) plots. `percentile` can be of length two, in which case the percentile *interval* is considered for use with CPF. For example, `percentile = c(90, 100)` will plot the CPF for concentrations between the 90 and 100th percentiles. Percentile intervals can be useful for identifying specific sources. In addition, `percentile` can also be of length 3. The third value is the 'trim' value to be applied. When calculating percentile intervals many can cover very low values where there is no useful information. The trim value ensures that values greater than or equal to the trim * mean value are considered *before* the percentile intervals are calculated. The effect is to extract more detail from many source signatures. See the manual for examples. Finally, if the trim value is less than zero the percentile range is interpreted as absolute concentration values and subsetting is carried out directly.

`weights` At the edges of the plot there may only be a few data points in each wind speed-direction interval, which could in some situations distort the plot if the concentrations are high. `weights` applies a weighting to reduce their influence. For example and by default if only a single data point exists then the weighting factor is 0.25 and for two points 0.5. To not apply any weighting and use the data as is, use `weights = c(1, 1, 1)`.
An alternative to down-weighting these points they can be removed altogether using `min.bin`.

`min.bin` The minimum number of points allowed in a wind speed/wind direction bin. The default is 1. A value of two requires at least 2 valid records in each bin an so on; bins with less than 2 valid records are set to NA. Care should be taken when using a value > 1 because of the risk of removing real data points. It is recommended to consider your data with care. Also, the `polarFreq` function can be of use in such circumstances.

`col.na` When `min.bin` is > 1 it can be useful to show where data are removed on the plots. This is done by shading the missing data in `col.na`. To not highlight missing data when `min.bin` > 1 choose `col.na = "transparent"`.

`upper` This sets the upper limit wind speed to be used. Often there are only a relatively few data points at very high wind speeds and plotting all of them can reduce the useful information in the plot.

`angle.scale` In radial plots (e.g., `polarPlot()`), the radial scale is drawn directly on the plot itself. While suitable defaults have been chosen, sometimes the placement of the scale may interfere with an interesting feature. `angle.scale` can take any value between `0` and `360` to place the scale at a different angle, or `FALSE` to move it to the side of the plots.

`units` The units shown on the polar axis scale.

force.positive The default is TRUE. Sometimes if smoothing data with steep gradients it is possible for predicted values to be negative. force.positive = TRUE ensures that predictions remain positive. This is useful for several reasons. First, with lots of missing data more interpolation is needed and this can result in artefacts because the predictions are too far from the original data. Second, if it is known beforehand that the data are all positive, then this option carries that assumption through to the prediction. The only likely time where setting force.positive = FALSE would be if background concentrations were first subtracted resulting in data that is legitimately negative. For the vast majority of situations it is expected that the user will not need to alter the default option.

k This is the smoothing parameter used by the gam function in package mgcv. Typically, value of around 100 (the default) seems to be suitable and will resolve important features in the plot. The most appropriate choice of k is problem-dependent; but extensive testing of polar plots for many different problems suggests a value of k of about 100 is suitable. Setting k to higher values will not tend to affect the surface predictions by much but will add to the computation time. Lower values of k will increase smoothing. Sometimes with few data to plot polarPlot will fail. Under these circumstances it can be worth lowering the value of k.

normalise If TRUE concentrations are normalised by dividing by their mean value. This is done *after* fitting the smooth surface. This option is particularly useful if one is interested in the patterns of concentrations for several pollutants on different scales e.g. NOx and CO. Often useful if more than one pollutant is chosen.

key.title Used to set the title of the legend. The legend title is passed to quickText() if auto.text = TRUE.

key.columns,key.rows Number of columns or rows to be used in a categorical legend. With many categories a single column or row can make the key fit poorly in the plot. The user can thus choose to use several columns by setting key.columns to be less than the number of categories.

auto.text Either TRUE (default) or FALSE. If TRUE titles and axis labels will automatically try and format pollutant names and units properly, e.g., by subscripting the "2" in "NO2". Passed to quickText().

ws_spread The value of sigma used for Gaussian kernel weighting of wind speed when statistic = "nwr" or when correlation and regression statistics are used such as *r*. Default is 0.5.

wd_spread The value of sigma used for Gaussian kernel weighting of wind direction when statistic = "nwr" or when correlation and regression statistics are used such as *r*. Default is 4.

x_error The x error / uncertainty used when statistic = "york_slope".

y_error The y error / uncertainty used when statistic = "york_slope".

kernel Type of kernel used for the weighting procedure for when correlation or regression techniques are used. Only "gaussian" is supported but this may be enhanced in the future.

formula.label When pair-wise statistics such as regression slopes are calculated and plotted, should a formula label be displayed? formula.label

will also determine whether concentration information is printed when `statistic = "cpf"`.

`tau` The quantile to be estimated when `statistic` is set to `"quantile.slope"`. Default is `0.5` which is equal to the median and will be ignored if `"quantile.slope"` is not used.

`plot` When `openair` plots are created they are automatically printed to the active graphics device. `plot = FALSE` deactivates this behaviour. This may be useful when the plot *data* is of more interest, or the plot is required to appear later (e.g., later in a Quarto document, or to be saved to a file).

`key` Deprecated; please use `key.position`. If `FALSE`, sets `key.position` to `"none"`.

`control` **Removed.** Please use `type`.

Value

Either:

- *Dynamic:* A leaflet object
- *Static:* A `ggplot2` object using `ggplot2::coord_sf()` coordinates with a `ggspatial` basemap

Parallel processing with mirai

Creating a directional analysis map can take a lot of time; each polar marker needs to be plot individually, and many of these require some expensive computations. `openairmaps` supports parallel processing with `{mirai}` to speed these computations up. Users may create workers by running `mirai::daemons()` in their R session. This requires the `{mirai}` and `{carrier}` packages, which are otherwise not needed; without daemons, markers are plotted one at a time.

```
mirai::daemons(4)
polarMap(polar_data, "no2")
```

Typically, spawning one fewer daemons than your number of available cores is a useful rule of thumb. Parallel processing will be most useful for the most computationally intensive plotting functions - i.e., `polarMap()` and `annulusMap()`.

Customisation of static maps using ggplot2

As all static plots functions are `ggplot2` figures, further customisation is possible using functions such as `ggplot2::theme()`, `ggplot2::guides()` and `ggplot2::labs()`.

Subscripting pollutants (e.g., the "x" in "NOx") is achieved using the ggtext package. Therefore if you choose to override the plot theme, it is recommended to use `[ggplot2::theme()]` and `[ggtext::element_markdown()]` to define the `strip.text` parameter.

Legends can be removed using `ggplot2::theme(legend.position = "none")`, or further customised using `ggplot2::guides()` and either `color = ggplot2::guide_colourbar()` for continuous legends or `color = ggplot2::guide_legend()` for discrete legends.

The extent of a map can be adjusted using the `xlim` and `ylim` arguments of `ggplot2::coord_sf()`.

```
polarMap(polar_data, "no2", static = TRUE) +
    ggplot2::coord_sf(
        xlim = c(-0.3, 0.2),
        ylim = c(51.2, 51.8)
    )
```

See Also

openair::polarDiff()

Other directional analysis maps: annulusMap(), freqMap(), percentileMap(), polarMap(), pollroseMap(), windroseMap()

Examples

```
## Not run:
# NB: "after" is some dummy data to demonstrate functionality
diffMap(
  before = polar_data,
  after = dplyr::mutate(polar_data, nox = jitter(nox, factor = 5)),
  pollutant = "nox"
)

## End(Not run)
```

---

freqMap *Polar frequency plots on dynamic and static maps*

Description

The freqMap() function creates a map using polar frequency plots as markers. Any number of pollutants can be specified using the pollutant argument, and multiple layers of markers can be created using type. By default, these maps are dynamic and can be panned, zoomed, and otherwise interacted with. Using the static argument allows for static images to be produced instead.

Usage

```
freqMap(
  data,
  pollutant = NULL,
  statistic = "mean",
  breaks = "free",
  latitude = NULL,
  longitude = NULL,
  crs = 4326,
  type = NULL,
  popup = NULL,
  label = NULL,
  provider = "OpenStreetMap",
```

```
  cols = "turbo",
  alpha = 1,
  theme = "default",
  key.position = "none",
  legend = TRUE,
  legend.position = NULL,
  legend.title = NULL,
  legend.title.autotext = TRUE,
  control.collapsed = FALSE,
  control.position = "topright",
  control.autotext = TRUE,
  d.icon = 200,
  d.fig = 3.5,
  static = FALSE,
  static.nrow = NULL,
  api_key = NULL,
  progress = TRUE,
  ...,
  control = NULL
)
```

Arguments

`data`	*Input data table with pollutant, wind, and geo-spatial information.* **required** | *scope:* dynamic & static A data frame. The data frame must contain the data to plot the directional analysis marker, which includes wind speed (`ws`), wind direction (`wd`), and the column representing the concentration of a pollutant. In addition, `data` must include a decimal latitude and longitude (or X/Y coordinate used in conjunction with `crs`).
`pollutant`	*Pollutant name(s).* **required** | *scope:* dynamic & static The column name(s) of the pollutant(s) to plot. If multiple pollutants are specified and a non-pairwise statistic is supplied, the `type` argument will no longer be able to be used and: • *Dynamic*: The pollutants can be toggled between using a "layer control" menu. • *Static*: The pollutants will each appear in a different panel. Multiple `pollutants` prohibit the use of the `type` argument for non-pairwise statistics.
`statistic`	*The statistic that should be applied to each wind speed/direction bin.* *default:* `"mean"` | *scope:* dynamic & static Can be "frequency", "mean", "median", "max" (maximum), "stdev" (standard deviation) or "weighted.mean". The option "frequency" is the simplest and plots the frequency of wind speed/direction in different bins. The scale therefore shows the counts in each bin. The option "mean" (the default) will plot the mean concentration of a pollutant (see next point) in wind speed/direction

bins, and so on. Finally, "weighted.mean" will plot the concentration of a pollutant weighted by wind speed/direction. Each segment therefore provides the percentage overall contribution to the total concentration. Note that for options other than "frequency", it is necessary to also provide the name of a pollutant. See function `openair::cutData()` for further details.

`breaks` *Specifier for the breaks of the plot colour scale.*

default: `"free"` | *scope:* dynamic & static

One of:

- `"fixed"` which ensures all of the markers use the same colour scale.
- `"free"` (the default) which allows all of the markers to use different colour scales.
- A numeric vector defining a sequence of numbers to use as the breaks. The sequence could represent one with equal spacing, e.g., `breaks = seq(0, 100, 10)` - a scale from 0-10 in intervals of 10, or a more flexible sequence, e.g., `breaks = c(0, 1, 5, 7, 10)`, which may be useful for some situations.

`latitude, longitude`

The decimal latitude(Y)/longitude(X).

default: `NULL` | *scope:* dynamic & static

Column names representing the decimal latitude and longitude (or other Y/X coordinate if using a different `crs`). If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).

`crs` *The coordinate reference system (CRS).*

default: `4326` | *scope:* dynamic & static

The coordinate reference system (CRS) of the data, passed to `sf::st_crs()`. By default this is EPSG:4326, the CRS associated with the commonly used latitude and longitude coordinates. Different coordinate systems can be specified using `crs` (e.g., `crs = 27700` for the British National Grid). Note that non-lat/lng coordinate systems will be re-projected to EPSG:4326 for plotting on the map.

`type` *A method to condition the* `data` *for separate plotting.*

default: `NULL` | *scope:* dynamic & static

Used for splitting the input data into different groups, passed to the `type` argument of `openair::cutData()`. When `type` is specified:

- *Dynamic*: The different data splits can be toggled between using a "layer control" menu.
- *Static*: The data splits will each appear in a different panel.

`type` cannot be used if multiple `pollutant` columns have been provided.

`popup` *Content for marker popups on dynamic maps.*

default: `NULL` | *scope:* dynamic

Columns to be used as the HTML content for marker popups on dynamic maps. Popups may be useful to show information about the individual sites (e.g., site names, codes, types, etc.). If a vector of column names are provided they are passed to `buildPopup()` using its default values.

Numeric columns (e.g., pollutant concentrations) have many values at each marker, so are averaged and labelled as such (e.g., "NO2 (mean)"). Names given to `popup` (e.g., `c("Mean NOx" = "nox")`) are used instead of these labels.

`label` *Content for marker hover-over on dynamic maps.*

default: `NULL` | *scope:* dynamic

Column to be used as the HTML content for hover-over labels. Labels are useful for the same reasons as popups, though are typically shorter. As with `popup`, numeric columns are averaged at each marker.

`provider` *The basemap(s) to be used.*

default: `"OpenStreetMap"` | *scope:* dynamic & static

The base map(s) to be used for the map. If not provided, will default to `"OpenStreetMap"`/`"osm"` for both dynamic and static maps.

- *Dynamic*: Any number of leaflet::providers. See `http://leaflet-extras.github.io/leaflet-providers/preview/` for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., `c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery")`)
- *Static*: One of the options listed in `rosm::osm.types()` (for example, `"osm"`, `"cartodark"`, `"cartolight"`, etc.).

There is some overlap in static and dynamic providers. For example, `{ggspatial}` uses "osm" to specify "OpenStreetMap". When static providers are provided to dynamic maps or vice versa, `{openairmaps}` will attempt to substitute the correct provider string.

`cols` *Colours to use for plotting.*

default: `"turbo"` | *scope:* dynamic & static

The colours used for plotting, passed to `openair::openColours()`. The default, `"turbo"`, is a rainbow palette with relatively perceptually uniform colours.

`alpha` *Transparency value for polar markers.*

default: `1` | *scope:* dynamic & static

A value between 0 (fully transparent) and 1 (fully opaque).

`theme` *Custom ggplot2 theme for the polar markers.*

default: `"default"` | *scope:* dynamic & static

A custom `ggplot2` theme to add to the polar markers. This should ideally be a partial theme (i.e., `ggplot2::theme()`) over a complete theme (e.g., `ggplot2::theme_bw()`) as other arguments like `key` interact with the plot theme *before* custom themes are set, so would be overridden by a complete theme.

`key.position` *Legend position for individual marker legends.*

default: `"none"` | *scope:* dynamic & static

When `key.position` is not `"none"`, a key will be drawn for each individual marker. Potentially useful when `limits = "free"`, but of limited use otherwise.

`legend` *Draw a shared legend?*

default: `TRUE` | *scope:* dynamic & static

When all markers share the same colour scale, should a shared legend be created at the side of the map? This is the case for:

- *Continuous* colour scales, when all markers share the same `limits` (e.g., when `limits != "free"` in `polarMap()`), which are shown as a colour bar. This legend is not drawn if the colour scale has been transformed using `trans` (e.g., `trans = "log10"`), as it would be misleading.
- *Discrete* colour scales, when all markers are binned into the same intervals (e.g., when `breaks` are passed to `polarMap()`, or in `windroseMap()`), which are shown as a set of labelled colours.

`legend.position`

Position of the shared legend.

default: `NULL` | *scope:* dynamic & static

When `legend = TRUE`, where should the legend be placed?

- *Dynamic*: One of "topright", "topleft", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLegend()`.
- *Static*: One of "top", "right", "bottom" or "left". Passed to the `legend.position` argument of `ggplot2::theme()`.

`legend.title` *Title of the legend.*

default: `NULL` | *scope:* dynamic & static

By default, when `legend.title = NULL`, the function will attempt to provide a sensible legend title. `legend.title` allows users to overwrite this - for example, to include units or other contextual information. For *dynamic* maps, users may wish to use HTML tags to format the title.

`legend.title.autotext`

Automatically format the title of the legend?

default: `TRUE` | *scope:* dynamic & static

When `legend.title.autotext = TRUE`, `legend.title` will be first run through `quickTextHTML()` (*dynamic*) or `openair::quickText()` (*static*).

`control.collapsed`

Show the layer control as a collapsed?

default: `FALSE` | *scope:* dynamic

For *dynamic* maps, should the "layer control" interface be collapsed? If `TRUE`, users will have to hover over an icon to view the options.

`control.position`

Position of the layer control menu

default: `"topright"` | *scope:* dynamic

When `type != NULL`, or multiple pollutants are specified, where should the "layer control" interface be placed? One of "topleft", "topright", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLayersControl()`.

`control.autotext`

Automatically format the content of the layer control menu?

default: `TRUE` | *scope:* dynamic

When `control.autotext = TRUE`, the content of the "layer control" interface will be first run through `quickTextHTML()`.

`d.icon` *The diameter of the plot on the map in pixels.*

default: `200` | *scope:* dynamic & static

This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`d.fig` *The diameter of the plots to be produced using* `{openair}` *in inches.*

default: `3.5` | *scope:* dynamic & static

This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`static` *Produce a static map?*

default: `FALSE`

This controls whether a *dynamic* or *static* map is produced. The former is the default and is broadly more useful, but the latter may be preferable for DOCX or PDF outputs (e.g., academic papers).

`static.nrow` *Number of rows in a static map.*

default: `NULL` | *scope:* static

Controls the number of rows of panels on a static map when multiple `pollutants` or `type` are specified; passed to the `nrow` argument of `ggplot2::facet_wrap()`. The default, `NULL`, results in a roughly square grid of panels.

`api_key` *API key for certain 'static' base map providers.*

default: `NULL` | *scope:* static

Passed to the `api_key` argument of `ggspatial::annotation_map_tile()`. Needed for certain `providers` (e.g., `"cartolight"`).

`progress` *Show a progress bar?*

default: `TRUE` | *scope:* dynamic & static

By default, a progress bar is shown to visualise the function's progress creating individual polar markers. This option allows this to be turned off, if desired.

`...` Arguments passed on to `openair::polarFreq`

`mydata` A data frame minimally containing a wind speed, a decimal wind direction, and `date`.

`ws` The name of the column in `mydata` representing the wind speed. Defaults to `"ws"`.

`wd` The name of the column in `mydata` representing the decimal wind direction, 0 to 360 where 0/360 are North and 180 is South. Defaults to `"wd"`.

`ws.int` Wind speed interval assumed. In some cases e.g. a low met mast, an interval of 0.5 may be more appropriate.

`wd.nint` Number of intervals of wind direction.

`grid.line` Radial spacing of grid lines.

`limits` The limits of the colour scale, in the form `c(lower, upper)`. For example, `limits = c(0, 100)` will set the colour scale to be between `0` and `100`. Values greater than `100` will be coloured as if they were `100`, and those lower than `0` will be coloured as if they were `0`. `limits` can be wider than the range of the data, which can be useful for ensuring multiple plots share the same colour scale.

`trans` Should a transformation be applied to the colour scale? If the distribution of data is skewed, the default scale may be dominated by a few high values, so a log or square-root transform may mean the whole colour scale is better presented on the plot. Can be:

- FALSE, which performs no transform.
- TRUE, which uses an appropriate transform for the plot type (usually "log10").
- A scales 'transform' object (e.g., scales::transform_log10()).
- A character string corresponding to a scales transform function. Useful options include "sqrt", "log10", "log2", "log1p", "pseudo_log" and "reverse".

min.bin The minimum number of points allowed in a wind speed/wind direction bin. The default is 1. A value of two requires at least 2 valid records in each bin an so on; bins with less than 2 valid records are set to NA. Care should be taken when using a value > 1 because of the risk of removing real data points. It is recommended to consider your data with care. Also, the polarFreq function can be of use in such circumstances.

ws.upper A user-defined upper wind speed to use. This is useful for ensuring a consistent scale between different plots. For example, to always ensure that wind speeds are displayed between 1-10, set ws.int = 10.

angle.scale In radial plots (e.g., polarPlot()), the radial scale is drawn directly on the plot itself. While suitable defaults have been chosen, sometimes the placement of the scale may interfere with an interesting feature. angle.scale can take any value between 0 and 360 to place the scale at a different angle, or FALSE to move it to the side of the plots.

offset offset controls the size of the 'hole' in the middle and is expressed on a scale of 0 to 100, where 0 is no hole and 100 is a hole that takes up the entire plotting area.

border.col The colour of the boundary of each wind speed/direction bin. The default is transparent. Another useful choice sometimes is "white".

key.title Used to set the title of the legend. The legend title is passed to quickText() if auto.text = TRUE.

key.columns,key.rows Number of columns or rows to be used in a categorical legend. With many categories a single column or row can make the key fit poorly in the plot. The user can thus choose to use several columns by setting key.columns to be less than the number of categories.

auto.text Either TRUE (default) or FALSE. If TRUE titles and axis labels will automatically try and format pollutant names and units properly, e.g., by subscripting the "2" in "NO2". Passed to quickText().

plot When openair plots are created they are automatically printed to the active graphics device. plot = FALSE deactivates this behaviour. This may be useful when the plot *data* is of more interest, or the plot is required to appear later (e.g., later in a Quarto document, or to be saved to a file).

key Deprecated; please use key.position. If FALSE, sets key.position to "none".

control **Removed.** Please use type.

Value

Either:

- *Dynamic:* A leaflet object
- *Static:* A ggplot2 object using ggplot2::coord_sf() coordinates with a ggspatial basemap

Parallel processing with mirai

Creating a directional analysis map can take a lot of time; each polar marker needs to be plot individually, and many of these require some expensive computations. openairmaps supports parallel processing with {mirai} to speed these computations up. Users may create workers by running mirai::daemons() in their R session. This requires the {mirai} and {carrier} packages, which are otherwise not needed; without daemons, markers are plotted one at a time.

```
mirai::daemons(4)
polarMap(polar_data, "no2")
```

Typically, spawning one fewer daemons than your number of available cores is a useful rule of thumb. Parallel processing will be most useful for the most computationally intensive plotting functions - i.e., polarMap() and annulusMap().

Customisation of static maps using ggplot2

As all static plots functions are ggplot2 figures, further customisation is possible using functions such as ggplot2::theme(), ggplot2::guides() and ggplot2::labs().

Subscripting pollutants (e.g., the "x" in "NOx") is achieved using the ggtext package. Therefore if you choose to override the plot theme, it is recommended to use [ggplot2::theme()] and [ggtext::element_markdown()] to define the strip.text parameter.

Legends can be removed using ggplot2::theme(legend.position = "none"), or further customised using ggplot2::guides() and either color = ggplot2::guide_colourbar() for continuous legends or color = ggplot2::guide_legend() for discrete legends.

The extent of a map can be adjusted using the xlim and ylim arguments of ggplot2::coord_sf().

```
polarMap(polar_data, "no2", static = TRUE) +
    ggplot2::coord_sf(
        xlim = c(-0.3, 0.2),
        ylim = c(51.2, 51.8)
    )
```

See Also

openair::polarFreq()

Other directional analysis maps: annulusMap(), diffMap(), percentileMap(), polarMap(), pollroseMap(), windroseMap()

Examples

```
## Not run:
freqMap(polar_data,
  pollutant = "nox",
  statistic = "mean",
```

```
  provider = "Esri.WorldGrayCanvas"
)

## End(Not run)
```

---

krigingMap *Spatially interpolated dynamic and static maps*

Description

[Experimental]

These functions create interpolated surfaces out of data at individual monitoring sites. This can be useful to 'fill in the gaps' to estimate pollution concentrations where no monitoring is occurring, or better identify geographical patterns in pollution data. `krigingMap()` creates a smooth spatially interpolated surface using either inverse distance weighting or point kriging. `voronoiMap()` creates a surface of 'closest observation' polygons. The kriging formula is currently always `pollutant ~ 1`; `krigingMap()` does not currently support more complex models.

When `type` is used, each `type` level is summarised and interpolated separately, but all levels share the same colour scale so that they can be compared.

Usage

```
krigingMap(
  data,
  pollutant = NULL,
  statistic = "mean",
  percentile = 95,
  newdata = NULL,
  method = c("idw", "kriging"),
  breaks = NULL,
  labels = NULL,
  limits = NULL,
  latitude = NULL,
  longitude = NULL,
  crs = 4326,
  type = NULL,
  popup = NULL,
  label = NULL,
  provider = "OpenStreetMap",
  cols = "turbo",
  alpha = 0.8,
  show.markers = TRUE,
  marker.border = "white",
  legend = TRUE,
  legend.position = NULL,
  legend.title = NULL,
  legend.title.autotext = TRUE,
```

```
  control.collapsed = FALSE,
  control.position = "topright",
  control.autotext = TRUE,
  static = FALSE,
  static.nrow = NULL,
  api_key = NULL,
  vgm = gstat::vgm(psill = 1, model = "Exp", range = 50, nugget = 1),
  args.idw = list(debug.level = 0),
  args.variogram = list(),
  args.fit.variogram = list(),
  args.krige = list(debug.level = 0),
  ...
)

voronoiMap(
  data,
  pollutant = NULL,
  statistic = "mean",
  percentile = 95,
  newdata = NULL,
  breaks = NULL,
  labels = NULL,
  limits = NULL,
  latitude = NULL,
  longitude = NULL,
  crs = 4326,
  type = NULL,
  popup = NULL,
  label = NULL,
  provider = "OpenStreetMap",
  cols = "turbo",
  alpha = 0.8,
  show.markers = TRUE,
  marker.border = "white",
  voronoi.border = "white",
  legend = TRUE,
  legend.position = NULL,
  legend.title = NULL,
  legend.title.autotext = TRUE,
  control.collapsed = FALSE,
  control.position = "topright",
  control.autotext = TRUE,
  static = FALSE,
  static.nrow = NULL,
  api_key = NULL,
  args.voronoi = list(),
  ...
)
```

Arguments

`data` *Input data table with pollutant and geo-spatial information.*

required | *scope:* dynamic & static

A data frame. The data frame must contain at least one numeric column to interpolate, plus a decimal latitude and longitude (or X/Y coordinate used in conjunction with `crs`).

`pollutant` *Pollutant name.*

required | *scope:* dynamic & static

The column name of the pollutant to plot. Multiple `pollutants` are prohibited by this function.

`statistic` *Statistic for aggregating pollutant data.*

default: `"mean"` | *scope:* dynamic & static

Pollutant data will be aggregated by latitude & longitude; `statistic` controls how this is achieved. Possible statistics include:

- `"mean"`: the arithmetic mean (using `base::mean()`)
- `"median"`: the median (middle) value (using `stats::median()`)
- `"max"`: the maximum value (using `max()`)
- `"min"`: the minimum value (using `min()`)
- `"sd"`: the standard deviation (using `stats::sd()`)
- `"percentile"`: a percentile value, defined using the `percentile` argument (using `stats::quantile()`)

To show this value in marker popups or labels, include `pollutant` in `popup` or `label` (e.g., `popup = c("site", "nox")`).

`percentile` *The percentile when 'statistic = "percentile"*

default: `95` | *scope:* dynamic & static

The percentile level used when `statistic = "percentile"`. The default is `95`, representing 95%. Should be between `0` and `100`.

`newdata` *A spatial dataset of prediction locations.*

default: `NULL` | *scope:* dynamic & static

By default, a bounding box of all latitudes and longitudes are used for prediction, but this is often not useful or aesthetically pleasing. `newdata` should be a spatial data frame (constructed with `sf::st_as_sf()`). This may be a country or authority boundary relevant to the `data` input.

`method` *Spatial interpolation method.*

default: `"idw"` | *scope:* dynamic & static

The spatial interpolation method to use for `krigingMap()`. `"idw"` uses inverse distance weighting (IDW) which is simpler and faster. `"kriging"` uses full point kriging which is typically more accurate, but is also more complex and computationally intensive.

`labels, breaks` *Discretise the map color scale.*

default: `NULL` | *scope:* dynamic & static

By default, a continuous colour scale is used. If `breaks` are provided, the colour scale will be discretised using `cut()`. `labels` can also be provided to customise how each factor level is labelled.

If the data extend beyond `breaks`, the outermost breaks are expanded to cover them so that no sites are left uncoloured. For example, `breaks = c(50, 100, 150)` would become `c(25, 100, 200)` for data ranging from 25 to 200.

`limits` *Specifier for the plot colour scale bounds.*

default: `NULL` | *scope:* dynamic & static

A numeric vector in the form `c(lower, upper)` used to define the colour scale. For example, `limits = c(0, 100)` would force the plot limits to span 0-100. If `NULL`, appropriate limits will be selected based on the range in `data[[pollutant]]`. Values outside of `limits` are given the colour of the nearest limit.

`latitude, longitude`

The decimal latitude(Y)/longitude(X).

default: `NULL` | *scope:* dynamic & static

Column names representing the decimal latitude and longitude (or other Y/X co-ordinate if using a different `crs`). If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).

`crs` *The coordinate reference system (CRS).*

default: `4326` | *scope:* dynamic & static

The coordinate reference system (CRS) of the data, passed to `sf::st_crs()`. By default this is EPSG:4326, the CRS associated with the commonly used latitude and longitude coordinates. Different coordinate systems can be specified using `crs` (e.g., `crs = 27700` for the British National Grid). Note that non-lat/lng coordinate systems will be re-projected to EPSG:4326 for plotting on the map.

`type` *A method to condition the* `data` *for separate plotting.*

default: `NULL` | *scope:* dynamic & static

Used for splitting the input data into different groups, passed to the `type` argument of `openair::cutData()`. When `type` is specified:

- *Dynamic*: The different data splits can be toggled between using a "layer control" menu.
- *Static*: The data splits will each appear in a different panel.

`type` cannot be used if multiple `pollutant` columns have been provided.

`popup` *Content for marker popups on dynamic maps.*

default: `NULL` | *scope:* dynamic

Columns to be used as the HTML content for site marker popups on dynamic maps. Popups may be useful to show information about the individual sites (e.g., site names, codes, types, etc.). If a vector of column names are provided they are passed to `buildPopup()` using its default values.

The `pollutant` column shows the site `statistic` (i.e., what the markers are coloured by), labelled as such (e.g., "NO2 (max)"). Any other numeric columns have many values at each site, so are averaged and labelled "(mean)". Names given to `popup` (e.g., `c("Site NOx" = "nox")`) are used instead of these labels.

`label` *Content for marker hover-over on dynamic maps.*

default: `NULL` | *scope:* dynamic

Column to be used as the HTML content for hover-over labels. Labels are useful for the same reasons as popups, though are typically shorter. The `pollutant`

column shows the site `statistic`, and other numeric columns are averaged at each site.

`provider` *The basemap(s) to be used.*

default: `"OpenStreetMap"` | *scope:* dynamic & static

The base map(s) to be used for the map. If not provided, will default to `"OpenStreetMap"`/`"osm"` for both dynamic and static maps.

- *Dynamic*: Any number of leaflet::providers. See `http://leaflet-extras.github.io/leaflet-providers/preview/` for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., `c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery")`)
- *Static*: One of the options listed in `rosm::osm.types()` (for example, `"osm"`, `"cartodark"`, `"cartolight"`, etc.).

There is some overlap in static and dynamic providers. For example, `{ggspatial}` uses "osm" to specify "OpenStreetMap". When static providers are provided to dynamic maps or vice versa, `{openairmaps}` will attempt to substitute the correct provider string.

`cols` *Colours to use for plotting.*

default: `"turbo"` | *scope:* dynamic & static

The colours used for plotting, passed to `openair::openColours()`. The default, `"turbo"`, is a rainbow palette with relatively perceptually uniform colours.

`alpha` *Transparency value for interpolated surface.*

default: `0.8` | *scope:* dynamic & static

A value between 0 (fully transparent) and 1 (fully opaque).

`show.markers` *Show original monitoring site markers?*

default: `TRUE` | *scope:* dynamic & static

When `TRUE`, the coordinates in the input `data` will be shown as coloured markers. Any popup or `label` is attached to these markers, so neither is shown when `show.markers = FALSE`.

`marker.border`, `voronoi.border`

Border colour to use for markers and voronoi tiles.

default: `"white"` | *scope:* dynamic & static

Any valid HTML colour (e.g., a hex code). Use `NA` for no border.

`legend` *Draw a legend?*

default: `TRUE` | *scope:* dynamic & static

When `TRUE`, a legend will appear on the map identifying the colour scale.

`legend.position`

Position of the shared legend.

default: `NULL` | *scope:* dynamic & static

When `legend = TRUE`, where should the legend be placed?

- *Dynamic*: One of "topright", "topleft", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLegend()`.

- *Static*: One of "top", "right", "bottom" or "left". Passed to the `legend.position` argument of `ggplot2::theme()`.

`legend.title` *Title of the legend.*

default: `NULL` | *scope:* dynamic & static

By default, when `legend.title = NULL`, the function will attempt to provide a sensible legend title. `legend.title` allows users to overwrite this - for example, to include units or other contextual information. For *dynamic* maps, users may wish to use HTML tags to format the title.

`legend.title.autotext`

Automatically format the title of the legend?

default: `TRUE` | *scope:* dynamic & static

When `legend.title.autotext = TRUE`, `legend.title` will be first run through `quickTextHTML()` (*dynamic*) or `openair::quickText()` (*static*).

`control.collapsed`

Show the layer control as a collapsed?

default: `FALSE` | *scope:* dynamic

For *dynamic* maps, should the "layer control" interface be collapsed? If `TRUE`, users will have to hover over an icon to view the options.

`control.position`

Position of the layer control menu

default: `"topright"` | *scope:* dynamic

When `type != NULL`, or multiple pollutants are specified, where should the "layer control" interface be placed? One of "topleft", "topright", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLayersControl()`.

`control.autotext`

Automatically format the content of the layer control menu?

default: `TRUE` | *scope:* dynamic

When `control.autotext = TRUE`, the content of the "layer control" interface will be first run through `quickTextHTML()`.

`static` *Produce a static map?*

default: `FALSE`

This controls whether a *dynamic* or *static* map is produced. The former is the default and is broadly more useful, but the latter may be preferable for DOCX or PDF outputs (e.g., academic papers).

`static.nrow` *Number of rows in a static map.*

default: `NULL` | *scope:* static

Controls the number of rows of panels on a static map when multiple `pollutants` or `type` are specified; passed to the `nrow` argument of `ggplot2::facet_wrap()`. The default, `NULL`, results in a roughly square grid of panels.

`api_key` *API key for certain 'static' base map providers.*

default: `NULL` | *scope:* static

Passed to the `api_key` argument of `ggspatial::annotation_map_tile()`. Needed for certain `providers` (e.g., `"cartolight"`).

vgm *A variogram model*

default: gstat::vgm(psill = 1, model = "Exp", range = 50, nugget = 1) | *scope:* dynamic & static

The variogram model to use when method = "kriging". Must be the output of gstat::vgm(). As the data are in latitude and longitude, {gstat} measures distances (e.g., the range) in kilometres.

args.idw, args.variogram, args.fit.variogram, args.krige

Extra arguments to pass to spatial interpolation functions for krigingMap().

scope: dynamic & static

Extra arguments passed to gstat::idw(), gstat::variogram(), gstat::fit.variogram(), and gstat::krige(). gstat::idw() and gstat::krige() are silenced by default (debug.level = 0), but this can be overridden (e.g., args.idw = list(debug.level = 1)).

... Arguments passed on to openair::cutData

names By default, the columns created by cutData() are named after their type option. Specifying names defines other names for the columns, which map onto the type options in the same order they are given. The length of names should therefore be equal to the length of type.

suffix If names is not specified, suffix will be appended to any added columns that would otherwise overwrite existing columns. For example, cutData(mydata, "nox", suffix = "_cuts") would append a nox_cuts column rather than overwriting nox.

hemisphere Can be "northern" or "southern", used to split data into seasons.

n.levels Number of quantiles to split numeric data into.

wd.res The number of wind direction bins; one of 4 (N/E/S/W), 8 (N/NE/E/SE/etc., the default), 16 (N/NNE/NE/ENE/etc.) or 32 (N/NbE/NNE/NEbN/etc.).

start.day What day of the week should the type = "weekday" start on? The user can change the start day by supplying an integer between 0 and 6. Sunday = 0, Monday = 1, ... For example to start the weekday plots on a Saturday, choose start.day = 6.

start.season What order should the season be. By default, the order is spring, summer, autumn, winter. start.season = "winter" would plot winter first.

is.axis A logical (TRUE/FALSE), used to request shortened cut labels for axes.

local.tz Used for identifying whether a date has daylight savings time (DST) applied or not. Examples include local.tz = "Europe/London", local.tz = "America/New_York", i.e., time zones that assume DST. https://en.wikipedia.org/wiki/List_of_zoneinfo_time_zones shows time zones that should be valid for most systems. It is important that the original data are in GMT (UTC) or a fixed offset from GMT.

latitude,longitude The decimal latitude and longitudes used when type = "daylight". Note that locations west of Greenwich have negative longitudes.

drop How to handle empty factor levels. One of:

- `"default"`: Sensible defaults selected on a case-by-case basis for different `type` options.
- `"empty"`: Drop all empty factor levels.
- `"none"`: Retain all empty factor levels, where possible. For example, for `type = "hour"`, all factor levels from `0` and `23` will be represented.
- `"outside"`: Retain empty factor levels within the range of the data. For example, for `type = "hour"` when the data only contains data for 1 AM and 5 AM, the factor levels, `1`, `2`, `3`, `4` and `5` will be retained.

Some of these options only apply to certain `type` options. For example, for `type = "year"`, `"outside"` is equivalent to `"none"` as there is no fixed range of years to use in the `"none"` case.

`args.voronoi` *Extra arguments to pass to spatial interpolation functions for* `voronoiMap()`.

scope: dynamic & static

Extra arguments passed to `terra::voronoi()`, with the exception of `x` which is dealt with by `voronoiMap()`.

Value

Either:

- *Dynamic:* A leaflet object
- *Static:* A `ggplot2` object using `ggplot2::coord_sf()` coordinates with a `ggspatial` basemap

Customisation of static maps using ggplot2

As all static plots functions are `ggplot2` figures, further customisation is possible using functions such as `ggplot2::theme()`, `ggplot2::guides()` and `ggplot2::labs()`.

Subscripting pollutants (e.g., the "x" in "NOx") is achieved using the ggtext package. Therefore if you choose to override the plot theme, it is recommended to use `[ggplot2::theme()]` and `[ggtext::element_markdown()]` to define the `strip.text` parameter.

Legends can be removed using `ggplot2::theme(legend.position = "none")`, or further customised using `ggplot2::guides()` and either `color = ggplot2::guide_colourbar()` for continuous legends or `color = ggplot2::guide_legend()` for discrete legends.

The extent of a map can be adjusted using the `xlim` and `ylim` arguments of `ggplot2::coord_sf()`.

```
polarMap(polar_data, "no2", static = TRUE) +
    ggplot2::coord_sf(
        xlim = c(-0.3, 0.2),
        ylim = c(51.2, 51.8)
    )
```

Examples

```
## Not run:
# import ozone DAQI
daqi <-
  openair::importUKAQ(
```

```
    pollutant = "o3",
    year = 2020,
    source = "aurn",
    data_type = "daqi",
    meta = TRUE
  )

# get a UK shapefile
uk <- rnaturalearth::ne_countries(scale = 10, country = "united kingdom")

# create spatially interpolated map
voronoiMap(
  daqi,
  pollutant = "poll_index",
  newdata = uk,
  statistic = "max",
  breaks = seq(0.5, 10.5, 1),
  labels = as.character(1:10),
  legend.title = "Max O3 DAQI",
  cols = "daqi"
)

krigingMap(
  daqi,
  pollutant = "poll_index",
  newdata = uk,
  statistic = "max",
  legend.title = "Max O3 DAQI",
  cols = openair::openColours("daqi", n = 10),
  limits = c(1, 10)
)

## End(Not run)
```

`networkMap` *Create a leaflet map of air quality measurement network sites*

Description

This function uses `openair::importMeta()` to obtain metadata for measurement sites and uses it to create an interactive `leaflet` map. By default a map will be created in which readers may toggle between a vector base map and a satellite/aerial image, although users can further customise the control menu using the `provider` and `control` parameters. Any argument which can be passed to `openair::importMeta()` can be passed via `...`, with the exception of `all` and `duplicate`.

Usage

```
networkMap(
  source = "aurn",
```

```
  ...,
  control = NULL,
  cluster = NULL,
  provider = c(Default = "OpenStreetMap", Satellite = "Esri.WorldImagery"),
  legend = TRUE,
  legend.position = "topright",
  control.collapsed = FALSE,
  control.position = "topright"
)
```

Arguments

`source` *One or more UK or European monitoring networks.*

default: `"aurn"`

One or more air quality networks for which data is available through openair. Available networks include:

- `"aurn"`, The UK Automatic Urban and Rural Network.
- `"aqe"`, The Air Quality England Network.
- `"saqn"`, The Scottish Air Quality Network.
- `"waqn"`, The Welsh Air Quality Network.
- `"ni"`, The Northern Ireland Air Quality Network.
- `"local"`, Locally managed air quality networks in England.
- `"imperial"`, Imperial College London (formerly King's College London) networks.
- `"europe"`, European AirBase/e-reporting data.

There are two additional options provided for convenience:

- `"ukaq"` will return metadata for all networks for which data is imported by importUKAQ() (i.e., AURN, AQE, SAQN, WAQN, NI, and the local networks).
- `"all"` will import all available metadata (i.e., "ukaq" plus "imperial" and "europe").

`...` Arguments passed on to `openair::importMeta`

`year` If a single year is selected, only sites that were open at some point in that year are returned. If `all = TRUE` only sites that measured a particular pollutant in that year are returned. Year can also be a sequence e.g. `year = 2010:2020` or of length 2 e.g. `year = c(2010, 2020)`, which will return only sites that were open over the duration.

`pollutant` Character vectors used to search the metadata for the specified `sources`. `pollutant` is case-insensitive. For example, `pollutant = c("nox", "o3")` will return sites which measure *either* NOx *or* O3. Can also take the shorthand `"hc"`, will returns all hydrocarbons. Similar to `code`, values are matched exactly (e.g., `pollutant = "no"` will only return NO and not NO2 or NOx). Note that `pollutant` only applies to networks available through `importUKAQ()`.

`site,code,site_type` Character vectors used to search the metadata for the specified `sources`. All of `code`, `site` and `site_type` are case-insensitive. `code` and `site_type` are matched exactly, but `site` is 'pattern matched' - e.g., `site = "Sunderland"` and `source = "aurn"` will return data for `"Sunderland"`, `"Sunderland Silksworth"` and `"Sunderland Wessington Way"` (plus any future sites with the string `"Sunderland"` in their name).

`lat,lng` Decimal latitude (`lat`) and longitude (`lng`) (or other Y/X coordinate if using a different `crs`). Both must be single numbers. If provided, the data will be returned with a `distance_km` column displaying the distance of each station from the target coordinate. The data will also be automatically sorted by this column.

`crs` The coordinate reference system (CRS) of the data, passed to `sf::st_crs()`. By default this is EPSG:4326, the CRS associated with the commonly used latitude and longitude coordinates. Different coordinate systems can be specified using `crs` (e.g., `crs = 27700` for the British National Grid). Note that non-lat/lng coordinate systems will be re-projected to EPSG:4326 for comparison with the site metadata.

`max_dist,max_n` If `lat` and `lng` are provided, `max_dist` and `max_n` further filter the metadata. `max_dist` defines a maximum distance from the target coordinate in kilometers, and `max_n` a maximum number of sites to be returned. `max_n` is applied after `max_dist`. Both are ignored, with a warning, if `lat` and `lng` are not provided.

`control` *Option to create a 'layer control' menu.*

default: `NULL`

A string to specify categories in a "layer control" menu, to allow readers to select between different site categories. Choices include:

- `"source"` to toggle between different networks
- `"variable"` to toggle between different pollutants
- `"site_type"` for different site classifications
- `"agglomeration"`, `"zone"` or `"local_authority"` for different regions of the UK

`cluster` *Cluster markers together when zoomed out?*

default: `NULL`

When `cluster = TRUE`, markers are clustered together. This may be useful for sources like `"imperial"` where there are many markers very close together. Defaults to `NULL`, which is `TRUE` if there are more than 25 sites mapped and `FALSE` if there are fewer.

`provider` *The basemap(s) to be used.*

default: `c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery")`

Any number of leaflet::providers. See `http://leaflet-extras.github.io/leaflet-providers/preview/` for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., `c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery")`)

`legend` *Draw a shared legend?*

default: `TRUE`

When multiple `sources` are defined, should a shared legend be created at the side of the map?

`legend.position`

Position of the legend

default: `"topright"`

Where should the shared legend be placed? One of "topleft", "topright", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLegend()`.

`control.collapsed`

Show the layer control as a collapsed?

default: `FALSE`

Should the "layer control" interface be collapsed? If `TRUE`, users will have to hover over an icon to view the options.

`control.position`

Position of the layer control menu

default: `"topright"`

Where should the "layer control" interface be placed? One of "topleft", "topright", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLayersControl()`.

Value

A leaflet object.

Examples

```
## Not run:
# view one network, grouped by site type
networkMap(source = "aurn", control = "site_type")

# view multiple networks, grouped by network
networkMap(source = c("aurn", "waqn", "saqn"), control = "source")

## End(Not run)
```

`percentileMap` *Percentile roses on dynamic and static maps*

Description

The `percentileMap()` function creates a map using polar percentile roses as markers. Any number of pollutants can be specified using the `pollutant` argument, and multiple layers of markers can be created using `type`. By default, these maps are dynamic and can be panned, zoomed, and otherwise interacted with. Using the `static` argument allows for static images to be produced instead.

Usage

```
percentileMap(
  data,
  pollutant = NULL,
  percentile = c(25, 50, 75, 90, 95),
  intervals = "fixed",
  latitude = NULL,
  longitude = NULL,
  crs = 4326,
  type = NULL,
  popup = NULL,
  label = NULL,
  provider = "OpenStreetMap",
  cols = "turbo",
  alpha = 1,
  theme = "default",
  key.position = "none",
  legend = TRUE,
  legend.position = NULL,
  legend.title = NULL,
  legend.title.autotext = TRUE,
  control.collapsed = FALSE,
  control.position = "topright",
  control.autotext = TRUE,
  d.icon = 200,
  d.fig = 3.5,
  static = FALSE,
  static.nrow = NULL,
  api_key = NULL,
  progress = TRUE,
  ...,
  control = NULL
)
```

Arguments

`data` *Input data table with pollutant, wind, and geo-spatial information.*

required | *scope:* dynamic & static

A data frame. The data frame must contain the data to plot the directional analysis marker, which includes wind speed (`ws`), wind direction (`wd`), and the column representing the concentration of a pollutant. In addition, `data` must include a decimal latitude and longitude (or X/Y coordinate used in conjunction with `crs`).

`pollutant` *Pollutant name(s).*

required | *scope:* dynamic & static

The column name(s) of the pollutant(s) to plot. If multiple pollutants are specified and a non-pairwise statistic is supplied, the `type` argument will no longer be able to be used and:

- *Dynamic*: The pollutants can be toggled between using a "layer control" menu.
- *Static*: The pollutants will each appear in a different panel.

Multiple `pollutants` prohibit the use of the `type` argument for non-pairwise statistics.

`percentile` *The percentile values for the colour scale bin.*

default: `c(25, 50, 75, 90, 95)` | *scope:* dynamic & static

The percentile value(s) to plot using `openair::percentileRose()`. Must be a vector of values between `0` and `100`. If `percentile = NA` then only a mean line will be shown.

`intervals` *Specifier for the percentile rose radial axis intervals.*

default: `"fixed"` | *scope:* dynamic & static

One of:

- `"fixed"` (the default) which ensures all of the markers use the same radial axis scale.
- `"free"` which allows all of the markers to use different radial axis scales.
- A numeric vector defining a sequence of numbers to use as the intervals, e.g., `intervals = c(0, 10, 30, 50)`.

`latitude, longitude`

The decimal latitude(Y)/longitude(X).

default: `NULL` | *scope:* dynamic & static

Column names representing the decimal latitude and longitude (or other Y/X co-ordinate if using a different `crs`). If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).

`crs` *The coordinate reference system (CRS).*

default: `4326` | *scope:* dynamic & static

The coordinate reference system (CRS) of the data, passed to `sf::st_crs()`. By default this is EPSG:4326, the CRS associated with the commonly used latitude and longitude coordinates. Different coordinate systems can be specified using `crs` (e.g., `crs = 27700` for the British National Grid). Note that non-lat/lng coordinate systems will be re-projected to EPSG:4326 for plotting on the map.

`type` *A method to condition the* `data` *for separate plotting.*

default: `NULL` | *scope:* dynamic & static

Used for splitting the input data into different groups, passed to the `type` argument of `openair::cutData()`. When `type` is specified:

- *Dynamic*: The different data splits can be toggled between using a "layer control" menu.
- *Static*: The data splits will each appear in a different panel.

`type` cannot be used if multiple `pollutant` columns have been provided.

`popup` *Content for marker popups on dynamic maps.*

default: `NULL` | *scope:* dynamic

Columns to be used as the HTML content for marker popups on dynamic maps. Popups may be useful to show information about the individual sites (e.g., site

names, codes, types, etc.). If a vector of column names are provided they are passed to `buildPopup()` using its default values.

Numeric columns (e.g., pollutant concentrations) have many values at each marker, so are averaged and labelled as such (e.g., "NO2 (mean)"). Names given to popup (e.g., `c("Mean NOx" = "nox")`) are used instead of these labels.

`label` *Content for marker hover-over on dynamic maps.*

default: `NULL` | *scope:* dynamic

Column to be used as the HTML content for hover-over labels. Labels are useful for the same reasons as popups, though are typically shorter. As with popup, numeric columns are averaged at each marker.

`provider` *The basemap(s) to be used.*

default: `"OpenStreetMap"` | *scope:* dynamic & static

The base map(s) to be used for the map. If not provided, will default to `"OpenStreetMap"`/`"osm"` for both dynamic and static maps.

- *Dynamic*: Any number of leaflet::providers. See `http://leaflet-extras.github.io/leaflet-providers/preview/` for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., `c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery")`)
- *Static*: One of the options listed in `rosm::osm.types()` (for example, `"osm"`, `"cartodark"`, `"cartolight"`, etc.).

There is some overlap in static and dynamic providers. For example, `{ggspatial}` uses "osm" to specify "OpenStreetMap". When static providers are provided to dynamic maps or vice versa, `{openairmaps}` will attempt to substitute the correct provider string.

`cols` *Colours to use for plotting.*

default: `"turbo"` | *scope:* dynamic & static

The colours used for plotting, passed to `openair::openColours()`. The default, `"turbo"`, is a rainbow palette with relatively perceptually uniform colours.

`alpha` *Transparency value for polar markers.*

default: `1` | *scope:* dynamic & static

A value between 0 (fully transparent) and 1 (fully opaque).

`theme` *Custom ggplot2 theme for the polar markers.*

default: `"default"` | *scope:* dynamic & static

A custom `ggplot2` theme to add to the polar markers. This should ideally be a partial theme (i.e., `ggplot2::theme()`) over a complete theme (e.g., `ggplot2::theme_bw()`) as other arguments like `key` interact with the plot theme *before* custom themes are set, so would be overridden by a complete theme.

`key.position` *Legend position for individual marker legends.*

default: `"none"` | *scope:* dynamic & static

When `key.position` is not `"none"`, a key will be drawn for each individual marker. Potentially useful when `limits = "free"`, but of limited use otherwise.

`legend` *Draw a shared legend?*

default: `TRUE` | *scope:* dynamic & static

When all markers share the same colour scale, should a shared legend be created at the side of the map? This is the case for:

- *Continuous* colour scales, when all markers share the same `limits` (e.g., when `limits != "free"` in `polarMap()`), which are shown as a colour bar. This legend is not drawn if the colour scale has been transformed using `trans` (e.g., `trans = "log10"`), as it would be misleading.
- *Discrete* colour scales, when all markers are binned into the same intervals (e.g., when `breaks` are passed to `polarMap()`, or in `windroseMap()`), which are shown as a set of labelled colours.

`legend.position`

Position of the shared legend.

default: `NULL` | *scope:* dynamic & static

When `legend = TRUE`, where should the legend be placed?

- *Dynamic*: One of "topright", "topleft", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLegend()`.
- *Static*: One of "top", "right", "bottom" or "left". Passed to the `legend.position` argument of `ggplot2::theme()`.

`legend.title` *Title of the legend.*

default: `NULL` | *scope:* dynamic & static

By default, when `legend.title = NULL`, the function will attempt to provide a sensible legend title. `legend.title` allows users to overwrite this - for example, to include units or other contextual information. For *dynamic* maps, users may wish to use HTML tags to format the title.

`legend.title.autotext`

Automatically format the title of the legend?

default: `TRUE` | *scope:* dynamic & static

When `legend.title.autotext = TRUE`, `legend.title` will be first run through `quickTextHTML()` (*dynamic*) or `openair::quickText()` (*static*).

`control.collapsed`

Show the layer control as a collapsed?

default: `FALSE` | *scope:* dynamic

For *dynamic* maps, should the "layer control" interface be collapsed? If `TRUE`, users will have to hover over an icon to view the options.

`control.position`

Position of the layer control menu

default: `"topright"` | *scope:* dynamic

When `type != NULL`, or multiple pollutants are specified, where should the "layer control" interface be placed? One of "topleft", "topright", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLayersControl()`.

`control.autotext`

Automatically format the content of the layer control menu?

default: `TRUE` | *scope:* dynamic

When `control.autotext = TRUE`, the content of the "layer control" interface will be first run through `quickTextHTML()`.

`d.icon` *The diameter of the plot on the map in pixels.*

default: `200` | *scope:* dynamic & static

This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`d.fig` *The diameter of the plots to be produced using* `{openair}` *in inches.*

default: `3.5` | *scope:* dynamic & static

This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`static` *Produce a static map?*

default: `FALSE`

This controls whether a *dynamic* or *static* map is produced. The former is the default and is broadly more useful, but the latter may be preferable for DOCX or PDF outputs (e.g., academic papers).

`static.nrow` *Number of rows in a static map.*

default: `NULL` | *scope:* static

Controls the number of rows of panels on a static map when multiple `pollutants` or `type` are specified; passed to the `nrow` argument of `ggplot2::facet_wrap()`. The default, `NULL`, results in a roughly square grid of panels.

`api_key` *API key for certain 'static' base map providers.*

default: `NULL` | *scope:* static

Passed to the `api_key` argument of `ggspatial::annotation_map_tile()`. Needed for certain `providers` (e.g., `"cartolight"`).

`progress` *Show a progress bar?*

default: `TRUE` | *scope:* dynamic & static

By default, a progress bar is shown to visualise the function's progress creating individual polar markers. This option allows this to be turned off, if desired.

`...` Arguments passed on to `openair::percentileRose`

`mydata` A data frame minimally containing a decimal wind direction and a numeric field to plot.

`ws` The name of the column in `mydata` representing the wind speed. Defaults to `"ws"`.

`wd` The name of the column in `mydata` representing the decimal wind direction, 0 to 360 where 0/360 are North and 180 is South. Defaults to `"wd"`.

`smooth` Should the wind direction data be smoothed using a cyclic spline?

`method` When `method = "default"` the supplied percentiles by wind direction are calculated. When `method = "cpf"` the conditional probability function (CPF) is plotted and a single (usually high) percentile level is supplied. The CPF is defined as CPF = my/ny, where my is the number of samples in the wind sector y with mixing ratios greater than the *overall* percentile concentration, and ny is the total number of samples in the same wind sector (see Ashbaugh et al., 1985).

`angle` Default angle of "spokes" is when `smooth = FALSE`.

`mean` Show the mean by wind direction as a line?

`mean.lty` Line type for mean line.

`mean.lwd` Line width for mean line.

`mean.col` Line colour for mean line.

`fill` Should the percentile intervals be filled (default) or should lines be drawn (`fill = FALSE`).

`angle.scale` In radial plots (e.g., `polarPlot()`), the radial scale is drawn directly on the plot itself. While suitable defaults have been chosen, sometimes the placement of the scale may interfere with an interesting feature. `angle.scale` can take any value between `0` and `360` to place the scale at a different angle, or `FALSE` to move it to the side of the plots.

`offset` `offset` controls the size of the 'hole' in the middle and is expressed on a scale of `0` to `100`, where `0` is no hole and `100` is a hole that takes up the entire plotting area.

`auto.text` Either `TRUE` (default) or `FALSE`. If `TRUE` titles and axis labels will automatically try and format pollutant names and units properly, e.g., by subscripting the "2" in "NO2". Passed to `quickText()`.

`key.title` Used to set the title of the legend. The legend title is passed to `quickText()` if `auto.text = TRUE`.

`key.columns,key.rows` Number of columns or rows to be used in a categorical legend. With many categories a single column or row can make the key fit poorly in the plot. The user can thus choose to use several columns by setting `key.columns` to be less than the number of categories.

`plot` When `openair` plots are created they are automatically printed to the active graphics device. `plot = FALSE` deactivates this behaviour. This may be useful when the plot *data* is of more interest, or the plot is required to appear later (e.g., later in a Quarto document, or to be saved to a file).

`key` Deprecated; please use `key.position`. If `FALSE`, sets `key.position` to `"none"`.

`control` **Removed.** Please use `type`.

Value

Either:

- *Dynamic:* A leaflet object
- *Static:* A `ggplot2` object using `ggplot2::coord_sf()` coordinates with a `ggspatial` basemap

Parallel processing with mirai

Creating a directional analysis map can take a lot of time; each polar marker needs to be plot individually, and many of these require some expensive computations. `openairmaps` supports parallel processing with `{mirai}` to speed these computations up. Users may create workers by running `mirai::daemons()` in their R session. This requires the `{mirai}` and `{carrier}` packages, which are otherwise not needed; without daemons, markers are plotted one at a time.

```
mirai::daemons(4)
polarMap(polar_data, "no2")
```

Typically, spawning one fewer daemons than your number of available cores is a useful rule of thumb. Parallel processing will be most useful for the most computationally intensive plotting functions - i.e., polarMap() and annulusMap().

Customisation of static maps using ggplot2

As all static plots functions are ggplot2 figures, further customisation is possible using functions such as ggplot2::theme(), ggplot2::guides() and ggplot2::labs().

Subscripting pollutants (e.g., the "x" in "NOx") is achieved using the ggtext package. Therefore if you choose to override the plot theme, it is recommended to use [ggplot2::theme()] and [ggtext::element_markdown()] to define the strip.text parameter.

Legends can be removed using ggplot2::theme(legend.position = "none"), or further customised using ggplot2::guides() and either color = ggplot2::guide_colourbar() for continuous legends or color = ggplot2::guide_legend() for discrete legends.

The extent of a map can be adjusted using the xlim and ylim arguments of ggplot2::coord_sf().

```
polarMap(polar_data, "no2", static = TRUE) +
    ggplot2::coord_sf(
        xlim = c(-0.3, 0.2),
        ylim = c(51.2, 51.8)
    )
```

See Also

openair::percentileRose()

Other directional analysis maps: annulusMap(), diffMap(), freqMap(), polarMap(), pollroseMap(), windroseMap()

Examples

```
## Not run:
percentileMap(polar_data,
  pollutant = "nox",
  provider = "Esri.WorldGrayCanvas"
)

## End(Not run)
```

---

pointMap *Site average concentrations on dynamic and static maps*

Description

pointMap() summarises pollutant data at each monitoring site (e.g., taking the mean concentration) and shows the result as coloured circles on a map. It is the simplest way to compare concentrations across a network of sites. Unlike krigingMap() and voronoiMap(), no attempt is made to estimate concentrations away from the sites themselves. By default, these maps are dynamic and can be panned, zoomed, and otherwise interacted with. Using the static argument allows for static images to be produced instead.

Usage

```
pointMap(
  data,
  pollutant = NULL,
  statistic = "mean",
  percentile = 95,
  breaks = NULL,
  labels = NULL,
  limits = NULL,
  latitude = NULL,
  longitude = NULL,
  crs = 4326,
  type = NULL,
  popup = NULL,
  label = NULL,
  provider = "OpenStreetMap",
  cols = "turbo",
  alpha = 1,
  marker.border = "white",
  legend = TRUE,
  legend.position = NULL,
  legend.title = NULL,
  legend.title.autotext = TRUE,
  control.collapsed = FALSE,
  control.position = "topright",
  control.autotext = TRUE,
  static = FALSE,
  static.nrow = NULL,
  api_key = NULL,
  ...
)
```

Arguments

`data`	*Input data table with pollutant and geo-spatial information.* **required** | *scope:* dynamic & static A data frame. The data frame must contain at least one numeric column to interpolate, plus a decimal latitude and longitude (or X/Y coordinate used in conjunction with `crs`).
`pollutant`	*Pollutant name.* **required** | *scope:* dynamic & static The column name of the pollutant to plot. Multiple `pollutants` are prohibited by this function.
`statistic`	*Statistic for aggregating pollutant data.* *default:* `"mean"` | *scope:* dynamic & static Pollutant data will be aggregated by latitude & longitude; `statistic` controls how this is achieved. Possible statistics include:

- "mean": the arithmetic mean (using base::mean())
- "median": the median (middle) value (using stats::median())
- "max": the maximum value (using max())
- "min": the minimum value (using min())
- "sd": the standard deviation (using stats::sd())
- "percentile": a percentile value, defined using the percentile argument (using stats::quantile())

To show this value in marker popups or labels, include pollutant in popup or label (e.g., popup = c("site", "nox")).

percentile *The percentile when 'statistic = "percentile"*

default: 95 | *scope:* dynamic & static

The percentile level used when statistic = "percentile". The default is 95, representing 95%. Should be between 0 and 100.

labels, breaks *Discretise the map color scale.*

default: NULL | *scope:* dynamic & static

By default, a continuous colour scale is used. If breaks are provided, the colour scale will be discretised using cut(). labels can also be provided to customise how each factor level is labelled.

If the data extend beyond breaks, the outermost breaks are expanded to cover them so that no sites are left uncoloured. For example, breaks = c(50, 100, 150) would become c(25, 100, 200) for data ranging from 25 to 200.

limits *Specifier for the plot colour scale bounds.*

default: NULL | *scope:* dynamic & static

A numeric vector in the form c(lower, upper) used to define the colour scale. For example, limits = c(0, 100) would force the plot limits to span 0-100. If NULL, appropriate limits will be selected based on the range in data[[pollutant]]. Values outside of limits are given the colour of the nearest limit.

latitude, longitude

The decimal latitude(Y)/longitude(X).

default: NULL | *scope:* dynamic & static

Column names representing the decimal latitude and longitude (or other Y/X co-ordinate if using a different crs). If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).

crs *The coordinate reference system (CRS).*

default: 4326 | *scope:* dynamic & static

The coordinate reference system (CRS) of the data, passed to sf::st_crs(). By default this is EPSG:4326, the CRS associated with the commonly used latitude and longitude coordinates. Different coordinate systems can be specified using crs (e.g., crs = 27700 for the British National Grid). Note that non-lat/lng coordinate systems will be re-projected to EPSG:4326 for plotting on the map.

type *A method to condition the* data *for separate plotting.*

default: NULL | *scope:* dynamic & static

Used for splitting the input data into different groups, passed to the type argument of openair::cutData(). When type is specified:

- *Dynamic*: The different data splits can be toggled between using a "layer control" menu.
- *Static*: The data splits will each appear in a different panel.

`type` cannot be used if multiple `pollutant` columns have been provided.

`popup` *Content for marker popups on dynamic maps.*

default: `NULL` | *scope:* dynamic

Columns to be used as the HTML content for site marker popups on dynamic maps. Popups may be useful to show information about the individual sites (e.g., site names, codes, types, etc.). If a vector of column names are provided they are passed to `buildPopup()` using its default values.

The `pollutant` column shows the site `statistic` (i.e., what the markers are coloured by), labelled as such (e.g., "NO2 (max)"). Any other numeric columns have many values at each site, so are averaged and labelled "(mean)". Names given to `popup` (e.g., `c("Site NOx" = "nox")`) are used instead of these labels.

`label` *Content for marker hover-over on dynamic maps.*

default: `NULL` | *scope:* dynamic

Column to be used as the HTML content for hover-over labels. Labels are useful for the same reasons as popups, though are typically shorter. The `pollutant` column shows the site `statistic`, and other numeric columns are averaged at each site.

`provider` *The basemap(s) to be used.*

default: `"OpenStreetMap"` | *scope:* dynamic & static

The base map(s) to be used for the map. If not provided, will default to `"OpenStreetMap"`/`"osm"` for both dynamic and static maps.

- *Dynamic*: Any number of leaflet::providers. See `http://leaflet-extras.github.io/leaflet-providers/preview/` for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., `c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery")`)
- *Static*: One of the options listed in `rosm::osm.types()` (for example, `"osm"`, `"cartodark"`, `"cartolight"`, etc.).

There is some overlap in static and dynamic providers. For example, `{ggspatial}` uses "osm" to specify "OpenStreetMap". When static providers are provided to dynamic maps or vice versa, `{openairmaps}` will attempt to substitute the correct provider string.

`cols` *Colours to use for plotting.*

default: `"turbo"` | *scope:* dynamic & static

The colours used for plotting, passed to `openair::openColours()`. The default, `"turbo"`, is a rainbow palette with relatively perceptually uniform colours.

`alpha` *Transparency value for the site markers.*

default: `1` | *scope:* dynamic & static

A value between 0 (fully transparent) and 1 (fully opaque).

`marker.border` *Border colour to use for the site markers.*

default: `"white"` | *scope:* dynamic & static

Any valid HTML colour (e.g., a hex code). Use `NA` for no border.

`legend` *Draw a legend?*

default: `TRUE` | *scope:* dynamic & static

When `TRUE`, a legend will appear on the map identifying the colour scale.

`legend.position`

Position of the shared legend.

default: `NULL` | *scope:* dynamic & static

When `legend = TRUE`, where should the legend be placed?

- *Dynamic*: One of "topright", "topleft", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLegend()`.
- *Static*: One of "top", "right", "bottom" or "left". Passed to the `legend.position` argument of `ggplot2::theme()`.

`legend.title` *Title of the legend.*

default: `NULL` | *scope:* dynamic & static

By default, when `legend.title = NULL`, the function will attempt to provide a sensible legend title. `legend.title` allows users to overwrite this - for example, to include units or other contextual information. For *dynamic* maps, users may wish to use HTML tags to format the title.

`legend.title.autotext`

Automatically format the title of the legend?

default: `TRUE` | *scope:* dynamic & static

When `legend.title.autotext = TRUE`, `legend.title` will be first run through `quickTextHTML()` (*dynamic*) or `openair::quickText()` (*static*).

`control.collapsed`

Show the layer control as a collapsed?

default: `FALSE` | *scope:* dynamic

For *dynamic* maps, should the "layer control" interface be collapsed? If `TRUE`, users will have to hover over an icon to view the options.

`control.position`

Position of the layer control menu

default: `"topright"` | *scope:* dynamic

When `type != NULL`, or multiple pollutants are specified, where should the "layer control" interface be placed? One of "topleft", "topright", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLayersControl()`.

`control.autotext`

Automatically format the content of the layer control menu?

default: `TRUE` | *scope:* dynamic

When `control.autotext = TRUE`, the content of the "layer control" interface will be first run through `quickTextHTML()`.

`static` *Produce a static map?*

default: `FALSE`

This controls whether a *dynamic* or *static* map is produced. The former is the default and is broadly more useful, but the latter may be preferable for DOCX or PDF outputs (e.g., academic papers).

`static.nrow` *Number of rows in a static map.*

default: `NULL` | *scope:* static

Controls the number of rows of panels on a static map when multiple `pollutants` or `type` are specified; passed to the `nrow` argument of `ggplot2::facet_wrap()`. The default, `NULL`, results in a roughly square grid of panels.

`api_key` *API key for certain 'static' base map providers.*

default: `NULL` | *scope:* static

Passed to the `api_key` argument of `ggspatial::annotation_map_tile()`. Needed for certain `providers` (e.g., `"cartolight"`).

`...` Arguments passed on to `openair::cutData`

`names` By default, the columns created by `cutData()` are named after their `type` option. Specifying `names` defines other names for the columns, which map onto the `type` options in the same order they are given. The length of `names` should therefore be equal to the length of `type`.

`suffix` If `names` is not specified, `suffix` will be appended to any added columns that would otherwise overwrite existing columns. For example, `cutData(mydata, "nox", suffix = "_cuts")` would append a `nox_cuts` column rather than overwriting `nox`.

`hemisphere` Can be `"northern"` or `"southern"`, used to split data into seasons.

`n.levels` Number of quantiles to split numeric data into.

`wd.res` The number of wind direction bins; one of `4` (N/E/S/W), `8` (N/NE/E/SE/etc., the default), `16` (N/NNE/NE/ENE/etc.) or `32` (N/NbE/NNE/NEbN/etc.).

`start.day` What day of the week should the `type = "weekday"` start on? The user can change the start day by supplying an integer between 0 and 6. Sunday = 0, Monday = 1, ... For example to start the weekday plots on a Saturday, choose `start.day = 6`.

`start.season` What order should the season be. By default, the order is spring, summer, autumn, winter. `start.season = "winter"` would plot winter first.

`is.axis` A logical (`TRUE`/`FALSE`), used to request shortened cut labels for axes.

`local.tz` Used for identifying whether a date has daylight savings time (DST) applied or not. Examples include `local.tz = "Europe/London"`, `local.tz = "America/New_York"`, i.e., time zones that assume DST. https://en.wikipedia.org/wiki/List_of_zoneinfo_time_zones shows time zones that should be valid for most systems. It is important that the original data are in GMT (UTC) or a fixed offset from GMT.

`latitude,longitude` The decimal latitude and longitudes used when `type = "daylight"`. Note that locations west of Greenwich have negative longitudes.

`drop` How to handle empty factor levels. One of:

- `"default"`: Sensible defaults selected on a case-by-case basis for different `type` options.

- `"empty"`: Drop all empty factor levels.
- `"none"`: Retain all empty factor levels, where possible. For example, for `type = "hour"`, all factor levels from `0` and `23` will be represented.
- `"outside"`: Retain empty factor levels within the range of the data. For example, for `type = "hour"` when the data only contains data for 1 AM and 5 AM, the factor levels, `1`, `2`, `3`, `4` and `5` will be retained.

Some of these options only apply to certain `type` options. For example, for `type = "year"`, `"outside"` is equivalent to `"none"` as there is no fixed range of years to use in the `"none"` case.

Details

When `type` is used, the data are summarised separately for each `type` level, but all levels share the same colour scale so that they can be compared.

Value

Either:

- *Dynamic:* A leaflet object
- *Static:* A `ggplot2` object using `ggplot2::coord_sf()` coordinates with a `ggspatial` basemap

Customisation of static maps using ggplot2

As all static plots functions are `ggplot2` figures, further customisation is possible using functions such as `ggplot2::theme()`, `ggplot2::guides()` and `ggplot2::labs()`.

Subscripting pollutants (e.g., the "x" in "NOx") is achieved using the ggtext package. Therefore if you choose to override the plot theme, it is recommended to use `[ggplot2::theme()]` and `[ggtext::element_markdown()]` to define the `strip.text` parameter.

Legends can be removed using `ggplot2::theme(legend.position = "none")`, or further customised using `ggplot2::guides()` and either `color = ggplot2::guide_colourbar()` for continuous legends or `color = ggplot2::guide_legend()` for discrete legends.

The extent of a map can be adjusted using the `xlim` and `ylim` arguments of `ggplot2::coord_sf()`.

```
polarMap(polar_data, "no2", static = TRUE) +
    ggplot2::coord_sf(
        xlim = c(-0.3, 0.2),
        ylim = c(51.2, 51.8)
    )
```

See Also

`krigingMap()` and `voronoiMap()` to spatially interpolate between the sites.

Examples

```
## Not run:
# import ozone DAQI
daqi <-
  openair::importUKAQ(
    pollutant = "o3",
    year = 2020,
    source = "aurn",
    data_type = "daqi",
    meta = TRUE
  )

# maximum DAQI at each site
pointMap(
  daqi,
  pollutant = "poll_index",
  statistic = "max",
  breaks = seq(0.5, 10.5, 1),
  labels = as.character(1:10),
  legend.title = "Max O3 DAQI",
  cols = "daqi"
)

## End(Not run)
```

---

polar_data *Example data for polar mapping functions*

Description

The `polar_data` dataset is provided as an example dataset as part of the `openairmaps` package. The dataset contains hourly measurements of air pollutant concentrations, location and meteorological data.

Usage

```
polar_data
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 35040 rows and 14 columns.

Details

site,code The site name and code. Useful for use with the `popup` and `label` arguments in `openairmaps` functions.

date The date and time of the measurement

nox, no2, pm2.5, pm10 Pollutant concentrations

wd Wind direction, in degrees from North, as a numeric vector.

ws Wind speed, in m/s, as numeric vector.

visibility The visibility in metres.

air_temp Air temperature in degrees Celsius.

latitude, longitude Decimal latitude and longitude of the sites.

site.type Site type of the site (either "Urban Traffic" or "Urban Background").

Source

`polar_data` was compiled from data using the `openair::importUKAQ()` function from the `openair` package with meteorological data from the `worldmet` package.

Examples

```
polar_data
```

`polarMap` *Bivariate polar plots on dynamic and static maps*

Description

The `polarMap()` function creates a map using bivariate polar plots as markers. Any number of pollutants can be specified using the `pollutant` argument, and multiple layers of markers can be created using `type`. By default, these maps are dynamic and can be panned, zoomed, and otherwise interacted with. Using the `static` argument allows for static images to be produced instead.

Usage

```
polarMap(
  data,
  pollutant = NULL,
  x = "ws",
  limits = "free",
  breaks = NULL,
  upper = "fixed",
  latitude = NULL,
  longitude = NULL,
  crs = 4326,
  type = NULL,
  popup = NULL,
  label = NULL,
  provider = "OpenStreetMap",
  cols = "turbo",
  alpha = 1,
  theme = "default",
  key.position = "none",
```

```
  legend = TRUE,
  legend.position = NULL,
  legend.title = NULL,
  legend.title.autotext = TRUE,
  control.collapsed = FALSE,
  control.position = "topright",
  control.autotext = TRUE,
  d.icon = 200,
  d.fig = 3.5,
  static = FALSE,
  static.nrow = NULL,
  api_key = NULL,
  progress = TRUE,
  ...,
  control = NULL
)
```

Arguments

`data` *Input data table with pollutant, wind, and geo-spatial information.*

required | *scope:* dynamic & static

A data frame. The data frame must contain the data to plot the directional analysis marker, which includes wind speed (`ws`), wind direction (`wd`), and the column representing the concentration of a pollutant. In addition, `data` must include a decimal latitude and longitude (or X/Y coordinate used in conjunction with `crs`).

`pollutant` *Pollutant name(s).*

required | *scope:* dynamic & static

The column name(s) of the pollutant(s) to plot. If multiple pollutants are specified and a non-pairwise statistic is supplied, the `type` argument will no longer be able to be used and:

- *Dynamic*: The pollutants can be toggled between using a "layer control" menu.
- *Static*: The pollutants will each appear in a different panel.

Multiple `pollutants` prohibit the use of the `type` argument for non-pairwise statistics.

`x` *The radial axis variable.*

default: `"ws"` | *scope:* dynamic & static

The column name for the radial axis variable to use in `openair::polarPlot()`. Defaults to using wind speed, `"ws"`, but other meteorological variables such as ambient temperature or atmospheric stability may be useful.

`limits` *Specifier for the plot colour scale bounds.*

default: `"free"` | *scope:* dynamic & static

One of:

- `"fixed"` which ensures all of the markers use the same colour scale.
- `"free"` (the default) which allows all of the markers to use different colour scales.

- A numeric vector in the form `c(lower, upper)` used to define the colour scale. For example, `limits = c(0, 100)` would force the plot limits to span 0-100.

`limits` is ignored if `breaks` is also provided.

`breaks` *Specifier for a binned colour scale.*

default: `NULL` | *scope:* dynamic & static

By default, markers use a continuous colour scale (see `limits`). If `breaks` is provided, the colour scale is instead binned into discrete intervals. One of:

- A numeric vector of breakpoints. For example, `breaks = c(0, 50, 100, 1000)` would bin the colour scale into three intervals.
- A single number, which is the number of intervals to bin the colour scale into.
- A list created using `openair::breakOpts()`, which provides finer control over the binning method and allows custom `labels` to be defined.

In all cases, a single set of breakpoints is calculated across *all* of the markers, so that they share the same intervals and a shared legend can be drawn. If any of the markers' data fall outside of the given breakpoints, the lowest and/or highest breakpoints are extended to cover them. If `breaks` is provided, `limits` is ignored.

`upper` *Specifier for the polar plot radial axis upper boundary.*

default: `"fixed"` | *scope:* dynamic & static

One of:

- `"fixed"` (the default) which ensures all of the markers use the same radial axis scale.
- `"free"` which allows all of the markers to use different radial axis scales.
- A numeric value, used as the upper limit for the radial axis scale.

`latitude, longitude`

The decimal latitude(Y)/longitude(X).

default: `NULL` | *scope:* dynamic & static

Column names representing the decimal latitude and longitude (or other Y/X co-ordinate if using a different `crs`). If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).

`crs` *The coordinate reference system (CRS).*

default: `4326` | *scope:* dynamic & static

The coordinate reference system (CRS) of the data, passed to `sf::st_crs()`. By default this is EPSG:4326, the CRS associated with the commonly used latitude and longitude coordinates. Different coordinate systems can be specified using `crs` (e.g., `crs = 27700` for the British National Grid). Note that non-lat/lng coordinate systems will be re-projected to EPSG:4326 for plotting on the map.

`type` *A method to condition the* `data` *for separate plotting.*

default: `NULL` | *scope:* dynamic & static

Used for splitting the input data into different groups, passed to the `type` argument of `openair::cutData()`. When `type` is specified:

- *Dynamic*: The different data splits can be toggled between using a "layer control" menu.
- *Static*: The data splits will each appear in a different panel.

`type` cannot be used if multiple `pollutant` columns have been provided.

`popup` *Content for marker popups on dynamic maps.*

default: `NULL` | *scope:* dynamic

Columns to be used as the HTML content for marker popups on dynamic maps. Popups may be useful to show information about the individual sites (e.g., site names, codes, types, etc.). If a vector of column names are provided they are passed to `buildPopup()` using its default values.

Numeric columns (e.g., pollutant concentrations) have many values at each marker, so are averaged and labelled as such (e.g., "NO2 (mean)"). Names given to `popup` (e.g., `c("Mean NOx" = "nox")`) are used instead of these labels.

`label` *Content for marker hover-over on dynamic maps.*

default: `NULL` | *scope:* dynamic

Column to be used as the HTML content for hover-over labels. Labels are useful for the same reasons as popups, though are typically shorter. As with `popup`, numeric columns are averaged at each marker.

`provider` *The basemap(s) to be used.*

default: `"OpenStreetMap"` | *scope:* dynamic & static

The base map(s) to be used for the map. If not provided, will default to `"OpenStreetMap"`/`"osm"` for both dynamic and static maps.

- *Dynamic*: Any number of leaflet::providers. See `http://leaflet-extras.github.io/leaflet-providers/preview/` for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., `c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery")`)
- *Static*: One of the options listed in `rosm::osm.types()` (for example, `"osm"`, `"cartodark"`, `"cartolight"`, etc.).

There is some overlap in static and dynamic providers. For example, `{ggspatial}` uses "osm" to specify "OpenStreetMap". When static providers are provided to dynamic maps or vice versa, `{openairmaps}` will attempt to substitute the correct provider string.

`cols` *Colours to use for plotting.*

default: `"turbo"` | *scope:* dynamic & static

The colours used for plotting, passed to `openair::openColours()`. The default, `"turbo"`, is a rainbow palette with relatively perceptually uniform colours.

`alpha` *Transparency value for polar markers.*

default: `1` | *scope:* dynamic & static

A value between 0 (fully transparent) and 1 (fully opaque).

`theme` *Custom ggplot2 theme for the polar markers.*

default: `"default"` | *scope:* dynamic & static

A custom ggplot2 theme to add to the polar markers. This should ideally be a partial theme (i.e., ggplot2::theme()) over a complete theme (e.g., ggplot2::theme_bw()) as other arguments like key interact with the plot theme *before* custom themes are set, so would be overridden by a complete theme.

key.position *Legend position for individual marker legends.*

default: "none" | *scope:* dynamic & static

When key.position is not "none", a key will be drawn for each individual marker. Potentially useful when limits = "free", but of limited use otherwise.

legend *Draw a shared legend?*

default: TRUE | *scope:* dynamic & static

When all markers share the same colour scale, should a shared legend be created at the side of the map? This is the case for:

- *Continuous* colour scales, when all markers share the same limits (e.g., when limits != "free" in polarMap()), which are shown as a colour bar. This legend is not drawn if the colour scale has been transformed using trans (e.g., trans = "log10"), as it would be misleading.
- *Discrete* colour scales, when all markers are binned into the same intervals (e.g., when breaks are passed to polarMap(), or in windroseMap()), which are shown as a set of labelled colours.

legend.position

Position of the shared legend.

default: NULL | *scope:* dynamic & static

When legend = TRUE, where should the legend be placed?

- *Dynamic*: One of "topright", "topleft", "bottomleft" or "bottomright". Passed to the position argument of leaflet::addLegend().
- *Static*: One of "top", "right", "bottom" or "left". Passed to the legend.position argument of ggplot2::theme().

legend.title *Title of the legend.*

default: NULL | *scope:* dynamic & static

By default, when legend.title = NULL, the function will attempt to provide a sensible legend title. legend.title allows users to overwrite this - for example, to include units or other contextual information. For *dynamic* maps, users may wish to use HTML tags to format the title.

legend.title.autotext

Automatically format the title of the legend?

default: TRUE | *scope:* dynamic & static

When legend.title.autotext = TRUE, legend.title will be first run through quickTextHTML() (*dynamic*) or openair::quickText() (*static*).

control.collapsed

Show the layer control as a collapsed?

default: FALSE | *scope:* dynamic

For *dynamic* maps, should the "layer control" interface be collapsed? If TRUE, users will have to hover over an icon to view the options.

`control.position`

Position of the layer control menu

default: `"topright"` | *scope:* dynamic

When `type != NULL`, or multiple pollutants are specified, where should the "layer control" interface be placed? One of "topleft", "topright", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLayersControl()`.

`control.autotext`

Automatically format the content of the layer control menu?

default: `TRUE` | *scope:* dynamic

When `control.autotext = TRUE`, the content of the "layer control" interface will be first run through `quickTextHTML()`.

`d.icon` *The diameter of the plot on the map in pixels.*

default: `200` | *scope:* dynamic & static

This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`d.fig` *The diameter of the plots to be produced using* `{openair}` *in inches.*

default: `3.5` | *scope:* dynamic & static

This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`static` *Produce a static map?*

default: `FALSE`

This controls whether a *dynamic* or *static* map is produced. The former is the default and is broadly more useful, but the latter may be preferable for DOCX or PDF outputs (e.g., academic papers).

`static.nrow` *Number of rows in a static map.*

default: `NULL` | *scope:* static

Controls the number of rows of panels on a static map when multiple `pollutants` or `type` are specified; passed to the `nrow` argument of `ggplot2::facet_wrap()`. The default, `NULL`, results in a roughly square grid of panels.

`api_key` *API key for certain 'static' base map providers.*

default: `NULL` | *scope:* static

Passed to the `api_key` argument of `ggspatial::annotation_map_tile()`. Needed for certain `providers` (e.g., `"cartolight"`).

`progress` *Show a progress bar?*

default: `TRUE` | *scope:* dynamic & static

By default, a progress bar is shown to visualise the function's progress creating individual polar markers. This option allows this to be turned off, if desired.

`...` Arguments passed on to `openair::polarPlot`

`mydata` A data frame minimally containing a decimal wind direction, another variable to plot in polar coordinates (the default is a column `"ws"` — wind speed) and a pollutant. Should also contain `date` if plots by time period are required.

wd The name of the column in mydata representing the decimal wind direction, 0 to 360 where 0/360 are North and 180 is South. Defaults to "wd".

statistic The statistic that should be applied to each wind speed/direction bin. Because of the smoothing involved, the colour scale for some of these statistics is only to provide an indication of overall pattern and should not be interpreted in concentration units e.g. for statistic = "weighted.mean" where the bin mean is multiplied by the bin frequency and divided by the total frequency. In many cases using polarFreq will be better. Setting statistic = "weighted.mean" can be useful because it provides an indication of the concentration * frequency of occurrence and will highlight the wind speed/direction conditions that dominate the overall mean.Can be:

- "mean" (default), "median", "max" (maximum), "frequency". "stdev" (standard deviation), "weighted.mean".
- statistic = "nwr" Implements the Non-parametric Wind Regression approach of Henry et al. (2009) that uses kernel smoothers. The openair implementation is not identical because Gaussian kernels are used for both wind direction and speed. The smoothing is controlled by ws_spread and wd_spread.
- statistic = "cpf" the conditional probability function (CPF) is plotted and a single (usually high) percentile level is supplied. The CPF is defined as CPF = my/ny, where my is the number of samples in the y bin (by default a wind direction, wind speed interval) with mixing ratios greater than the *overall* percentile concentration, and ny is the total number of samples in the same wind sector (see Ashbaugh et al., 1985). Note that percentile intervals can also be considered; see percentile for details.
- When statistic = "r" or statistic = "Pearson", the Pearson correlation coefficient is calculated for *two* pollutants. The calculation involves a weighted Pearson correlation coefficient, which is weighted by Gaussian kernels for wind direction an the radial variable (by default wind speed). More weight is assigned to values close to a wind speed-direction interval. Kernel weighting is used to ensure that all data are used rather than relying on the potentially small number of values in a wind speed-direction interval.
- When statistic = "Spearman", the Spearman correlation coefficient is calculated for *two* pollutants. The calculation involves a weighted Spearman correlation coefficient, which is weighted by Gaussian kernels for wind direction an the radial variable (by default wind speed). More weight is assigned to values close to a wind speed-direction interval. Kernel weighting is used to ensure that all data are used rather than relying on the potentially small number of values in a wind speed-direction interval.
- "robust_slope" is another option for pair-wise statistics and "quantile.slope", which uses quantile regression to estimate the slope for a particular quantile level (see also tau for setting the quantile level).
- "york_slope" is another option for pair-wise statistics which uses the *York regression method* to estimate the slope. In this method the un-

certainties in x and y are used in the determination of the slope. The uncertainties are provided by x_error and y_error — see below.

trans Should a transformation be applied to the colour scale? If the distribution of data is skewed, the default scale may be dominated by a few high values, so a log or square-root transform may mean the whole colour scale is better presented on the plot. Can be:

- FALSE, which performs no transform.
- TRUE, which uses an appropriate transform for the plot type (usually "log10").
- A scales 'transform' object (e.g., scales::transform_log10()).
- A character string corresponding to a scales transform function. Useful options include "sqrt", "log10", "log2", "log1p", "pseudo_log" and "reverse".

exclude.missing Setting this option to TRUE (the default) removes points from the plot that are too far from the original data. The smoothing routines will produce predictions at points where no data exist i.e. they predict. By removing the points too far from the original data produces a plot where it is clear where the original data lie. If set to FALSE missing data will be interpolated.

uncertainty Should the uncertainty in the calculated surface be shown? If TRUE three plots are produced on the same scale showing the predicted surface together with the estimated lower and upper uncertainties at the 95% confidence interval. Calculating the uncertainties is useful to understand whether features are real or not. For example, at high wind speeds where there are few data there is greater uncertainty over the predicted values. The uncertainties are calculated using the GAM and weighting is done by the frequency of measurements in each wind speed-direction bin. Note that if uncertainties are calculated then the type is set to "default".

percentile If statistic = "percentile" then percentile is used, expressed from 0 to 100. Note that the percentile value is calculated in the wind speed, wind direction 'bins'. For this reason it can also be useful to set min.bin to ensure there are a sufficient number of points available to estimate a percentile. See quantile for more details of how percentiles are calculated. percentile is also used for the Conditional Probability Function (CPF) plots. percentile can be of length two, in which case the percentile *interval* is considered for use with CPF. For example, percentile = c(90, 100) will plot the CPF for concentrations between the 90 and 100th percentiles. Percentile intervals can be useful for identifying specific sources. In addition, percentile can also be of length 3. The third value is the 'trim' value to be applied. When calculating percentile intervals many can cover very low values where there is no useful information. The trim value ensures that values greater than or equal to the trim * mean value are considered *before* the percentile intervals are calculated. The effect is to extract more detail from many source signatures. See the manual for examples. Finally, if the trim value is less than zero the percentile range is interpreted as absolute concentration values and subsetting is carried out directly.

weights At the edges of the plot there may only be a few data points in each wind speed-direction interval, which could in some situations distort the

plot if the concentrations are high. `weights` applies a weighting to reduce their influence. For example and by default if only a single data point exists then the weighting factor is 0.25 and for two points 0.5. To not apply any weighting and use the data as is, use `weights = c(1, 1, 1)`.
An alternative to down-weighting these points they can be removed altogether using `min.bin`.

`min.bin` The minimum number of points allowed in a wind speed/wind direction bin. The default is 1. A value of two requires at least 2 valid records in each bin an so on; bins with less than 2 valid records are set to NA. Care should be taken when using a value > 1 because of the risk of removing real data points. It is recommended to consider your data with care. Also, the `polarFreq` function can be of use in such circumstances.

`col.na` When `min.bin` is > 1 it can be useful to show where data are removed on the plots. This is done by shading the missing data in `col.na`. To not highlight missing data when `min.bin` > 1 choose `col.na = "transparent"`.

`angle.scale` In radial plots (e.g., `polarPlot()`), the radial scale is drawn directly on the plot itself. While suitable defaults have been chosen, sometimes the placement of the scale may interfere with an interesting feature. `angle.scale` can take any value between `0` and `360` to place the scale at a different angle, or `FALSE` to move it to the side of the plots.

`units` The units shown on the polar axis scale.

`force.positive` The default is `TRUE`. Sometimes if smoothing data with steep gradients it is possible for predicted values to be negative. `force.positive = TRUE` ensures that predictions remain positive. This is useful for several reasons. First, with lots of missing data more interpolation is needed and this can result in artefacts because the predictions are too far from the original data. Second, if it is known beforehand that the data are all positive, then this option carries that assumption through to the prediction. The only likely time where setting `force.positive = FALSE` would be if background concentrations were first subtracted resulting in data that is legitimately negative. For the vast majority of situations it is expected that the user will not need to alter the default option.

`k` This is the smoothing parameter used by the `gam` function in package `mgcv`. Typically, value of around 100 (the default) seems to be suitable and will resolve important features in the plot. The most appropriate choice of `k` is problem-dependent; but extensive testing of polar plots for many different problems suggests a value of `k` of about 100 is suitable. Setting `k` to higher values will not tend to affect the surface predictions by much but will add to the computation time. Lower values of `k` will increase smoothing. Sometimes with few data to plot `polarPlot` will fail. Under these circumstances it can be worth lowering the value of `k`.

`normalise` If `TRUE` concentrations are normalised by dividing by their mean value. This is done *after* fitting the smooth surface. This option is particularly useful if one is interested in the patterns of concentrations for several pollutants on different scales e.g. NOx and CO. Often useful if more than one `pollutant` is chosen.

`key.title` Used to set the title of the legend. The legend title is passed to `quickText()` if `auto.text = TRUE`.

`key.columns,key.rows` Number of columns or rows to be used in a categorical legend. With many categories a single column or row can make the key fit poorly in the plot. The user can thus choose to use several columns by setting `key.columns` to be less than the number of categories.

`auto.text` Either `TRUE` (default) or `FALSE`. If `TRUE` titles and axis labels will automatically try and format pollutant names and units properly, e.g., by subscripting the "2" in "NO2". Passed to `quickText()`.

`ws_spread` The value of sigma used for Gaussian kernel weighting of wind speed when `statistic = "nwr"` or when correlation and regression statistics are used such as *r*. Default is `0.5`.

`wd_spread` The value of sigma used for Gaussian kernel weighting of wind direction when `statistic = "nwr"` or when correlation and regression statistics are used such as *r*. Default is `4`.

`x_error` The x error / uncertainty used when `statistic = "york_slope"`.

`y_error` The y error / uncertainty used when `statistic = "york_slope"`.

`kernel` Type of kernel used for the weighting procedure for when correlation or regression techniques are used. Only `"gaussian"` is supported but this may be enhanced in the future.

`formula.label` When pair-wise statistics such as regression slopes are calculated and plotted, should a formula label be displayed? `formula.label` will also determine whether concentration information is printed when `statistic = "cpf"`.

`tau` The quantile to be estimated when `statistic` is set to `"quantile.slope"`. Default is `0.5` which is equal to the median and will be ignored if `"quantile.slope"` is not used.

`plot` When `openair` plots are created they are automatically printed to the active graphics device. `plot = FALSE` deactivates this behaviour. This may be useful when the plot *data* is of more interest, or the plot is required to appear later (e.g., later in a Quarto document, or to be saved to a file).

`key` Deprecated; please use `key.position`. If `FALSE`, sets `key.position` to `"none"`.

`control` **Removed.** Please use `type`.

Value

Either:

- *Dynamic:* A leaflet object
- *Static:* A `ggplot2` object using `ggplot2::coord_sf()` coordinates with a `ggspatial` basemap

Parallel processing with mirai

Creating a directional analysis map can take a lot of time; each polar marker needs to be plot individually, and many of these require some expensive computations. `openairmaps` supports parallel processing with `{mirai}` to speed these computations up. Users may create workers by running `mirai::daemons()` in their R session. This requires the `{mirai}` and `{carrier}` packages, which are otherwise not needed; without daemons, markers are plotted one at a time.

```
mirai::daemons(4)
polarMap(polar_data, "no2")
```

Typically, spawning one fewer daemons than your number of available cores is a useful rule of thumb. Parallel processing will be most useful for the most computationally intensive plotting functions - i.e., `polarMap()` and `annulusMap()`.

Customisation of static maps using ggplot2

As all static plots functions are `ggplot2` figures, further customisation is possible using functions such as `ggplot2::theme()`, `ggplot2::guides()` and `ggplot2::labs()`.

Subscripting pollutants (e.g., the "x" in "NOx") is achieved using the ggtext package. Therefore if you choose to override the plot theme, it is recommended to use `[ggplot2::theme()]` and `[ggtext::element_markdown()]` to define the `strip.text` parameter.

Legends can be removed using `ggplot2::theme(legend.position = "none")`, or further customised using `ggplot2::guides()` and either `color = ggplot2::guide_colourbar()` for continuous legends or `color = ggplot2::guide_legend()` for discrete legends.

The extent of a map can be adjusted using the `xlim` and `ylim` arguments of `ggplot2::coord_sf()`.

```
polarMap(polar_data, "no2", static = TRUE) +
    ggplot2::coord_sf(
        xlim = c(-0.3, 0.2),
        ylim = c(51.2, 51.8)
    )
```

See Also

`openair::polarPlot()`

Other directional analysis maps: `annulusMap()`, `diffMap()`, `freqMap()`, `percentileMap()`, `pollroseMap()`, `windroseMap()`

Examples

```
## Not run:
polarMap(polar_data,
  pollutant = "nox",
  x = "ws",
  provider = "Esri.WorldGrayCanvas"
)

## End(Not run)
```

pollroseMap *Pollution roses on dynamic and static maps*

Description

The pollroseMap() function creates a map using pollution roses as markers. Any number of pollutants can be specified using the pollutant argument, and multiple layers of markers can be created using type. By default, these maps are dynamic and can be panned, zoomed, and otherwise interacted with. Using the static argument allows for static images to be produced instead.

Usage

```
pollroseMap(
  data,
  pollutant = NULL,
  statistic = "prop.count",
  breaks = NULL,
  latitude = NULL,
  longitude = NULL,
  crs = 4326,
  type = NULL,
  popup = NULL,
  label = NULL,
  provider = "OpenStreetMap",
  cols = "turbo",
  alpha = 1,
  theme = "default",
  key.position = "none",
  legend = TRUE,
  legend.position = NULL,
  legend.title = NULL,
  legend.title.autotext = TRUE,
  control.collapsed = FALSE,
  control.position = "topright",
  control.autotext = TRUE,
  d.icon = 200,
  d.fig = 3.5,
  static = FALSE,
  static.nrow = NULL,
  api_key = NULL,
  progress = TRUE,
  ...,
  control = NULL
)
```

Arguments

data *Input data table with pollutant, wind, and geo-spatial information.*

required | *scope:* dynamic & static

A data frame. The data frame must contain the data to plot the directional analysis marker, which includes wind speed (`ws`), wind direction (`wd`), and the column representing the concentration of a pollutant. In addition, `data` must include a decimal latitude and longitude (or X/Y coordinate used in conjunction with `crs`).

`pollutant` *Pollutant name(s).*

required | *scope:* dynamic & static

The column name(s) of the pollutant(s) to plot. If multiple pollutants are specified and a non-pairwise statistic is supplied, the `type` argument will no longer be able to be used and:

- *Dynamic*: The pollutants can be toggled between using a "layer control" menu.
- *Static*: The pollutants will each appear in a different panel.

Multiple `pollutants` prohibit the use of the `type` argument for non-pairwise statistics.

`statistic` *The statistic to be applied to each data bin in the plot*

default: `"prop.mean"` | *scope:* dynamic & static

Options currently include `"prop.count"`, `"prop.mean"` and `"abs.count"`. `"prop.count"` sizes bins according to the proportion of the frequency of measurements. Similarly, `"prop.mean"` sizes bins according to their relative contribution to the mean. `"abs.count"` provides the absolute count of measurements in each bin.

`breaks` *Specifier for the number of breaks of the colour axis.*

default: `NULL` | *scope:* dynamic & static

Most commonly, the number of break points. If not specified, each marker will independently break its supplied data at approximately 6 sensible break points. When `breaks` are specified, all markers will use the same break points. Breaks can also be used to set specific break points. For example, the argument `breaks = c(0, 1, 10, 100)` breaks the data into segments <1, 1-10, 10-100, >100.

`latitude, longitude`

The decimal latitude(Y)/longitude(X).

default: `NULL` | *scope:* dynamic & static

Column names representing the decimal latitude and longitude (or other Y/X coordinate if using a different `crs`). If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).

`crs` *The coordinate reference system (CRS).*

default: `4326` | *scope:* dynamic & static

The coordinate reference system (CRS) of the data, passed to `sf::st_crs()`. By default this is EPSG:4326, the CRS associated with the commonly used latitude and longitude coordinates. Different coordinate systems can be specified using `crs` (e.g., `crs = 27700` for the British National Grid). Note that non-lat/lng coordinate systems will be re-projected to EPSG:4326 for plotting on the map.

`type` *A method to condition the* `data` *for separate plotting.*

default: `NULL` | *scope:* dynamic & static

Used for splitting the input data into different groups, passed to the `type` argument of `openair::cutData()`. When `type` is specified:

- *Dynamic*: The different data splits can be toggled between using a "layer control" menu.
- *Static*: The data splits will each appear in a different panel.

`type` cannot be used if multiple `pollutant` columns have been provided.

`popup` *Content for marker popups on dynamic maps.*

default: `NULL` | *scope:* dynamic

Columns to be used as the HTML content for marker popups on dynamic maps. Popups may be useful to show information about the individual sites (e.g., site names, codes, types, etc.). If a vector of column names are provided they are passed to `buildPopup()` using its default values.

Numeric columns (e.g., pollutant concentrations) have many values at each marker, so are averaged and labelled as such (e.g., "NO2 (mean)"). Names given to `popup` (e.g., `c("Mean NOx" = "nox")`) are used instead of these labels.

`label` *Content for marker hover-over on dynamic maps.*

default: `NULL` | *scope:* dynamic

Column to be used as the HTML content for hover-over labels. Labels are useful for the same reasons as popups, though are typically shorter. As with `popup`, numeric columns are averaged at each marker.

`provider` *The basemap(s) to be used.*

default: `"OpenStreetMap"` | *scope:* dynamic & static

The base map(s) to be used for the map. If not provided, will default to `"OpenStreetMap"`/`"osm"` for both dynamic and static maps.

- *Dynamic*: Any number of leaflet::providers. See `http://leaflet-extras.github.io/leaflet-providers/preview/` for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., `c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery")`)
- *Static*: One of the options listed in `rosm::osm.types()` (for example, `"osm"`, `"cartodark"`, `"cartolight"`, etc.).

There is some overlap in static and dynamic providers. For example, `{ggspatial}` uses "osm" to specify "OpenStreetMap". When static providers are provided to dynamic maps or vice versa, `{openairmaps}` will attempt to substitute the correct provider string.

`cols` *Colours to use for plotting.*

default: `"turbo"` | *scope:* dynamic & static

The colours used for plotting, passed to `openair::openColours()`. The default, `"turbo"`, is a rainbow palette with relatively perceptually uniform colours.

`alpha` *Transparency value for polar markers.*

default: `1` | *scope:* dynamic & static

A value between 0 (fully transparent) and 1 (fully opaque).

theme *Custom ggplot2 theme for the polar markers.*

default: "default" | *scope:* dynamic & static

A custom ggplot2 theme to add to the polar markers. This should ideally be a partial theme (i.e., ggplot2::theme()) over a complete theme (e.g., ggplot2::theme_bw()) as other arguments like key interact with the plot theme *before* custom themes are set, so would be overridden by a complete theme.

key.position *Legend position for individual marker legends.*

default: "none" | *scope:* dynamic & static

When key.position is not "none", a key will be drawn for each individual marker. Potentially useful when limits = "free", but of limited use otherwise.

legend *Draw a shared legend?*

default: TRUE | *scope:* dynamic & static

When all markers share the same colour scale, should a shared legend be created at the side of the map? This is the case for:

- *Continuous* colour scales, when all markers share the same limits (e.g., when limits != "free" in polarMap()), which are shown as a colour bar. This legend is not drawn if the colour scale has been transformed using trans (e.g., trans = "log10"), as it would be misleading.
- *Discrete* colour scales, when all markers are binned into the same intervals (e.g., when breaks are passed to polarMap(), or in windroseMap()), which are shown as a set of labelled colours.

legend.position

Position of the shared legend.

default: NULL | *scope:* dynamic & static

When legend = TRUE, where should the legend be placed?

- *Dynamic*: One of "topright", "topleft", "bottomleft" or "bottomright". Passed to the position argument of leaflet::addLegend().
- *Static*: One of "top", "right", "bottom" or "left". Passed to the legend.position argument of ggplot2::theme().

legend.title *Title of the legend.*

default: NULL | *scope:* dynamic & static

By default, when legend.title = NULL, the function will attempt to provide a sensible legend title. legend.title allows users to overwrite this - for example, to include units or other contextual information. For *dynamic* maps, users may wish to use HTML tags to format the title.

legend.title.autotext

Automatically format the title of the legend?

default: TRUE | *scope:* dynamic & static

When legend.title.autotext = TRUE, legend.title will be first run through quickTextHTML() (*dynamic*) or openair::quickText() (*static*).

control.collapsed

Show the layer control as a collapsed?

default: FALSE | *scope:* dynamic

For *dynamic* maps, should the "layer control" interface be collapsed? If TRUE, users will have to hover over an icon to view the options.

`control.position`

Position of the layer control menu

default: `"topright"` | *scope:* dynamic

When `type != NULL`, or multiple pollutants are specified, where should the "layer control" interface be placed? One of "topleft", "topright", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLayersControl()`.

`control.autotext`

Automatically format the content of the layer control menu?

default: `TRUE` | *scope:* dynamic

When `control.autotext = TRUE`, the content of the "layer control" interface will be first run through `quickTextHTML()`.

`d.icon` *The diameter of the plot on the map in pixels.*

default: `200` | *scope:* dynamic & static

This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`d.fig` *The diameter of the plots to be produced using* `{openair}` *in inches.*

default: `3.5` | *scope:* dynamic & static

This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`static` *Produce a static map?*

default: `FALSE`

This controls whether a *dynamic* or *static* map is produced. The former is the default and is broadly more useful, but the latter may be preferable for DOCX or PDF outputs (e.g., academic papers).

`static.nrow` *Number of rows in a static map.*

default: `NULL` | *scope:* static

Controls the number of rows of panels on a static map when multiple `pollutants` or `type` are specified; passed to the `nrow` argument of `ggplot2::facet_wrap()`. The default, `NULL`, results in a roughly square grid of panels.

`api_key` *API key for certain 'static' base map providers.*

default: `NULL` | *scope:* static

Passed to the `api_key` argument of `ggspatial::annotation_map_tile()`. Needed for certain `providers` (e.g., `"cartolight"`).

`progress` *Show a progress bar?*

default: `TRUE` | *scope:* dynamic & static

By default, a progress bar is shown to visualise the function's progress creating individual polar markers. This option allows this to be turned off, if desired.

`...` Arguments passed on to `openair::pollutionRose`

`key.title` Used to set the title of the legend. The legend title is passed to `quickText()` if `auto.text = TRUE`.

`paddle` Either `TRUE` or `FALSE`. If `TRUE` plots rose using 'paddle' style spokes. If `FALSE` plots rose using 'wedge' style spokes.

`seg` `seg` determines with width of the segments. For example, `seg = 0.5` will produce segments 0.5 * `angle`.

`normalise` If `TRUE` each wind direction segment is normalised to equal one. This is useful for showing how the concentrations (or other parameters) contribute to each wind sector when the proportion of time the wind is from that direction is low. A line showing the probability that the wind directions is from a particular wind sector is also shown.

`key` Deprecated; please use `key.position`. If `FALSE`, sets `key.position` to `"none"`.

`control` **Removed.** Please use `type`.

Value

Either:

- *Dynamic:* A leaflet object
- *Static:* A `ggplot2` object using `ggplot2::coord_sf()` coordinates with a `ggspatial` basemap

Parallel processing with mirai

Creating a directional analysis map can take a lot of time; each polar marker needs to be plot individually, and many of these require some expensive computations. `openairmaps` supports parallel processing with `{mirai}` to speed these computations up. Users may create workers by running `mirai::daemons()` in their R session. This requires the `{mirai}` and `{carrier}` packages, which are otherwise not needed; without daemons, markers are plotted one at a time.

```
mirai::daemons(4)
polarMap(polar_data, "no2")
```

Typically, spawning one fewer daemons than your number of available cores is a useful rule of thumb. Parallel processing will be most useful for the most computationally intensive plotting functions - i.e., `polarMap()` and `annulusMap()`.

Customisation of static maps using ggplot2

As all static plots functions are `ggplot2` figures, further customisation is possible using functions such as `ggplot2::theme()`, `ggplot2::guides()` and `ggplot2::labs()`.

Subscripting pollutants (e.g., the "x" in "NOx") is achieved using the ggtext package. Therefore if you choose to override the plot theme, it is recommended to use `[ggplot2::theme()]` and `[ggtext::element_markdown()]` to define the `strip.text` parameter.

Legends can be removed using `ggplot2::theme(legend.position = "none")`, or further customised using `ggplot2::guides()` and either `color = ggplot2::guide_colourbar()` for continuous legends or `color = ggplot2::guide_legend()` for discrete legends.

The extent of a map can be adjusted using the `xlim` and `ylim` arguments of `ggplot2::coord_sf()`.

```
polarMap(polar_data, "no2", static = TRUE) +
    ggplot2::coord_sf(
```

```
        xlim = c(-0.3, 0.2),
        ylim = c(51.2, 51.8)
    )
```

See Also

`openair::pollutionRose()`

Other directional analysis maps: `annulusMap()`, `diffMap()`, `freqMap()`, `percentileMap()`, `polarMap()`, `windroseMap()`

Examples

```
## Not run:
pollroseMap(polar_data,
  pollutant = "nox",
  statistic = "prop.count",
  provider = "Esri.WorldGrayCanvas"
)

## End(Not run)
```

---

quickTextHTML *Automatic text formatting for* openairmaps

Description

Workhorse function that automatically applies routine text formatting to common pollutant names which may be used in the HTML widgets produced by `openairmaps`.

Usage

```
quickTextHTML(text)
```

Arguments

`text`	*A character vector.* **required** A character vector containing common pollutant names to be formatted. Commonly, this will insert super- and subscript HTML tags, e.g., "NO2" will be replaced with "NO2".

Details

`quickTextHTML()` is routine formatting lookup table. It screens the supplied character vector `text` and automatically applies formatting to any recognised character sub-series to properly render in HTML.

Value

a character vector

Author(s)

Jack Davison.

See Also

`openair::quickText()`, useful for non-HTML/static maps and plots

Examples

```
labs <- c("no2", "o3", "so2")
quickTextHTML(labs)
```

---

traj_data *Example data for trajectory mapping functions*

Description

The `traj_data` dataset is provided as an example dataset as part of the `openairmaps` package. The dataset contains HYSPLIT back trajectory data for air mass parcels arriving in London in 2009. It has been joined with air quality pollutant concentrations from the "London N. Kensington" AURN urban background monitoring site.

Usage

```
traj_data
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 5432 rows and 17 columns.

Details

date The arrival time of the air-mass

receptor The receptor number

year Trajectory year

month Trajectory month

day Trajectory day

hour Trajectory hour

hour.inc Trajectory hour offset from the arrival date

lat Latitude

lon Longitude

height Height of trajectory in m

pressure Pressure of the trajectory in Pa

date2 Date of the trajectory

nox Concentration of oxides of nitrogen (NO + NO2)

no2 Concentration of nitrogen dioxide (NO2)

o3 Concentration of ozone (O3)

pm10 Concentration of particulates (PM10)

pm2.5 Concentration of fine particulates (PM2.5)

Source

`traj_data` was compiled from data using the `openair::importTraj()` function from the `openair` package with air quality data from `openair::importAURN()` function.

Examples

```
traj_data
```

`trajLevelMap` *Trajectory level plots in* `leaflet`

Description

This function plots back trajectories on a `leaflet` map. This function requires that data are imported using the `openair::importTraj()` function.

Usage

```
trajLevelMap(
  data,
  latitude = NULL,
  longitude = NULL,
  pollutant = NULL,
  type = NULL,
  smooth = FALSE,
  statistic = "frequency",
  percentile = 90,
  lon.inc = 1,
  lat.inc = 1,
  breaks = NULL,
  min.bin = 1,
  .combine = NULL,
  sigma = 1.5,
  cols = "turbo",
```

```
  alpha = 0.5,
  tile.border = NA,
  provider = "OpenStreetMap",
  legend.position = "topright",
  legend.title = NULL,
  legend.title.autotext = TRUE,
  control.collapsed = FALSE,
  control.position = "topright",
  ...
)
```

Arguments

data	*A data frame containing a HYSPLIT trajectory, perhaps accessed with* openair::importTraj(). **required** A data frame containing HYSPLIT model outputs. If this data were not obtained using openair::importTraj().
latitude, longitude	*The decimal latitude/longitude.* *default:* NULL Column names representing the decimal latitude and longitude. If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively), such as the "lat" and "lon" columns returned by openair::importTraj().
pollutant	The pollutant of interest. Not required when statistic = "frequency".
type	*A method to condition the* data *for separate plotting.* *default:* NULL Used for splitting the trajectories into different groups which can be selected between using a "layer control" menu. Passed to openair::cutData().
smooth	Should the trajectory surface be smoothed? Defaults to FALSE. Note that, when smooth = TRUE, no popup information will be available.
statistic	One of: • "frequency" (the default) shows trajectory frequencies. • "hexbin", which is similar to "frequency" but shows a hexagonal grid of counts. • "difference" - in this case trajectories where the associated concentration is greater than percentile are compared with the the full set of trajectories to understand the differences in frequencies of the origin of air masses. The comparison is made by comparing the percentage change in gridded frequencies. For example, such a plot could show that the top 10\ to the east. • "pscf" for a Potential Source Contribution Function map. This statistic method interacts with percentile. • "cwt" for concentration weighted trajectories. • "sqtba" to undertake Simplified Quantitative Transport Bias Analysis. This statistic method interacts with .combine and sigma.

`percentile` The percentile concentration of `pollutant` against which the all trajectories are compared.

`lon.inc`, `lat.inc`

The longitude and latitude intervals to be used for binning data. If `statistic = "hexbin"`, the minimum value out of of `lon.inc` and `lat.inc` is passed to the `binwidth` argument of `ggplot2::geom_hex()`.

`breaks` `breaks` bins a continuous axis into discrete bins. It can either take a single number (e.g., `breaks = 5`) to split the scale into quantiles, a vector of numbers (e.g., `breaks = c(0, 50, 100, 200, 500)` to define specific break-points, or a named list. See `breakOpts()` for more details. Note that, in `trajLevel()`, each `statistic` provides a different default break strategy; to override those that bin the limits by default (e.g., `"frequency"`), pass `breaks = NULL` explicitly.

`min.bin` The minimum number of unique points in a grid cell. Counts below `min.bin` are set as missing.

`.combine` When statistic is "SQTBA" it is possible to combine lots of receptor locations to derive a single map. `.combine` identifies the column that differentiates different sites (commonly a column named `"site"`). Note that individual site maps are normalised first by dividing by their mean value.

`sigma` For the SQTBA approach `sigma` determines the amount of back trajectory spread based on the Gaussian plume equation. Values in the literature suggest 5.4 km after one hour. However, testing suggests lower values reveal source regions more effectively while not introducing too much noise.

`cols` *Colours to use for plotting.*

default: `"turbo"`

The colours used for plotting, passed to `openair::openColours()`.

`alpha` *Transparency value for trajectories.*

default: `0.5`

A value between `0` (fully transparent) and `1` (fully opaque).

`tile.border` Colour to use for the border of binned tiles. Defaults to `NA`, which draws no border.

`provider` *The basemap(s) to be used.*

default: `"OpenStreetMap"`

Any number of leaflet::providers. See `http://leaflet-extras.github.io/leaflet-providers/preview/` for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., `c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery")`)

`legend.position`

Position of the shared legend.

default: `"topright"`

Where should the legend be placed? One of "topright", "topleft", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLegend()`. `NULL` defaults to "topright".

`legend.title` *Title of the legend.*

default: `NULL`

By default, when `legend.title = NULL`, the function will attempt to provide a sensible legend title based on `colour`. `legend.title` allows users to overwrite this - for example, to include units or other contextual information. Users may wish to use HTML tags to format the title.

`legend.title.autotext`

Automatically format the title of the legend?

default: `TRUE`

When `legend.title.autotext = TRUE`, `legend.title` will be first run through `quickTextHTML()`.

`control.collapsed`

Show the layer control as a collapsed?

default: `FALSE`

Should the "layer control" interface be collapsed? If `TRUE`, users will have to hover over an icon to view the options.

`control.position`

Position of the layer control menu

default: `"topright"`

Where should the "layer control" interface be placed? One of "topleft", "topright", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLayersControl()`.

`...` Arguments passed on to `openair::cutData`

`names` By default, the columns created by `cutData()` are named after their `type` option. Specifying `names` defines other names for the columns, which map onto the `type` options in the same order they are given. The length of `names` should therefore be equal to the length of `type`.

`suffix` If `names` is not specified, `suffix` will be appended to any added columns that would otherwise overwrite existing columns. For example, `cutData(mydata, "nox", suffix = "_cuts")` would append a `nox_cuts` column rather than overwriting `nox`.

`hemisphere` Can be `"northern"` or `"southern"`, used to split data into seasons.

`n.levels` Number of quantiles to split numeric data into.

`wd.res` The number of wind direction bins; one of `4` (N/E/S/W), `8` (N/NE/E/SE/etc., the default), `16` (N/NNE/NE/ENE/etc.) or `32` (N/NbE/NNE/NEbN/etc.).

`start.day` What day of the week should the `type = "weekday"` start on? The user can change the start day by supplying an integer between 0 and 6. Sunday = 0, Monday = 1, ... For example to start the weekday plots on a Saturday, choose `start.day = 6`.

`start.season` What order should the season be. By default, the order is spring, summer, autumn, winter. `start.season = "winter"` would plot winter first.

`is.axis` A logical (`TRUE`/`FALSE`), used to request shortened cut labels for axes.

`local.tz` Used for identifying whether a date has daylight savings time (DST) applied or not. Examples include `local.tz = "Europe/London"`, `local.tz`

= "America/New_York", i.e., time zones that assume DST. https://en.wikipedia.org/wiki/List_of_zoneinfo_time_zones shows time zones that should be valid for most systems. It is important that the original data are in GMT (UTC) or a fixed offset from GMT.

latitude,longitude The decimal latitude and longitudes used when type = "daylight". Note that locations west of Greenwich have negative longitudes.

drop How to handle empty factor levels. One of:

- "default": Sensible defaults selected on a case-by-case basis for different type options.
- "empty": Drop all empty factor levels.
- "none": Retain all empty factor levels, where possible. For example, for type = "hour", all factor levels from 0 and 23 will be represented.
- "outside": Retain empty factor levels within the range of the data. For example, for type = "hour" when the data only contains data for 1 AM and 5 AM, the factor levels, 1, 2, 3, 4 and 5 will be retained.

Some of these options only apply to certain type options. For example, for type = "year", "outside" is equivalent to "none" as there is no fixed range of years to use in the "none" case.

Value

A leaflet object.

See Also

openair::trajLevel()

addTrajLevel() to add gridded trajectory statistics to any leaflet map

Other interactive trajectory maps: trajMap()

Examples

```
## Not run:
trajLevelMap(traj_data, pollutant = "pm2.5", statistic = "pscf", min.bin = 10)

## End(Not run)
```

---

trajMap *Trajectory line plots in* leaflet

Description

This function plots back trajectories on a leaflet map. This function requires that data are imported using the openair::importTraj() function. Options are provided to colour the individual trajectories (e.g., by pollutant concentrations) or create "layer control" menus to show/hide different layers.

Usage

```
trajMap(
  data,
  latitude = NULL,
  longitude = NULL,
  colour = NULL,
  type = NULL,
  cols = "turbo",
  alpha = 0.5,
  npoints = 12,
  provider = "OpenStreetMap",
  legend.position = "topright",
  legend.title = NULL,
  legend.title.autotext = TRUE,
  control.collapsed = FALSE,
  control.position = "topright",
  ...
)
```

Arguments

data	*A data frame containing a HYSPLIT trajectory, perhaps accessed with* `openair::importTraj()`. **required** A data frame containing HYSPLIT model outputs. If this data were not obtained using `openair::importTraj()`.
latitude, longitude	*The decimal latitude/longitude.* *default:* `NULL` Column names representing the decimal latitude and longitude. If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively), such as the "lat" and "lon" columns returned by `openair::importTraj()`.
colour	*Column to be used for colouring each trajectory.* *default:* `NULL` This column may be numeric, character, factor or date(time). This will commonly be a pollutant concentration which has been joined (e.g., by `dplyr::left_join()`) to the trajectory data by "date". Alternatively, a single colour (e.g., `"red"` or `"#FF0000"`) used for every trajectory. If `NULL`, trajectories are drawn in black.
type	*A method to condition the* `data` *for separate plotting.* *default:* `NULL` Used for splitting the trajectories into different groups which can be selected between using a "layer control" menu. Passed to `openair::cutData()`.
cols	*Colours to use for plotting.* *default:* `"turbo"` The colours used for plotting, passed to `openair::openColours()`.

alpha
: *Transparency value for trajectories.*

default: 0.5

A value between 0 (fully transparent) and 1 (fully opaque).

npoints
: *Interval at which points are placed along the trajectory paths.*

default: 12

A dot is placed every npoints along each full trajectory. For hourly back trajectories points are plotted every npoints hours. This helps to understand where the air masses were at particular times and get a feel for the speed of the air (points closer together correspond to slower moving air masses). Defaults to 12.

provider
: *The basemap(s) to be used.*

default: "OpenStreetMap"

Any number of leaflet::providers. See http://leaflet-extras.github.io/leaflet-providers/preview/ for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery"))

legend.position
: *Position of the shared legend.*

default: "topright"

Where should the legend be placed? One of "topright", "topleft", "bottomleft" or "bottomright". Passed to the position argument of leaflet::addLegend(). NULL defaults to "topright".

legend.title
: *Title of the legend.*

default: NULL

By default, when legend.title = NULL, the function will attempt to provide a sensible legend title based on colour. legend.title allows users to overwrite this - for example, to include units or other contextual information. Users may wish to use HTML tags to format the title.

legend.title.autotext
: *Automatically format the title of the legend?*

default: TRUE

When legend.title.autotext = TRUE, legend.title will be first run through quickTextHTML().

control.collapsed
: *Show the layer control as a collapsed?*

default: FALSE

Should the "layer control" interface be collapsed? If TRUE, users will have to hover over an icon to view the options.

control.position
: *Position of the layer control menu*

default: "topright"

Where should the "layer control" interface be placed? One of "topleft", "topright", "bottomleft" or "bottomright". Passed to the position argument of leaflet::addLayersControl().

... Arguments passed on to openair::cutData

names By default, the columns created by cutData() are named after their type option. Specifying names defines other names for the columns, which map onto the type options in the same order they are given. The length of names should therefore be equal to the length of type.

suffix If names is not specified, suffix will be appended to any added columns that would otherwise overwrite existing columns. For example, cutData(mydata, "nox", suffix = "_cuts") would append a nox_cuts column rather than overwriting nox.

hemisphere Can be "northern" or "southern", used to split data into seasons.

n.levels Number of quantiles to split numeric data into.

wd.res The number of wind direction bins; one of 4 (N/E/S/W), 8 (N/NE/E/SE/etc., the default), 16 (N/NNE/NE/ENE/etc.) or 32 (N/NbE/NNE/NEbN/etc.).

start.day What day of the week should the type = "weekday" start on? The user can change the start day by supplying an integer between 0 and 6. Sunday = 0, Monday = 1, ... For example to start the weekday plots on a Saturday, choose start.day = 6.

start.season What order should the season be. By default, the order is spring, summer, autumn, winter. start.season = "winter" would plot winter first.

is.axis A logical (TRUE/FALSE), used to request shortened cut labels for axes.

local.tz Used for identifying whether a date has daylight savings time (DST) applied or not. Examples include local.tz = "Europe/London", local.tz = "America/New_York", i.e., time zones that assume DST. https://en.wikipedia.org/wiki/List_of_zoneinfo_time_zones shows time zones that should be valid for most systems. It is important that the original data are in GMT (UTC) or a fixed offset from GMT.

latitude,longitude The decimal latitude and longitudes used when type = "daylight". Note that locations west of Greenwich have negative longitudes.

drop How to handle empty factor levels. One of:

- "default": Sensible defaults selected on a case-by-case basis for different type options.
- "empty": Drop all empty factor levels.
- "none": Retain all empty factor levels, where possible. For example, for type = "hour", all factor levels from 0 and 23 will be represented.
- "outside": Retain empty factor levels within the range of the data. For example, for type = "hour" when the data only contains data for 1 AM and 5 AM, the factor levels, 1, 2, 3, 4 and 5 will be retained.

Some of these options only apply to certain type options. For example, for type = "year", "outside" is equivalent to "none" as there is no fixed range of years to use in the "none" case.

Value

A leaflet object.

See Also

openair::trajPlot()

Other interactive trajectory maps: trajLevelMap()

Examples

```
## Not run:
trajMap(traj_data, colour = "pm10")

## End(Not run)
```

---

windroseMap *Wind roses on dynamic and static maps*

Description

The windroseMap() function creates a map using wind roses as markers. Multiple layers of markers can be created using the type argument. By default, these maps are dynamic and can be panned, zoomed, and otherwise interacted with. Using the static argument allows for static images to be produced instead.

Usage

```
windroseMap(
  data,
  ws.int = 2,
  breaks = 4,
  latitude = NULL,
  longitude = NULL,
  crs = 4326,
  type = NULL,
  popup = NULL,
  label = NULL,
  provider = "OpenStreetMap",
  cols = "turbo",
  alpha = 1,
  theme = "default",
  key.position = "none",
  legend = TRUE,
  legend.position = NULL,
  legend.title = NULL,
  legend.title.autotext = TRUE,
  control.collapsed = FALSE,
  control.position = "topright",
  control.autotext = TRUE,
  d.icon = 200,
```

```
  d.fig = 3.5,
  static = FALSE,
  static.nrow = NULL,
  api_key = NULL,
  progress = TRUE,
  ...,
  control = NULL
)
```

Arguments

`data`	*Input data table with wind and geo-spatial information.* **required** | *scope:* dynamic & static A data frame. The data frame must contain the data to plot the directional analysis marker, which includes wind speed (`ws`) and wind direction (`wd`). In addition, `data` must include a decimal latitude and longitude (or X/Y coordinate used in conjunction with `crs`).
`ws.int`	*The wind speed interval of the colour axis.* *default:* `2` | *scope:* dynamic & static The wind speed interval. Default is 2 m/s but for low met masts with low mean wind speeds a value of 1 or 0.5 m/s may be better.
`breaks`	*Specifier for the number of breaks of the colour axis.* *default:* `4` | *scope:* dynamic & static Most commonly, the number of break points for wind speed in `openair::windRose()`. For the `ws.int` default of `2`, the default `breaks`, `4`, generates the break points 2, 4, 6, and 8. Breaks can also be used to set specific break points. For example, the argument 'breaks = c(0, 1, 10, 100)" breaks the data into segments <1, 1-10, 10-100, >100.
`latitude, longitude`	*The decimal latitude(Y)/longitude(X).* *default:* `NULL` | *scope:* dynamic & static Column names representing the decimal latitude and longitude (or other Y/X coordinate if using a different `crs`). If not provided, will be automatically inferred from data by looking for a column named "lat"/"latitude" or "lon"/"lng"/"long"/"longitude" (case-insensitively).
`crs`	*The coordinate reference system (CRS).* *default:* `4326` | *scope:* dynamic & static The coordinate reference system (CRS) of the data, passed to `sf::st_crs()`. By default this is EPSG:4326, the CRS associated with the commonly used latitude and longitude coordinates. Different coordinate systems can be specified using `crs` (e.g., `crs = 27700` for the British National Grid). Note that non-lat/lng coordinate systems will be re-projected to EPSG:4326 for plotting on the map.
`type`	*A method to condition the* `data` *for separate plotting.* *default:* `NULL` | *scope:* dynamic & static Used for splitting the input data into different groups, passed to the `type` argument of `openair::cutData()`. When `type` is specified:

- *Dynamic*: The different data splits can be toggled between using a "layer control" menu.
- *Static*: The data splits will each appear in a different panel.

`type` cannot be used if multiple `pollutant` columns have been provided.

`popup` *Content for marker popups on dynamic maps.*

default: `NULL` | *scope:* dynamic

Columns to be used as the HTML content for marker popups on dynamic maps. Popups may be useful to show information about the individual sites (e.g., site names, codes, types, etc.). If a vector of column names are provided they are passed to `buildPopup()` using its default values.

Numeric columns (e.g., pollutant concentrations) have many values at each marker, so are averaged and labelled as such (e.g., "NO2 (mean)"). Names given to `popup` (e.g., `c("Mean NOx" = "nox")`) are used instead of these labels.

`label` *Content for marker hover-over on dynamic maps.*

default: `NULL` | *scope:* dynamic

Column to be used as the HTML content for hover-over labels. Labels are useful for the same reasons as popups, though are typically shorter. As with `popup`, numeric columns are averaged at each marker.

`provider` *The basemap(s) to be used.*

default: `"OpenStreetMap"` | *scope:* dynamic & static

The base map(s) to be used for the map. If not provided, will default to `"OpenStreetMap"`/`"osm"` for both dynamic and static maps.

- *Dynamic*: Any number of leaflet::providers. See `http://leaflet-extras.github.io/leaflet-providers/preview/` for a list of all base maps that can be used. If multiple base maps are provided, they can be toggled between using a "layer control" interface. By default, the interface will use the provider names as labels, but users can define their own using a named vector (e.g., `c("Default" = "OpenStreetMap", "Satellite" = "Esri.WorldImagery")`)
- *Static*: One of the options listed in `rosm::osm.types()` (for example, `"osm"`, `"cartodark"`, `"cartolight"`, etc.).

There is some overlap in static and dynamic providers. For example, `{ggspatial}` uses "osm" to specify "OpenStreetMap". When static providers are provided to dynamic maps or vice versa, `{openairmaps}` will attempt to substitute the correct provider string.

`cols` *Colours to use for plotting.*

default: `"turbo"` | *scope:* dynamic & static

The colours used for plotting, passed to `openair::openColours()`. The default, `"turbo"`, is a rainbow palette with relatively perceptually uniform colours.

`alpha` *Transparency value for polar markers.*

default: `1` | *scope:* dynamic & static

A value between 0 (fully transparent) and 1 (fully opaque).

`theme` *Custom ggplot2 theme for the polar markers.*

default: `"default"` | *scope:* dynamic & static

A custom ggplot2 theme to add to the polar markers. This should ideally be a partial theme (i.e., ggplot2::theme()) over a complete theme (e.g., ggplot2::theme_bw()) as other arguments like key interact with the plot theme *before* custom themes are set, so would be overridden by a complete theme.

key.position *Legend position for individual marker legends.*

default: "none" | *scope:* dynamic & static

When key.position is not "none", a key will be drawn for each individual marker. Potentially useful when limits = "free", but of limited use otherwise.

legend *Draw a shared legend?*

default: TRUE | *scope:* dynamic & static

When all markers share the same colour scale, should a shared legend be created at the side of the map? This is the case for:

- *Continuous* colour scales, when all markers share the same limits (e.g., when limits != "free" in polarMap()), which are shown as a colour bar. This legend is not drawn if the colour scale has been transformed using trans (e.g., trans = "log10"), as it would be misleading.
- *Discrete* colour scales, when all markers are binned into the same intervals (e.g., when breaks are passed to polarMap(), or in windroseMap()), which are shown as a set of labelled colours.

legend.position

Position of the shared legend.

default: NULL | *scope:* dynamic & static

When legend = TRUE, where should the legend be placed?

- *Dynamic*: One of "topright", "topleft", "bottomleft" or "bottomright". Passed to the position argument of leaflet::addLegend().
- *Static*: One of "top", "right", "bottom" or "left". Passed to the legend.position argument of ggplot2::theme().

legend.title *Title of the legend.*

default: NULL | *scope:* dynamic & static

By default, when legend.title = NULL, the function will attempt to provide a sensible legend title. legend.title allows users to overwrite this - for example, to include units or other contextual information. For *dynamic* maps, users may wish to use HTML tags to format the title.

legend.title.autotext

Automatically format the title of the legend?

default: TRUE | *scope:* dynamic & static

When legend.title.autotext = TRUE, legend.title will be first run through quickTextHTML() (*dynamic*) or openair::quickText() (*static*).

control.collapsed

Show the layer control as a collapsed?

default: FALSE | *scope:* dynamic

For *dynamic* maps, should the "layer control" interface be collapsed? If TRUE, users will have to hover over an icon to view the options.

`control.position`

Position of the layer control menu

default: `"topright"` | *scope:* dynamic

When `type != NULL`, or multiple pollutants are specified, where should the "layer control" interface be placed? One of "topleft", "topright", "bottomleft" or "bottomright". Passed to the `position` argument of `leaflet::addLayersControl()`.

`control.autotext`

Automatically format the content of the layer control menu?

default: `TRUE` | *scope:* dynamic

When `control.autotext = TRUE`, the content of the "layer control" interface will be first run through `quickTextHTML()`.

`d.icon` *The diameter of the plot on the map in pixels.*

default: `200` | *scope:* dynamic & static

This will affect the size of the individual polar markers. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`d.fig` *The diameter of the plots to be produced using* `{openair}` *in inches.*

default: `3.5` | *scope:* dynamic & static

This will affect the resolution of the markers on the map. Alternatively, a vector in the form `c(width, height)` can be provided if a non-circular marker is desired.

`static` *Produce a static map?*

default: `FALSE`

This controls whether a *dynamic* or *static* map is produced. The former is the default and is broadly more useful, but the latter may be preferable for DOCX or PDF outputs (e.g., academic papers).

`static.nrow` *Number of rows in a static map.*

default: `NULL` | *scope:* static

Controls the number of rows of panels on a static map when multiple `pollutants` or `type` are specified; passed to the `nrow` argument of `ggplot2::facet_wrap()`. The default, `NULL`, results in a roughly square grid of panels.

`api_key` *API key for certain 'static' base map providers.*

default: `NULL` | *scope:* static

Passed to the `api_key` argument of `ggspatial::annotation_map_tile()`. Needed for certain `providers` (e.g., `"cartolight"`).

`progress` *Show a progress bar?*

default: `TRUE` | *scope:* dynamic & static

By default, a progress bar is shown to visualise the function's progress creating individual polar markers. This option allows this to be turned off, if desired.

`...` Arguments passed on to `openair::windRose`

`mydata` A data frame containing fields `ws` and `wd`

`ws` The name of the column in `mydata` representing the wind speed. Defaults to `"ws"`.

`wd` The name of the column in `mydata` representing the decimal wind direction, 0 to 360 where 0/360 are North and 180 is South. Defaults to `"wd"`.

`ws2,wd2` The user can supply a second set of wind speed and wind direction values with which the first can be compared. See `pollutionRose()` for more details.

`angle` Default angle of "spokes" is 30. Other potentially useful angles are 45 and 10. Note that the width of the wind speed interval may need adjusting using `width`.

`calm.thresh` By default, conditions are considered to be calm when the wind speed is zero. The user can set a different threshold for calms be setting `calm.thresh` to a higher value. For example, `calm.thresh = 0.5` will identify wind speeds **below** 0.5 as calm.

`bias.corr` When `angle` does not divide exactly into 360 a bias is introduced in the frequencies when the wind direction is already supplied rounded to the nearest 10 degrees, as is often the case. For example, if `angle = 22.5`, N, E, S, W will include 3 wind sectors and all other angles will be two. A bias correction can made to correct for this problem. A simple method according to Applequist (2012) is used to adjust the frequencies.

`grid.line` Grid line interval to use. If `NULL`, as in default, this is assigned based on the available data range. However, it can also be forced to a specific value, e.g. `grid.line = 10`. `grid.line` can also be a list to control the interval, line type and colour. For example `grid.line = list(value = 10, lty = 5, col = "purple")`.

`width` For paddle = TRUE, the adjustment factor for width of wind speed intervals. For example, width = 1.5 will make the paddle width 1.5 times wider.

`seg` seg determines with width of the segments. For example, `seg = 0.5` will produce segments 0.5 * `angle`.

`auto.text` Either `TRUE` (default) or `FALSE`. If `TRUE` titles and axis labels will automatically try and format pollutant names and units properly, e.g., by subscripting the "2" in "NO2". Passed to `quickText()`.

`offset` `offset` controls the size of the 'hole' in the middle and is expressed on a scale of `0` to `100`, where `0` is no hole and `100` is a hole that takes up the entire plotting area.

`normalise` If `TRUE` each wind direction segment is normalised to equal one. This is useful for showing how the concentrations (or other parameters) contribute to each wind sector when the proportion of time the wind is from that direction is low. A line showing the probability that the wind directions is from a particular wind sector is also shown.

`max.freq` Controls the scaling used by setting the maximum value for the radial limits. This is useful to ensure several plots use the same radial limits.

`paddle` Either `TRUE` or `FALSE`. If `TRUE` plots rose using 'paddle' style spokes. If `FALSE` plots rose using 'wedge' style spokes.

`key.title` Used to set the title of the legend. The legend title is passed to `quickText()` if `auto.text = TRUE`.

`key.columns,key.rows` Number of columns or rows to be used in a categorical legend. With many categories a single column or row can make the key fit poorly in the plot. The user can thus choose to use several columns by setting `key.columns` to be less than the number of categories.

dig.lab The number of significant figures at which scientific number formatting is used in break point and key labelling. Default 5.

include.lowest Logical. If FALSE (the default), the first interval will be left exclusive and right inclusive. If TRUE, the first interval will be left and right inclusive. Passed to the include.lowest argument of cut().

statistic The statistic to be applied to each data bin in the plot. Options currently include "prop.count", "prop.mean" and "abs.count". The default "prop.count" sizes bins according to the proportion of the frequency of measurements. Similarly, "prop.mean" sizes bins according to their relative contribution to the mean. "abs.count" provides the absolute count of measurements in each bin.

pollutant Alternative data series to be sampled instead of wind speed. The windRose() default NULL is equivalent to pollutant = "ws". Use in pollutionRose().

annotate If TRUE then the percentage calm and mean values are printed in each panel together with a description of the statistic below the plot. If FALSE then only the statistic will be printed.

angle.scale In radial plots (e.g., polarPlot()), the radial scale is drawn directly on the plot itself. While suitable defaults have been chosen, sometimes the placement of the scale may interfere with an interesting feature. angle.scale can take any value between 0 and 360 to place the scale at a different angle, or FALSE to move it to the side of the plots.

border Border colour for shaded areas. Default is no border.

plot When openair plots are created they are automatically printed to the active graphics device. plot = FALSE deactivates this behaviour. This may be useful when the plot *data* is of more interest, or the plot is required to appear later (e.g., later in a Quarto document, or to be saved to a file).

key Deprecated; please use key.position. If FALSE, sets key.position to "none".

control **Removed.** Please use type.

Value

Either:

- *Dynamic:* A leaflet object
- *Static:* A ggplot2 object using ggplot2::coord_sf() coordinates with a ggspatial basemap

Parallel processing with mirai

Creating a directional analysis map can take a lot of time; each polar marker needs to be plot individually, and many of these require some expensive computations. openairmaps supports parallel processing with {mirai} to speed these computations up. Users may create workers by running mirai::daemons() in their R session. This requires the {mirai} and {carrier} packages, which are otherwise not needed; without daemons, markers are plotted one at a time.

```
mirai::daemons(4)
polarMap(polar_data, "no2")
```

Typically, spawning one fewer daemons than your number of available cores is a useful rule of thumb. Parallel processing will be most useful for the most computationally intensive plotting functions - i.e., `polarMap()` and `annulusMap()`.

Customisation of static maps using ggplot2

As all static plots functions are `ggplot2` figures, further customisation is possible using functions such as `ggplot2::theme()`, `ggplot2::guides()` and `ggplot2::labs()`.

Subscripting pollutants (e.g., the "x" in "NOx") is achieved using the ggtext package. Therefore if you choose to override the plot theme, it is recommended to use `[ggplot2::theme()]` and `[ggtext::element_markdown()]` to define the `strip.text` parameter.

Legends can be removed using `ggplot2::theme(legend.position = "none")`, or further customised using `ggplot2::guides()` and either `color = ggplot2::guide_colourbar()` for continuous legends or `color = ggplot2::guide_legend()` for discrete legends.

The extent of a map can be adjusted using the `xlim` and `ylim` arguments of `ggplot2::coord_sf()`.

```
polarMap(polar_data, "no2", static = TRUE) +
    ggplot2::coord_sf(
        xlim = c(-0.3, 0.2),
        ylim = c(51.2, 51.8)
    )
```

See Also

`openair::windRose()`

Other directional analysis maps: `annulusMap()`, `diffMap()`, `freqMap()`, `percentileMap()`, `polarMap()`, `pollroseMap()`

Examples

```
## Not run:
windroseMap(polar_data,
  provider = "Esri.WorldGrayCanvas"
)

## End(Not run)
```

Index

∗ **datasets**
  polar_data, 66
  traj_data, 85
∗ **directional analysis maps**
  annulusMap, 11
  diffMap, 23
  freqMap, 33
  percentileMap, 52
  polarMap, 67
  pollroseMap, 78
  windroseMap, 94
∗ **interactive trajectory maps**
  trajLevelMap, 86
  trajMap, 90
∗ **spatial interpolation maps**
  krigingMap, 41

addLayersControl(), *4*
addPolarDiffMarkers *(*addPolarMarkers*)*, 2
addPolarDiffMarkers(), *4*
addPolarMarkers, 2
addTrajLevel, 6
addTrajLevel(), *8*, *90*
addTrajPaths, 9
addTrajPaths(), *10*
annulusMap, 11
annulusMap(), *11*, *19*, *32*, *33*, *40*, *59*, *77*, *83*, *84*, *101*

base::mean(), *43*, *61*
breakOpts(), *7*, *29*, *88*
buildPopup, 20
buildPopup(), *14*, *25*, *35*, *44*, *55*, *62*, *70*, *80*, *96*

clearGroup(), *4*
convertPostcode, 22
cut(), *43*, *61*, *100*
cutData(), *25*, *47*, *64*, *89*, *93*

diffMap, 23
diffMap(), *20*, *23*, *24*, *40*, *59*, *77*, *84*, *101*
dplyr::left_join(), *91*

freqMap, 33
freqMap(), *20*, *33*, *59*, *77*, *84*, *101*

ggplot2::coord_sf(), *19*, *32*, *40*, *48*, *58*, *59*, *65*, *76*, *77*, *83*, *100*, *101*
ggplot2::facet_wrap(), *16*, *28*, *38*, *46*, *57*, *64*, *72*, *82*, *98*
ggplot2::geom_hex(), *7*, *88*
ggplot2::guides(), *19*, *32*, *40*, *48*, *59*, *65*, *77*, *83*, *101*
ggplot2::labs(), *19*, *32*, *40*, *48*, *59*, *65*, *77*, *83*, *101*
ggplot2::theme(), *14*, *15*, *19*, *26*, *27*, *32*, *36*, *37*, *40*, *46*, *48*, *55*, *56*, *59*, *63*, *65*, *71*, *77*, *81*, *83*, *97*, *101*
ggplot2::theme_bw(), *14*, *26*, *36*, *55*, *71*, *81*, *97*
ggspatial::annotation_map_tile(), *16*, *28*, *38*, *46*, *57*, *64*, *72*, *82*, *98*
ggtext, *19*, *32*, *40*, *48*, *59*, *65*, *77*, *83*, *101*
gstat::fit.variogram(), *47*
gstat::idw(), *47*
gstat::krige(), *47*
gstat::variogram(), *47*
gstat::vgm(), *47*

importUKAQ(), *50*

krigingMap, 41
krigingMap(), *41*, *43*, *47*, *59*, *65*

labelOptions(), *4*
leaflet(), *4*
leaflet::addCircleMarkers(), *2*, *10*
leaflet::addLayersControl(), *7–9*, *15*, *27*, *37*, *46*, *52*, *56*, *63*, *72*, *82*, *89*, *92*, *98*
leaflet::addLegend(), *8*, *15*, *26*, *37*, *45*, *52*, *56*, *63*, *71*, *81*, *88*, *92*, *97*

leaflet::addMarkers(), 2, 9
leaflet::addPolylines(), 10
leaflet::addRectangles(), 6, 8
leaflet::clearGroup(), 8, 9
leaflet::leaflet(), 4–6, 9
leaflet::providers, 14, 25, 36, 45, 51, 55, 62, 70, 80, 88, 92, 96

markerClusterOptions(), 4
max(), 43, 61
min(), 43, 61
mirai::daemons(), 19, 32, 40, 58, 76, 83, 100

networkMap, 49
networkMap(), 22

openair::breakOpts(), 13, 69
openair::cutData, 47, 64, 89, 93
openair::cutData(), 7, 13, 35, 44, 54, 61, 69, 80, 87, 91, 95
openair::importAURN(), 86
openair::importMeta, 50
openair::importMeta(), 49
openair::importTraj(), 6, 9, 86, 87, 90, 91
openair::importUKAQ(), 67
openair::openColours(), 8, 14, 26, 36, 45, 55, 62, 70, 80, 88, 91, 96
openair::percentileRose, 57
openair::percentileRose(), 4, 54, 59
openair::polarAnnulus, 16
openair::polarAnnulus(), 4, 20
openair::polarDiff(), 4, 5, 33
openair::polarFreq, 38
openair::polarFreq(), 4, 40
openair::polarPlot, 28, 72
openair::polarPlot(), 4, 5, 68, 77
openair::pollutionRose, 82
openair::pollutionRose(), 4, 84
openair::quickText(), 15, 27, 37, 46, 56, 63, 71, 81, 85, 97
openair::trajLevel(), 6, 8, 90
openair::trajPlot(), 94
openair::windRose, 98
openair::windRose(), 4, 95, 101

percentileMap, 52
percentileMap(), 20, 33, 40, 52, 77, 84, 101
pointMap, 59
pointMap(), 59

polar_data, 66
polarMap, 67
polarMap(), 2, 15, 19–21, 26, 32, 33, 37, 40, 56, 59, 67, 71, 77, 81, 83, 84, 97, 101
polarPlot(), 17, 24, 30, 39, 58, 75, 100
pollroseMap, 78
pollroseMap(), 20, 33, 40, 59, 77, 78, 101
pollutionRose(), 99, 100
popupOptions(), 4

quickText(), 18, 31, 39, 58, 75, 76, 82, 99
quickTextHTML, 84
quickTextHTML(), 8, 15, 16, 20, 27, 37, 46, 56, 63, 71, 72, 81, 82, 84, 89, 92, 97, 98

scales::transform_log10(), 17, 29, 39, 74
sf::st_as_sf(), 43
sf::st_crs(), 13, 25, 35, 44, 51, 54, 61, 69, 79, 95
shiny::shinyApp(), 5, 10
stats::median(), 43, 61
stats::quantile(), 43, 61
stats::sd(), 43, 61

terra::voronoi(), 48
tibble, 22
traj_data, 85
trajLevelMap, 86
trajLevelMap(), 6, 94
trajMap, 90
trajMap(), 9, 90

voronoiMap (krigingMap), 41
voronoiMap(), 41, 48, 59, 65

windRose(), 100
windroseMap, 94
windroseMap(), 15, 20, 26, 33, 37, 40, 56, 59, 71, 77, 81, 84, 94, 97