Package: qlcVisualize (via r-universe)

September 4, 2024

Title Visualization for Quantitative Language Comparison

Description Collection of visualizations as used in quantitative language comparison. Currently implemented are visualisations dealing nominal data with multiple levels (``level map'' and ``factor map''), and assistance for making weighted geographical Voronoi-maps (``weighted map'').

Version 0.4

Date 2024-07-31

URL https://github.com/cysouw/qlcVisualize

BugReports https://github.com/cysouw/qlcVisualize/issues

License GPL-3

Encoding UTF-8

Depends R (>= 3.5.0)

Imports alphahull, automap, cartogramR, concaveman, fields, geodata, gstat, mapplots, maps, MASS, methods, qlcMatrix, RSpectra, seriation, sf, sp, spatstat.geom, spatstat.random, stars

Suggests rnaturalearth, mapdata, lingtypology

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

RemoteUrl https://github.com/cysouw/qlcvisualize

RemoteRef HEAD

RemoteSha a688f501d3d6526d8261ae9f938d879ebb806bb5

Contents

`qlcVisualize-package` *Visualizations for Quantitative Language Comparison*

Description

A collection of specific visualisations of data as used in quantitative language comparison.

Details

Package:	qlcVisualize
Type:	Package
Version:	0.4
Date:	2024-07-31
License:	GPL-3

Currently implemented are visualisations dealing nominal data with multiple levels lmap ("levelMap") and fmap ("factorMap"), and some assistance for making of weighted geographic Voronoi maps wmap ("weigthedMap").

Author(s)

Michael Cysouw <cysouw@mac.com>

`addContour` *Add contourlines to a map*

Description

Convenience function to add contourlines to a map, specifically geared towards suggesting boundaries to the result of weightedMap. Internally based on a krige-interpolation.

Usage

```
addContour(heights, points, window, crs, add = TRUE,
           levels = c(0.4, 0.45, 0.5), grid = 50000, ...)
```

Arguments

heights	Numeric vector with the same length as points. Typically a 0/1 vector describing presence or absence of a features.
points	Locations of the datapoints as sfc_POINTS or an sf object with such a geometry.
window	Window for the interpolation as sfc_POLYGON or an sf object with such a geometry.
crs	A crs in WKT format.
add	By default, contourlines are added to the previous plot. Otherwise, they are returns as sfc_LINESTRINGS.
levels	Levels on which to draw the contourlines. Multiple lines get thicker towards higher values to suggest a center. These levels have to be related relative to the heights.
grid	Number of points inside the window for the krige-interpolation. Higher numbers lead to nicers contourlines, but take longer to evaluate.
...	Additional specifications passed internally to contour.

Details

Internally, a grid is made inside the window and the height is interpolated using ordinary kriging from [gstat]{krige} with a model suggested by autofitVariogram.

Value

Contourlines are added to the current plot or (when add=FALSE) they are returned as an sf object with a sfc_LINESTRING geometry.

Note

This is a preliminary convenience function that will be used to overhaul levelMap

Author(s)

Michael Cysouw <cysouw@mac.com>

See Also

weightedMap for more involved example

Examples

```
data(hessen)

# continuous variable between 0 and 1
data <- hessen$data[,1:3]
heights <- round(data[,1]/rowSums(data), digits = 1)
cols <- heat.colors(11)
names(cols) <- names(table(heights))
```

```
# boundary as sf
w <- sf::st_as_sf(hessen$boundary)
sf::st_crs(w) <- 4326
w <- sf::st_transform(w, 2397)

# points as sf
p <- sf::st_as_sf(hessen$villages, coords = c("longitude", "latitude"))
sf::st_crs(p) <- 4326
p <- sf::st_transform(p, 2397)

# plot map
plot(st_geometry(p), col = cols[as.character(heights)], pch = 19)
plot(st_geometry(w), add = TRUE, border = "grey")

# add boundary
addContour(heights, points = p, window = w, crs = 2397, grid = 1000,
            levels = c(0.25, 0.35, 0.45, 0.55), col = "blue")
```

`boundary` *Checking boundary parameters for plotting of* `levelMap`

Description

The function `levelMap` can be tweaked by various parameters determining the boundary of the interpolation. The function `boundary` helps finding suitable parameters.

Usage

```
boundary(points, density = 0.02, grid = 10, box.offset = 0.1
  , tightness = "auto", manual = NULL, plot = TRUE)
```

Arguments

`points`	Points, typically a two-column matrix with x and y coordinates.
`density`	Density of points below which there should be no interpolation.
`grid`	Density of the grid.
`box.offset`	Distance of the box around the points.
`tightness`	Parameter influencing how tightly the boundary should be wrapped around the points. Passed internally to kde2d. When `"auto"` this defaults to bandwidth.nrd. Lower values will result in tighter boundaries.
`manual`	Manually added boundary points in the form of a two-column matrix with coordinates.
`plot`	Logical: by default the impact of the chosen parameters is shown. If `FALSE` then coordinates are returned that are the outside of the boundary.

Details

Instead of trying to use a polygon as a boundary for the interpolation internally in `levelMap` it turned out to be easier to use a collection of points that mark the outside.

Value

By default, returns a plot with the original points in black, the points below density in red, and the box around the points in blue. Contour lines of the density are shown to choose different density parameters.

When `plot = FALSE`, the blue and red points from the graphic are returned as a two-column matrix of x and y coordinates.

Author(s)

Michael Cysouw <cysouw@mac.com>

See Also

Used internally in `levelMap`. The parameters of this function can be passed through, typically `density` and `box.offset`.

Examples

```
data(hessen)

# show impact of the chosen parameters
boundary(hessen$villages, density = 0.1, grid = 20
  , manual = cbind(x = c(8.3, 9.2), y = c(49.9, 50.0)))

# return coordinates
boundary(hessen$villages, plot = FALSE)

# abstract example, showing tightness in action
oldpar<-par("mfrow")
par(mfrow = c(1,3))

p <- cbind(c(1:10, 1:10), c(1:10, 10:1))
boundary(p, density = 0.005, grid = 20, tightness = "auto")
boundary(p, density = 0.005, grid = 20, tightness = 5)
boundary(p, density = 0.005, grid = 20, tightness = 3)

par(mfrow = oldpar)
```

dialects *Multiple correspondences of "f"-like sounds in German Dialects*

Description

In total 34 different words in which an f-like sound occurs. The different pronunciations of this sound in 183 different German villages are included in this dataset.

Usage

```
data(dialects)
```

Format

List of 2:

`villages` Dataframe with two variables LONGITUDE and LATITUDE for all 183 villages.

`data` Matrix with 34 columns showing the pronunciation in the 183 villages.

Source

Excerpt from https://github.com/cysouw/PAD/

Examples

```
## Not run:
# might give error message because of non-ASCII phonetic symbols
data(dialects)
require(mapdata)
map("worldHires", "Germany", fill = TRUE, col = "grey90")

lmap(dialects$villages, dialects$data[,21]
  , levels = c(0.20, 0.22, 0.24), add = TRUE, position = "topleft")

title(main = "f-sound in \'Kochlöffel\'")

## End(Not run)
```

factorMap *Visualising nominal data with various factors.*

Description

A factor map ("fmap") is a counterpart of the base function `image`. In contrast to an image, a factor map can be used for nominal data with various levels (instead of continuous numerical data). A matrix (or a dataframe coerced as matrix) is visualised by showing the most frequent contents of the cells by colouring. There are various methods for ordering of rows and columns provided, alike to a `heatmap`).

Usage

```
factorMap(x, order = NULL, col = rainbow(4), show.remaining = FALSE,
  col.remaining = "grey", pch.na = 20, col.na = "lightgrey", legend = length(col),
  labels.x = rownames(x), labels.y = colnames(x), cex.axis = 1, cex.legend = 1,
  cex.remaining = 1, font = "", asp = nrow(x)/ncol(x), method = "hamming",
  control = NULL, plot = TRUE)
```

Arguments

x	A matrix or dataframe with the data to be displayed. Rows are shown on the x-axis, columns on the y-axis, showing the row- and column-names in the display. All data in the whole matrix is interpreted as one large factor with different levels.
order	How should rows and columns be ordered? By default the order of the data matrix x is used. Many possible algorithmic orderings are available, see Details. custom orderings should simply be applied to the x matrix beforehand, then without invoking this order argument.
col	Colors to be used for the display. By default, the colours specified here are used in order of frequency of the phenomena in the data (i.e. order(table(x), decreasing = TRUE). A named vector of colors (or a named list) are applied to the levels as named. All other levels are shown as 'others'. Optionally use show.remaining to show labels for these others in the visualisation.
show.remaining	Logical: should all levels without color be shown inside the boxes as text?
col.remaining	Which color should the text of the uncolored levels have?
pch.na	Symbol to be used for NA elements. Use NULL for no symbol, but complete coloring of the boxes.
col.na	Color to be used for NA elements.
legend	How many levels should be shown in the legend (in the order of frequency? Alternatively, provide a vector with names of the levels to be shown. Use NULL to suppress the legend.
labels.x	Labels to be used on the x-axis. Defaults to rownames of the data. Use NULL to suppress labels.
labels.y	Labels to be used on the y-axis. Defaults to colnames of the data. Use NULL to suppress labels.
cex.axis	Size of the row and columns names of x, shown as axis labels.
cex.legend	Size of the legend text.
cex.remaining	Size of the text in the boxes. Only shown when show.remaining = TRUE.
font	Font to be used in the plotting, can be necessary for unusual unicode symbols. Passed internally to par(family).
asp	Aspect-ratio of the plotting of the boxes. By default the complete plot will be approximately square. Use the value 1 for all square boxes. Manually resizing the boxes by changing the plotting window can be achieved by setting asp = NA.
method	Method used to determine similarity, passed to sim.obs, which is used internally to determine the order of rows and columns, using the method chosen in order.

control	List of options passed to seriate.
plot	By default, a plot is returned. When FALSE, nothing is plotted, but the reordering is returned.

Details

There are many different orderings implemented: "pca" and "varimax" use the second dimension of prcomp and varimax respectively. "eig" will use the first eigenvector as computed by eigs. This is really quick for large datasets. "mds" will use the first dimension of cmdscale.

Further, all methods as provided in the function seriate can be called. Specifcally, "R2E" and "MDS_angle" seem worthwhile to try out. Any parameters for these methods can be passed using the option control.

Value

A plot is returned by default. When plot = FALSE, a list is returned with the reordering of the rows and the columns.

Note

Note that it is slightly confusing that the resulting image is a transposed version of the data matrix (rows of the matrix are shown as horizontal lines in the graphic, and they are shown from bottom to top). This is standard practice though, also used in image and heatmap, so it is continued here.

Author(s)

Michael Cysouw <cysouw@mac.com>

See Also

image in base and pimage in the package seriation.

Examples

```
# a simple data matrix
x <- matrix(letters[1:5],3,5)
x[2,3] <- x[1,4] <- NA
rownames(x) <- c("one", "two", "three")
colnames(x) <- 1:5
x

# some basic factor maps
factorMap(x, asp = 1)
factorMap(x, col = heat.colors(5), asp = NA)
factorMap(x, col = list(b = "red", e = "blue"), show.remaining = TRUE)

## Not run:
# more interesting example, different "f" sounds in german dialects
# note that fonts might be problematic on some platforms
# plotting window should be made really large as well
data(dialects)
```

```
factorMap(dialects$data, col = rainbow(8), order = "R2E"
    , cex.axis = 0.3, cex.legend = 0.7
    , show.remaining = TRUE, cex.remaining = 0.2)

# get reordering of rows
# to identify the group of words with "p-f" correspondences
factorMap(dialects$data, order = "R2E", plot = FALSE)

## End(Not run)
```

`haspelmath` *Data about indefinite constructions in 39 different languages*

Description

Summary of the data in Haspelmath (1997), classical example of the usage of semantic maps.

Usage

```
data(haspelmath)
```

Format

The dataset is a matrix with nine rows, describing the different indefinite functions, and 134 columns documenting for each individual construction which functions are possibly expressed by it.

Source

Haspelmath, Martin. Indefinite Pronouns. Oxford Studies in Typology and Linguistic Theory. Oxford: Clarendon, 1997.

Examples

```
data(haspelmath)
## English data
haspelmath[,7:9]
```

`heeringa` *Heeringa-style colours*

Description

Proposed in Heeringa (2004) to colour a (dis)similarity by decomposing it into three dimensions (using `cmdscale` here) and then mapping these dimensions to RGB to make colours. Highly useful to visualize pairwise similarities between geographic regions.

Usage

```
heeringa(dist, power = 0.5, mapping = c(1, 2, 3), method = "eigs", center = NULL)
```

Arguments

dist	dist object specifying distances between points.
power	Factor used to influence the results of the multidimensional scaling. Values closer to one will lead to clearer separated colours, while higher values will lead to more gradual colours.
mapping	Optional vector to change the mapping of the dimensions to the colours. Should be of length 3, specifying to which color each of the three dimensions is mapped. A 1 means 'red', a 2 means 'green' and a 3 means 'blue'. Adding a minus reverses the mapping.
method	Method used to determine the colour dimensions. Either mds (nicer colourbalance) or eigs (much faster).
center	Optionally, specify an index of one of the points to be put in the center of the coloring scheme, i.e. this point will become grey and all other points will be colored relative to this point.

Details

This proposal goes back to Heeringa (2004). The idea is to visualize distances by mapping the first three dimensions of a multidimensional scaling to the the red-green-blue scales. The mapping vector can be used to change the mapping to the colours.

Value

A vector of colours of the same length as the size of the dist object.

Author(s)

Michael Cysouw <cysouw@mac.com>

References

Heeringa, Wilbert. "Measuring Dialect Pronunciation Differences Using Levenshtein Distance." Ph.D. Thesis, Rijksuniversiteit Groningen, 2004.

Examples

```
data(hessen)
tess <- weightedMap(hessen$villages, window = hessen$boundary, crs = 2397)
d <- dist(hessen$data, method = "canberra")

# different mappings of the colors
c1 <- heeringa(d)
plot(tess$weightedVoronoi, col = c1, border = NA)

c2 <- heeringa(d, power = 1, mapping = c(3, -2, 1))
```

```
plot(tess$weightedVoronoi, col = c2, border = NA)
```

`hessen` *Extract from the SyHD Project on the syntax of the dialect of Hessen (Germany)*

Description

An example dataset of dialect data.

Usage

```
data(hessen)
```

Format

List of 3

`boundary` An object of type `owin` describing a geographical boundary. This format is necessary for Voronoi diagrams.

`villages` A dataframe with two variables "longitude" and "latitude" for the 157 villages on this there is data in this dataset.

`data` Dataframe with 56 different characteristics of these 157 villages, distributed over 15 different variables (as indicated in the column names).

Source

Data from https://www.syhd.info/startseite/index.html

References

Jürg Fleischer, Simon Kasper & Alexandra N. Lenz (2012): Die Erhebung syntaktischer Phänomene durch die indirekte Methode: Ergebnisse und Erfahrungen aus dem Forschungsprojekt "Syntax hessischer Dialekte" (SyHD). In: Zeitschrift für Dialektologie und Linguistik 79/1, 2-42.

Examples

```
data(hessen)

tessalation <- weightedMap(hessen$villages, window = hessen$boundary, crs = 2397)
plot(tessalation$weightedVoronoi)
```

levelMap *Drawing multi-level maps (e.g. semantic maps or linguistic isoglosses)*

Description

A multi-level map ("lmap") is a plot of the distribution of nominal data with multiple levels in space. Such visualisations have two direct use-case in linguistics, viz. semantic maps and isoglosses. The drawing if the lines in space is performed by interpolation in this function (see details).

Semantic maps (Haspelmath 2003) are a visualisation of linguistic diversity. A semantic map shows a predefined configuration of functions/senses in two-dimensional space with an overlay of language-specific encoding of these functions/senses. An level-map tries to emulate this linguistic visualisation in an automatic fashion with various options for visual presentation.

Isoglosses show lines surrounding similar phenomena in space. Instead of drawing an exact boundary around measured points, an interpolation-technique is used here to show areas of interest. By only showing boundaries, multiple phenomena can be shown in one graphic.

Usage

```
levelMap(points, data,
   main = NULL, draw = 5, levels = c(0.41, 0.46, 0.51),
   labels = NULL, cex = 0.7, col = "rainbow", add = FALSE,
   ignore.others = FALSE, normalize.frequency = FALSE, scale.pies = FALSE,
   lambda = NA, legend = TRUE, position = "bottomleft", cex.legend = 0.7,
   font = "", note = TRUE, file.out = NULL, ...)
```

Arguments

points	Coordinates of the data points specified as a two-column matrix or dataframe.
data	Language data to be plotted as contour-overlay over the points. Either specified as a vector of language-specific forms, or as a numeric matrix with the forms as columns and the points as rows (the language-specific forms should be specified as colnames). The values in the matrix designate the occurrence of the forms, allowing for the encoding of frequency/typicality and of overlap of different forms being used in the same function. see Details.
main	Title for the plot
draw	Which forms to be drawn by contours. Specifying a numeric value will only draw the uppermost frequent forms in the data, by default only the topmost five forms are drawn (automatically ordered by frequency). Alternatively, a vector with names or column-indices of the forms to be drawn can be specified.
levels	height of contours to be drawn. Internally, all values are normalized between zero and one, so only values between those extremes are sensible. Line thickness is automatically balanced.

labels	Optionally, character vector with labels for the points, to be drawn instead of symbols in the plot. Should be a vector of the same length as the number of points. Alternatively, a single character-string is repeated for all points.
cex	Character expansion of the labels (see previous option). Also influences the size of symbols or pie-charts.
col	Colour specification, either in the form of the name of a built-in color palettes, like rainbow, or a manually specified vector of colors. When NULL, an attempt is made to use grey-scales.
add	Logical: should the plot be added to an existing plot or not?
ignore.others	Logical: ignore all other categories, not selected through draw.
normalize.frequency	Logical: should rows of data (points in the plot) be normalized to 1? Useful only for data that represent frequency of occurence as columns. Note that setting this to FALSE influences the behaviour of levels.
scale.pies	logical: for multivalued data: should the size of the pies represent frequencies (TRUE) or all be of the same size (FALSE, by default)?
lambda	Parameter for the interpolation, passed internally to the function Krig. Low values result in more detailed boundaries around the measured points.
legend	Logical: should a legend be added?
position	Where should the legend be positioned? Passed internally to legend.
cex.legend	Character expansion passed to legend, and also used for the indication of the levels in the plot
font	Font to be used for the legend and the labels. Passed internally to par(family).
note	Logical: should a note be added to the bottom of the graphic to document the levels of the countour lines?
file.out	Location for writing the image to a file instead of plotting it on screen
...	Additional parameters optionally passed to boundary for the specification of the area of interpolation.

Details

The basic idea is to use some kind of interpolation to show areas of high-occurrence of a specific phenomenon. Internally Kriging is used, and then only contour lines are shown of the interpolation. Multiple lines are suggested to indicate the probibalistic interpretation of the lines.

Value

A plot is produces with the different phenomena in space surrounded by lines. When multiple options are possible at each point then pie charts are added.

Author(s)

Michael Cysouw <cysouw@mac.com>

Examples

```
# isogloss example
# choose one feature from hessen dataset (number 4)
data(hessen)
f4 <- hessen$data[,9:13]

# look for area for interpolation, changing density and grid parameters
# suitable parameters can be passed through to function levelMap below
boundary(hessen$villages, density = 0.1, grid = 10)

# useful size of pies has to be determined by changing cex
plot(hessen$boundary, main = NULL)
levelMap(hessen$villages, f4, draw = 3, cex = 0.8, normalize.frequency = TRUE
  , density = 0.1, grid = 10, add = TRUE, cex.legend = 0.5, scale.pies = TRUE)

## Not run:
# another isogloss example:
# "f" sounds in German dialects in the words "Kochlöffel"
# might give Unicode-errors because of phonetic symbols
require(mapdata)
map("worldHires", "Germany", fill = TRUE, col = "grey90")

data(dialects)
levelMap(dialects$villages, dialects$data[,21], levels = c(0.20, 0.22, 0.24)
    , add = TRUE, position = "topleft")
title(main = "f-sound in \'Kochlöffel\'")

## End(Not run)

# semantic map example
# location of points via multidimensional scaling of complete data
data(haspelmath)
d <- dist(haspelmath)
p <- MASS::isoMDS(d)$points

# testing boundary parameters
boundary(p)
boundary(p, density = 0.004, box = 0.15, tightness = 8)

# labels to be plotted instead of points
text <- gsub("\\.", "\n", rownames(haspelmath))

# show a few languages for Haspelmaths indefinite data
# using a quick dummy function to set all parameters
indef <- function(columns) {
  levelMap(p, haspelmath[,columns]
    , levels = 0.1, labels = text
    , density = 0.004, box = 0.15, tightness = 8
    , lambda = 0.1, note = FALSE)
}

oldpar <- par("mfcol")
```

```
par(mfcol = c(2,3))

indef(1:3)
indef(4:6)
indef(7:9)
indef(10:12)
indef(13:17)
indef(18:22)

par(mfcol = oldpar)
```

---

vmap *Plotting a Voronoi-map ("v-map")*

Description

These functions are deprecated: use weightedMap instead.

A Voronoi-map (voronoi-tessellation, also known as dirichlet tessellation) is used in quantitative dialectology. This function is a convenience wrapper to easily produce dialect maps with voronoi tessellations. Also described here are a helper functions to produce the tessellation.

Usage

```
vmap(tessellation, col = NULL, add = FALSE,
  outer.border = "black", border = "grey", lwd = 1, ...)

voronoi(points, window)
```

Arguments

tessellation	Tessellation of class tess from the library spatstat.geom. Can easily be produces by using the convenience function voronoi provided here.
col	Vector of colors for the filling of the tessellation. Is recycled when there are more tiles than colours. The order of the tiles is the same as the order of the points as specified in the function voronoi.
add	Add graphics to an existing plot
outer.border	Colour of the outer border. Specifying NA removes the border.
border	Colour of the inner borders. Specifying NA removes all borders.
lwd	Line width of borders.
...	Further arguments passed to polygon.
points	Two-column matrix with all coordinates of the points to make a Voronoi tessellation.
window	Outer boundary for the Voronoi tessellation. Should be in the form of an owin object. There are two helper functions provided here to get such object. Note that the function voronoi will give warnings if there are points outside of this window.

Details

This code is almost completely based on functions from the spatstat.geom package. For convenience, first some geographical boundaries can easily be accessed and converted for use in spatstat.geom. Then a Voronoi tessellation can be made (based on the function dirichlet, which in turn is based on deldir from the package deldir). Finally, this tessellation can be plotted filled with different colours.

Any legends have to be added manually by using legend, see examples below.

The function voronoi returns a warning when points are attested that lie outside of the specified border. For these points there is no polygon specified. Indices for the rejected points outside the border can be accessed by attr(x, "rejects").

Value

voronoi returns a tessellation of the class tess from the package spatstat.geom. When points outside of the border are attested, the indices of these points are added to an attribute "rejects". vmap plots a map.

Author(s)

Michael Cysouw <cysouw@mac.com>

Examples

```
## Not run:
# make a Voronoi tessellation for some villages in hessen
data(hessen)
plot(hessen$boundary)
points(hessen$villages, cex = 0.3)

tessellation <- voronoi(hessen$villages, hessen$boundary)
plot(tessellation)

# make a resizable plot with random colour specification
vmap(tessellation, col = rainbow(5), border = NA)
legend("bottomright", legend = c("a","b","c","d","e"), fill = rainbow(5))

# use actual colors from data, using first feature from supplied data
# multiple levels cannot easily be shown
# consider \link{lmap} for more detail
d1 <- hessen$data[,1:3]
d1 <- d1[,1]/rowSums(d1)
vmap(tessellation, col = rgb(1, 1-d1, 1-d1))
text(hessen$villages,labels=hessen$data[,1],cex=.5)
legend("bottomright", legend = c("es mir", "mir es / other"),
  fill = c("red", "white"))

# Use distances to determine colour, as proposed by Heeringa (2004)
# Note that different methods to establish distances can lead to rather
# different results! Also try method = "euclidean"
d <- dist(hessen$data, method = "canberra")
```

```
cols <- heeringa(d)
vmap(tessellation, col = cols, border = NA)

## End(Not run)
```

`weightedMap` *Construct weighted map using Voronoi tessellation and cartogram weighting*

Description

A weighted map ("wmap") is a combination of a Voronoi tessellation with cartogram weighting. A Voronoi map is a tessellation of a surface based on a set of geographic points. It is used to display areal patterns without overlap. Additionally, the size of the tiles can be weighted by cartogram-deformation to allow for varying the visual impression of the data. Specifically, this allows for equal-area-sized tiles to equally represent all data-points in the visual display.

Usage

```
weightedMap(x, y = NULL, window = NULL, crs = NULL, weights = "equal",
  grouping = NULL, holes = NULL, concavity = 2, expansion = 1000,
  method = "dcn", maxit = 10, verbose = 0)
```

Arguments

`x`	Coordinates of the data-points, either an `sf` object or a two-column matrix/dataframe with x ('longitude') and y ('latitude') coordinates. Alternatively, only specify the x-coordinates here and use the y parameter for the y-coordinates.
`y`	Latitude (y-coordinates), when the parameter `x` is used for longitude only.
`window`	Geographical window within which the Voronoi-tessellation will be displayed. Typically an `sf` (multi)polygon, but an attempt is made to interpret other formats (e.g. `owin` from Spatstat, `SpatVector` from Terra and `Spatial` from sp). Consider libraries like `geodata` and `rnaturalearth` to obtain suitable windows. Polygons that do not contain any coordinates are removed (with a warning). Coordinates that do not lie within the window are removed (with a warning). When no window is provided (by default), then a concave hull is induced from the coordinates (using `concaveman`). Various other parameters explained below can be used to influence this hull.
`crs`	Coordinate reference system that is necessary for the projection of the map. When not provided, an attempt is made to extract a crs from the provided coordinates or from the provided window. Without any crs there will be a warning and `EPSG:3857` will be assumed. Note that the ubiquitous `EPSG:4326` is strictly speaking not a projection and results in various errors; use a projected version like `EPSG:3857` instead, or use any of the numerous better alternatives (see examples below for some ideas).

`weights` Vector with weights for the deformation of the Voronoi-tiles. Should have the same length as the number of coordinates provided. The weights are passed to `cartogramR` to perform the deformation. Defaults to `"equal"` for equal-area tiles. When `NULL` no weighting is performed, but a non-weighted Voronoi-map is still produced.

`grouping` Influence the form of the concave window around the coordinates. Only used when there is no `window` provided. A vector with the same length as the number of coordinates, listing for each coordinate to which group it belongs. An attempt is made to make separate windows for each group. Note that a high value for the parameter `expansion` might result in overlap.

`holes` A list of `x,y` coordinates where holes should be inferred. When `window = NULL` there will be holes inserted inside the window around the coordinates specified within a distance as specified in `expansion` from the nearest points around the coordinates. Not used when there is an explicit window provided.

`concavity` Influence the form of the concave window around the coordinates. Only used when there is no `window` provided. Parameter passed internally to `concaveman` determining the concavity of the hull. High values result in more convex hulls. Low values (especially between 1 and 0) lead to highly concave ("wiggly") windows.

`expansion` Influence the form of the concave window around the coordinates. Only used when there is no `window` provided. Expands the window (value in meters), and results in more "rounded" windows.

`method` Method used for cartogram deformation, passed to `cartogramR`. By default, the older-quicker-less accurate method `dcn` is used. More modern and accurate methods `gsm` and `gn` can also be used, but they might lead to strange results with more complex windows. They also sometimes lead to strange results, or downright stall. Lower `maxit` might prevent these problems, but lead to less accurate weighting. Also consider the option `verbose` to get an indication where things go wring.

`maxit` Maximum number of iterations to find a suitable deformation. Parameter passed internally to `cartogramR`. Higher values lead to better approximations of the size of the polygons to the weights. However, with complex maps it might take very long to converge (or even never finish).

`verbose` With `verbose = 1` turn on verbose output for `cartogramR` to check where the deformation might run wild

Details

Internally, the Voronoi-tessellation is made without respecting the window, and only afterwards the window is superimposed on the tessellation. In some circumstances with internal holes in the window provided, an attempt is made to return tiles that do not jump across such holes. However, sometimes artefacts are still visible in the output.

Warnings are produced when coordinates lie outside the window provided. The results should still work, but without these points outside. Any colouring or other uses of the results have to be adapted accordingly by using the information in `$outsideWindow`. Polygons without any points inside are likewise removed with a warning.

To deal with overlapping coordinates some jitter is automatically applied to the coordinates provided.

Value

List of various lengths, depending on specified parameters. Use the names to select any of these results:

`crs:`	The crs in WKT format.
`points:`	The coordinates as provided, but as a projected `sfc_POINT` object.
`grouping:`	Character vector with the provided grouping, or the grouping as induced from any provided window.
`window:`	The window around the coordinates as a projected `sfc_POLYGON` or `sfc_MULTIPOLYGON` object. Some polygons from a provided window might have been removed because they are empty.
`emptyPolygons:`	Numeric vector with the indices of the polygons that are removed because they do not contain any of the coordinates provided.
`outsideWindow:`	Numeric vector with the indices of the coordinates that are removed because they are outside of the window provided.
`voronoi:`	Voronoi-tessellation of the window as a projected `sfc_MULTIPOLYGON` object.
`weights:`	Numeric vector with the weights used for the deformation. Weights for coordinates outside of the window are removed.
`weightedPoints:`	Coordinates after the weighting-deformation, specified as a projected `sfc_POINT` object.
`weightedWindow:`	Window after the weighting-deformation, specified as a projected `sfc_POLYGON` or `sfc_MULTIPOLYGON` object.
`weightedMap:`	Voronoi-tessellation after the weighting-deformation, specified as a projected `sfc_MULTIPOLYGON` object.

Note

With more complex windows the deformation by `cartogramR` might throw errors (like "IllegalArgumentException") or fall into an infinite loop. Try to reduce `maxit` and check `verbose = 1` to get an idea what might be going wrong.

Author(s)

Michael Cysouw <cysouw@mac.com>

Examples

```
# generate a window from coordinates
# note the Germany-centered Gauss-Kruger projection "EPSG:2397"
# consider increasing 'maxit' to remove the warning about convergence
data(hessen)
```

```
v <- weightedMap(hessen$villages, expansion = 4000, crs = 2397, maxit = 2)
plot(v$weightedVoronoi)

# show the original locations before the transformation in orange
plot(v$points, add = TRUE, col = "green", cex = .5)
# show the new locations after transformation in red
plot(v$weightedPoints, add = TRUE, col = "blue", cex = .5)

# add the real border of Hessen for comparison
h <- sf::st_as_sf(hessen$boundary)
sf::st_crs(h) <- 4326
h <- sf::st_transform(h, 2397)
plot(h, add = TRUE, border = "red")

# use the Voronoi tiles e.g. for Heeringa-colouring (see function "heeringa()")
d <- dist(hessen$data, method = "canberra")
plot(v$weightedVoronoi, col = heeringa(d), border = NA)
plot(v$weightedWindow, add = TRUE, lwd = 2)

# grouping-vector can be used to make separations in the base-map
groups <- rep("a", times = 157)
groups[157] <- "b"
groups[c(58,59)] <- "c"
groups[c(101, 102, 107)] <- "d"
# holes-list can be used to add holes inside the region
holes <- list(c(9, 50.5), c(9.6, 51.3), c(8.9, 51))
v <- weightedMap(hessen$villages, grouping = groups, holes = holes,
                 crs = 2397, expansion = 3000)
plot(v$weightedVoronoi, col = "grey")

## Not run:
# extensive example using data from WALS (https://wals.info). Both the worldmap
# and the WALS data are downloaded directly. The worldmap is projected and
# deformed so that each datapoint has equal area on the map.

# load worldmap
data(world)

# try different projections
azimuth_equaldist  <- "+proj=aeqd +lat_0=90 +lon_0=45"
mollweide_atlantic <- "+proj=moll +lon_0=11.5"
mollweide_pacific  <- "+proj=moll +lon_0=151"

plot(sf::st_transform(world, crs = azimuth_equaldist))

# load WALS data, example feature 13: "tone"
library(lingtypology)
feature <- "13A"
wals <- wals.feature(feature)
head(wals)

# get glottolog coordinates and correct errors in WALS
wals$glottocode[wals$glottocode == "poqo1257"] <- "poqo1253"
```

```
wals$glottocode[wals$glottocode == "mamc1234"] <- "mamm1241"
wals$glottocode[wals$glottocode == "tuka1247"] <- "tuka1248"
wals$glottocode[wals$glottocode == "bali1280"] <- "unea1237"
wals <- merge(wals[,c("glottocode", "wals.code", feature)],
              glottolog[,c("glottocode", "longitude", "latitude")])

# calculate an equally-weighted voronoi transformation
v <- weightedMap(wals$longitude, wals$latitude, window = world,
                 crs = mollweide_atlantic, method = "dcn", maxit = 10)

# prepare colors
cols <- c("lightsalmon", "grey", "lightpink")
names(cols) <- names(table(wals[,feature]))
cols <- cols[c(2,3,1)]

# the map
plot(v$weightedVoronoi, col = cols[wals[,feature]], border = "darkgrey", lwd = 0.2)
plot(v$weightedWindow, border = "black", add = TRUE, lwd = 0.5)
legend("bottomleft", legend = names(cols), fill = cols, cex = .7)

# add contourlines
height <- c(0, 0.5, 1)
names(height) <- c("No tones", "Simple tone system", "Complex tone system")
addContour(height = height[wals[,feature]],
           points = v$weightedPoints,
           window = v$weightedWindow,
           crs = v$crs,
           col = "darkred", levels = c(0.2, 0.4, 0.6))

# Alternative: using points instead of polygons
cols[2:3] <- c("orange", "red")
plot(v$weightedPoints, col = cols[wals[,feature]], cex = 1, pch = 19)
plot(v$weightedWindow, add = T, border = "darkgrey", lwd = 0.5)

## End(Not run)
```

`window` *Producing windows of class "owin"*

Description

These functions are deprecated: use weightedMap instead.

Different ways to easily produce windows of class "owin" from the package "spatstat" are presented here. These are used by voronoi.

Usage

```
hullToOwin(points, shift, alpha)
mapsToOwin(country, database = "world")
gadmToOwin(country, sub = NULL, level = 0)
```

Arguments

points	Set of points that need a window around them. Two column matrix.
shift	The amount of space around the outer points at determining the window.
alpha	Parameter for the 'curviness': lower values show more detail. Passed internally to ahull.
country	Name of the country to obtain borders and turn them into an owin object needed for the function voronoi. For mapsToOwin check map how to specify the names. For gadmToOwin, check https://gadm.org.
database	Database as used by map.
sub, level	Names for Subdivisions of countries as available in the GADM database

Details

For hullToOwin, the function ahull is used to make a hull around the points. This is then converted to an "owin" window.

The functions mapsToOwin and GadmToWin use external topogaphic boundaries to produce windows.

Value

All functions return an object of class owin from the package spatstat.

Note

Includes code from code from Andrew Bevan, based on code from Dylan Beaudette, see https://stat.ethz.ch/pipermail/r-sig-geo/2012-March/014409.html.

The function gadmToOwin needs online access to download the data. The data is saved in the current working directory, and will not be downloaded again when it is already available there.

Author(s)

Michael Cysouw <cysouw@mac.com>

Examples

```
## Not run:
# Boundary of the German state "Hessen"
# This will need to access the online GADM database
# and might take some time
boundary <- gadmToOwin("DEU", "Hessen", 1)

# A window does not have to be continuous
random <- mapsToOwin(c("Germany", "Greece"))
plot(random, main = NULL)

# hull around some points
# note influence of alpha and shift
data(hessen)
```

```
hull <- hullToOwin(hessen$villages, shift = 0.2, alpha = 1)
plot(hull)
points(hessen$villages)

hull <- hullToOwin(hessen$villages, shift = 0.1, alpha = 0.2)
plot(hull)
points(hessen$villages)

## End(Not run)
```

---

world *Boundary of the World fitting all Glottolog languages*

Description

A polygon representing a worldmap tailored to be not too detailed, but still fitting all glottolog languages inside the polygons.

Usage

```
data("world")
```

Format

The format is a `sfc_POLYGON` of length 475 with an EPSG:4326 projection.

Details

Some trickery was needed to produce a lightweight polygon to represent the worldmap in reasonably accuracy without becoming too large and unwiedly. The polygons are such that all coordinates for languages as listed in the glottolog (version 5) are inside these polygons.

The map has a basic EPSG:4326 projection, so longitude-latitude coordinates can immediately be added to it. However, this does not look very nice, because the polygon from Eurasia wraps around. Consider more suitable projections, see examples. To allow for a nice pacific-centered projection Greenland has been clipped.

Source

Polygons are based on the data from https://www.naturalearthdata.com with adjustments. Glottolog coordinates to select and adjust the polygons are from https://glottolog.org.

Examples

```
data(world)
plot(world)

# use different projections
azimuth_equaldist <- "+proj=aeqd +lat_0=90 +lon_0=45"
mollweide_atlantic <- "+proj=moll +lon_0=11.5"
```

```
mollweide_pacific <- "+proj=moll +lon_0=151"

plot(sf::st_transform(world, crs = azimuth_equaldist))
```

Index