Package ‘rangeBuilder’

October 30, 2024

Type Package

Title Occurrence Filtering, Geographic Standardization and Generation of Species Range Polygons

Version 2.2

Imports alphahull (>= 2.5), stringi, sf, terra, pbapply, units, rnaturalearth, methods, Rcpp (>= 0.12.9)

Depends R (>= 3.5.0)

Description Provides tools for filtering occurrence records, generating alpha-hull-derived range polygons and mapping species distributions.

License GPL (>= 3)

URL `https://github.com/ptitle/rangeBuilder`

BugReports `https://github.com/ptitle/rangeBuilder/issues`

NeedsCompilation yes

LinkingTo Rcpp

LazyData true

Encoding UTF-8

RoxygenNote 7.3.2

ByteCompile true

Author Pascal Title [aut, cre] (<`https://orcid.org/0000-0002-6316-0736`>)

Maintainer Pascal Title <`ptitle@umich.edu`>

Repository CRAN

Date/Publication 2024-10-30 19:30:02 UTC

Contents

`addRasterLegend` *addRasterLegend*

Description

Adds a legend to an existing raster plot, with some additional manual control

Usage

```
addRasterLegend(
  r,
  direction,
  side,
  location = "right",
  nTicks = 2,
  adj = NULL,
  shortFrac = 0.02,
  longFrac = 0.3,
  axisOffset = 0,
  border = TRUE,
  ramp = "terrain",
  isInteger = "auto",
  ncolors = 64,
  breaks = NULL,
  minmax = NULL,
  locs = NULL,
  cex.axis = 0.8,
  labelDist = 0.7,
  digits = 2,
  bigmark = "",
  ...
)
```

Arguments

`r`	the rasterLayer object that has been plotted
`direction`	direction of color ramp. If omitted, then direction is automatically inferred, otherwise can be specified as `horizontal` or `vertical`.
`side`	side for tick marks, see `axis` documentation. Automatically inferred if omitted.
`location`	either a location name (see `Details`), or coordinates for the corners of the bar legend `c(xmin, xmax, ymin, ymax)`.
`nTicks`	number of tick marks, besides min and max.
`adj`	if location is top, left, bottom or right, use this argument to adjust the location of the legend, defined in percent of the figure width. See Details for additional information.
`shortFrac`	Percent of the plot width range that will be used as the short dimention of the legend. Only applies to preset location options.
`longFrac`	Percent of the plot width range that will be used as the long dimention of the legend. Only applies to preset location options.
`axisOffset`	distance from color bar for labels, as a percent of the plot width.
`border`	logical, should the color legend have a black border
`ramp`	either a vector of color names for defining the color ramp, or "terrain" (default raster behavior)
`isInteger`	If `auto`, automatically determines if raster is made up of integer values, otherwise `TRUE` or `FALSE`
`ncolors`	grain size of color ramp
`breaks`	If a custom set of color breaks were used in plotting the raster, pass those color breaks here. This overrides the minmax option.
`minmax`	min and max values from which the color ramp will be derived. If left as `NULL`, the min and max of the raster will be used.
`locs`	locations of tick marks, if `NULL` automatically placed
`cex.axis`	size of axis labels
`labelDist`	distance from axis to axis labels (passed to `mgp`)
`digits`	number of decimal places for labels
`bigmark`	character used to separate thousands and millions, passed to `format`
`...`	additional parameters to be passed to `axis`.

Details

A number of predefined locations exist in this function to make it easy to add a legend to a raster plot. Preset `locations` are: `topleft`, `topright`, `bottomleft`, `bottomright`, `left`, `right`, `top` and `bottom`. If more fine-tuned control is desired, then a numeric vector of length 4 can be supplied to `location`, specifying the min x, max x, min y and max y values for the legend. Additionally, the `adj` argument can be used to more intuitively adjust where the legend is placed. `adj` is defined as a percentage of the figure width or height, left to right, or bottom to top, respectively. For example, if the legend is at the bottom, `adj = 0.8` will place the legend 80 the figure, horizontally centered. See examples.

Value

Invisibly returns a list with the following components.

- coords: 2-column matrix of xy coordinates for each color bin in the legend.
- width: Coordinates for the short dimension of the legend.
- pal: the color ramp
- tickLocs: the tick mark locations in plotting units
- labels: the values associated with those tick locations.

Author(s)

Pascal Title

`closestCountry` *Return country from point*

Description

Determines which country a given point falls in.

Usage

```
closestCountry(pt, crs = 4326)
```

Arguments

`pt`	longitude and latitude, as a numeric vector, 2-column table, or spatial points object.
`crs`	the CRS of the coordinate. If `pt` is a spatial object, this argument is ignored. The default 4326 indicates longlat unprojected.

Details

Based on a predetermined set of global points, this function finds the country of occurrence. This can be useful for checking the validity of a point by comparing the returned country to the country listed with the occurrence record. If a point falls close to the boundary between two countries, the names of the nearby countries are returned. This function will not be of much value if the point falls in the ocean, as it will return the country that is closest, regardless of how far away it is.

Value

If one point is provided, a character vector is returned. If multiple points are provided, a list of character vectors is returned.

Author(s)

Pascal Title

Examples

```
#point near a country border
closestCountry(c(-115.436, 32.657))

# testing different input options
samp <- sample(1:nrow(crotalus), 10)
xy <- crotalus[samp, c('decimallongitude', 'decimallatitude')]
sfpts <- sf::st_as_sf(xy, coords = c('decimallongitude', 'decimallatitude'), crs = 4326)
sfptsEA <- sf::st_transform(sfpts, crs = '+proj=eqearth')
spPts <- as(sfpts, 'Spatial')
closestCountry(xy)
closestCountry(sfpts)
closestCountry(sfptsEA)
closestCountry(spPts)
```

coordError *Coordinate error*

Description

Calculates the potential error in coordinates due to lack of coordinate precision.

Usage

```
coordError(coords, nthreads = 1)
```

Arguments

`coords`	longitude and latitude in decimal degrees, either as a long/lat vector, or as a 2-column table. Can be either as numeric or character format
`nthreads`	number of threads to use for parallelization of the function. The R package `parallel` must be loaded for `nthreads` > `1`.

Details

This function assumes that the true precision of the coordinates is equivalent to the greatest number of decimals in either the longitude or latitude that are not trailing zeroes. In other words:
`(-130.45670, 45.53000)` is interpreted as `(-130.4567, 45.5300)`
`(-130.20000, 45.50000)` is interpreted as `(-130.2, 45.5)`

If we use `(-130.45670, 45.53000)` as an example, these coordinates are interpreted as `(-130.4567, 45.5300)` and the greatest possible error is inferred as two endpoints: `(-130.45670, 45.53000)` and `(-130.45679, 45.53009)`

The distance between these two is then calculated and returned.

Value

Returns a vector of coordinate error in meters.

Author(s)

Pascal Title

Examples

```
data(crotalus)

xy <- crotalus[1:100, c('decimallongitude','decimallatitude')]

coordError(xy)
```

`filterByLand` *Filter occurrences based on land vs ocean*

Description

Identifies occurrence records that do not occur on land.

Usage

```
filterByLand(coords, crs = 4326)
```

Arguments

`coords` coordinates in the form of a 2 column numeric matrix, data.frame, numeric vector, or spatial points object (sf or sp). If spatial object, crs must be defined.

`crs` crs of input coords. Ignored if input coords are spatial object.

Details

This function uses a rasterized version of the GSHHG (global self-consistent, hierarchical, high-resolution geography database, `https://www.soest.hawaii.edu/pwessel/gshhg/`), that has been buffered by 2 km.

Value

returns a logical vector where `TRUE` means the point falls on land.

Author(s)

Pascal Title

Examples

```
data(crotalus)

#identify points that fall off land
filterByLand(crotalus[,c('decimallongitude','decimallatitude')])


# testing different input options
samp <- sample(1:nrow(crotalus), 10)
xy <- crotalus[samp, c('decimallongitude', 'decimallatitude')]
sfpts <- sf::st_as_sf(xy, coords = c('decimallongitude', 'decimallatitude'), crs = 4326)
sfptsEA <- sf::st_transform(sfpts, crs = '+proj=eqearth')
spPts <- as(sfpts, 'Spatial')
filterByLand(xy)
filterByLand(sfpts)
filterByLand(sfptsEA)
filterByLand(spPts)
```

`filterByProximity` *Filter by proximity*

Description

Filter occurrence records by their proximity to each other.

Usage

```
filterByProximity(xy, dist, returnIndex = FALSE)
```

Arguments

`xy`	longitude and latitude in decimal degrees, either as a matrix, dataframe, or spatial points object.
`dist`	minimum allowed distance in km
`returnIndex`	if `TRUE`, will return indices of points that would be dropped, if `FALSE`, returns the points that satisfy the distance filter.

Details

This function will discard coordinates that fall within a certain distance from other points.

Value

If `returnIndex` = `TRUE`, returns a numeric vector of indices. If `returnIndex` = `FALSE`, returns coordinates of the same class as the input.

Author(s)

Pascal Title

Examples

```
data(crotalus)

# within the first 100 points in the dataset, identify the set of points to
# drop in order to have points no closer to each other than 20 km

subset <- crotalus[1:100,]
tooClose <- filterByProximity(xy= subset[ ,c('decimallongitude','decimallatitude')],
dist=20, returnIndex = TRUE)

plot(subset[ ,c('decimallongitude','decimallatitude')], pch=1, col='blue', cex=1.5)
points(subset[tooClose, c('decimallongitude','decimallatitude')], pch=20, col='red')


# testing different input options
samp <- sample(1:nrow(crotalus), 100)
xy <- crotalus[samp, c('decimallongitude', 'decimallatitude')]
sfpts <- sf::st_as_sf(xy, coords = c('decimallongitude', 'decimallatitude'), crs = 4326)
sfptsEA <- sf::st_transform(sfpts, crs = '+proj=eqearth')
spPts <- as(sfpts, 'Spatial')
filterByProximity(xy, dist=20, returnIndex = TRUE)
filterByProximity(sfpts, dist=20, returnIndex = TRUE)
filterByProximity(sfptsEA, dist=20, returnIndex = TRUE)
filterByProximity(spPts, dist=20, returnIndex = TRUE)
```

flipSign *Flip sign of coordinates*

Description

Checks for coordinate sign mistakes by checking all possibilities against country occupancy.

Usage

```
flipSign(
  coordVec,
  country,
  returnMultiple = FALSE,
  filterByLand = TRUE,
  crs = 4326
)
```

Arguments

`coordVec` numeric vector of length 2: longitude, latitude

`country` the country that is associated with the record

`returnMultiple` if multiple sign flips lead to the correct country, return all options. If `FALSE`, returns the coords with the fewest needed sign flips.

`filterByLand` if `TRUE`, alternative coords will be tested for whether or not they fall on land.

`crs` the crs of the coordinate.

Details

This function generates all possible coordinates with different signs, and runs `closestCountry` on each, returning the coordinates that lead to a country match. It ignores coordinate options that do not pass `filterByLand`.

If a point falls close to the boundary between two countries, it is still considered a match.

Value

list with 2 elements

`matched` logical: Was the country matched

`newcoords` matrix of coordinates that were successful.

Author(s)

Pascal Title

Examples

```
#correct coordinates
flipSign(c(4.28, 39.98), country = 'Spain')

#mistake in coordinate sign
flipSign(c(115.436, 32.657), country = 'United States')

#incorrect sign on both long and lat, but not possible to distinguish for longitude
#except when we consider which alternative coords fall on land.
flipSign(c(-4.28, -39.98), country = 'Spain', filterByLand = FALSE, returnMultiple = TRUE)
flipSign(c(-4.28, -39.98), country = 'Spain', returnMultiple = TRUE)

#coordinates are incorrect
flipSign(c(4.28, 59.98), country = 'Spain')
```

getDynamicAlphaHull *Generate polygon based on alpha hulls*

Description

Generates an apha hull polygon, where the alpha parameter is determined by the spatial distribution of the coordinates.

Usage

```
getDynamicAlphaHull(
  x,
  fraction = 0.95,
  partCount = 3,
  buff = 10000,
  initialAlpha = 3,
  coordHeaders = c("Longitude", "Latitude"),
  clipToCoast = "terrestrial",
  alphaIncrement = 1,
  verbose = FALSE,
  alphaCap = 400
)
```

Arguments

`x`	dataframe of coordinates in decimal degrees, with a minimum of 3 rows.
`fraction`	the minimum fraction of occurrences that must be included in polygon.
`partCount`	the maximum number of disjunct polygons that are allowed.
`buff`	buffering distance in meters
`initialAlpha`	the starting value for alpha
`coordHeaders`	the column names for the longitude and latitude columns, respectively. If x has two columns, these are assumed to be longitude and latitude, and `coordHeaders` is ignored.
`clipToCoast`	Either "no" (no clipping), "terrestrial" (only terrestrial part of range is kept) or "aquatic" (only non-terrestrial part is clipped). See Details.
`alphaIncrement`	the amount to increase alpha with each iteration
`verbose`	prints the alpha value to the console, intended for debugging.
`alphaCap`	Max alpha value before function aborts and returns a minimum convex hull.

Details

From a set of coordinates, this function will create an alpha hull with `alpha = initialAlpha`, and will then increase `alpha` by `alphaIncrement` until both the `fraction` and `partCount` conditions are met.

If the conditions cannot be satisfied, then a minimum convex hull is returned.

If clipToCoast is set to "terrestrial" or "aquatic", the resulting polygon is clipped to the coastline, using a basemap from naturalearth. The first time this function is run, this basemap will be downloaded. Subsequent calls will use the downloaded map.

Value

a list with 2 elements:

hull	a sf polygon object
alpha	the alpha value that was found to satisfy the criteria. If a convex hull was returned, this will list MCH.

Author(s)

Pascal Title

See Also

Alpha hulls are created with ahull.

Examples

```
data(crotalus)

# create a polygon range for Crotalus atrox
x <- crotalus[which(crotalus$genSp == 'Crotalus_atrox'),]
x <- x[sample(1:nrow(x), 50),]

range <- getDynamicAlphaHull(x, coordHeaders=c('decimallongitude','decimallatitude'),
clipToCoast = 'no')

plot(range[[1]], col=transparentColor('dark green', 0.5), border = NA)
points(x[,c('decimallongitude','decimallatitude')], cex = 0.5, pch = 3)

# to add a basic coastline, you can use the internal map
# world <- rangeBuilder:::loadWorldMap()
# plot(world, add = TRUE, lwd = 0.5)
```

getExtentOfList *Get extent of list*

Description

Given a list of SpatialPolygons or sf objects, return a bounding box object that encompasses all items.

Usage

```
getExtentOfList(shapes)
```

Arguments

shapes	a list of SpatialPolygons or simple features

Value

An object of class bbox.

Author(s)

Pascal Title

Examples

```
data(crotalus)

# create some polygons, in this case convex hulls
sp <- split(crotalus, crotalus$genSp)
sp <- lapply(sp, function(x) x[,c('decimallongitude','decimallatitude')])
sp <- lapply(sp, function(x) x[chull(x),])
poly <- lapply(sp, function(x)
sf::st_convex_hull(sf::st_combine(sf::st_as_sf(x, coords = 1:2, crs = 4326))))

getExtentOfList(poly)
```

rangeBuilder *rangeBuilder*

Description

Provides tools for filtering occurrence records, standardizing countries names, generating alpha-hull-derived range polygons and mapping species distributions.

Author(s)

Pascal Title <ptitle@umich.edu>

References

Davis Rabosky, A.R., C.L. Cox, D.L. Rabosky, P.O. Title, I.A. Holmes, A. Feldman and J.A. McGuire. 2016. Coral snakes predict the evolution of mimicry across New World snakes. Nature Communications 7:11484.

See Also

Useful links:

- https://github.com/ptitle/rangeBuilder
- Report bugs at https://github.com/ptitle/rangeBuilder/issues

rangeBuilder-example *rangeBuilder datasets*

Description

Included datasets in rangeBuilder

Details

The crotalus dataset is the result of a query for genus Crotalus on the VertNet search portal (http://portal.vertnet.org/search), and has been thinned and lightly filtered, to serve as an example dataset for this package.

rasterStackFromPolyList *Polygon List to rasterStack*

Description

Takes a list of polygons and creates a multi-layer SpatRaster.

Usage

```
rasterStackFromPolyList(
  polyList,
  resolution = 50000,
  retainSmallRanges = TRUE,
  extent = "auto"
)
```

Arguments

polyList	a list of spatial polygon objects, named with taxon names. It is assumed that all items in last have same crs.
resolution	vertical and horizontal size of raster cell, in units of the polygons' projection
retainSmallRanges	boolean; should small ranged species be dropped or preserved. See details.
extent	if 'auto', then the maximal extent of the polygons will be used. If not auto, must be a numeric vector of length 4 with minLong, maxLong, minLat, maxLat.

Details

In the rasterization process, all cells for which the polygon covers the midpoint are considered as present and receive a value of 1. If retainSmallRanges = FALSE, then species whose ranges are so small that no cell registers as present will be dropped. If retainSmallRanges = TRUE, then the cells that the small polygon is found in will be considered as present.

Value

an object of class SpatRaster where all rasters contain values of either NA or 1.

Author(s)

Pascal Title

Examples

```
## Not run:
data(crotalus)
library(sf)
library(terra)

# get 10 species occurrence sets
uniqueSp <- split(crotalus, crotalus$genSp)
uniqueSp <- lapply(uniqueSp, function(x)
x[!duplicated(x[, c('decimallongitude', 'decimallatitude')]),])
uniqueSp <- names(uniqueSp[sapply(uniqueSp, nrow) > 5])
uniqueSp <- uniqueSp[1:10]

# create range polygons
ranges <- vector('list', length = length(uniqueSp))
for (i in 1:length(uniqueSp)) {
x <- crotalus[which(crotalus$genSp == uniqueSp[i]),]

ranges[[i]] <- getDynamicAlphaHull(x, coordHeaders = c('decimallongitude',
'decimallatitude'), clipToCoast = 'terrestrial')
}

# name the polygons
names(ranges) <- uniqueSp

# keep only the polygons
ranges <- lapply(ranges, function(x) x[[1]])

# Create a SpatRaster with the extent inferred from the polygons, and a cell
# resolution of 0.2 degrees.
# cells with the presence of a species get a value of 1, NA if absent.

rangeStack <- rasterStackFromPolyList(ranges, resolution = 0.2)

# calculate species richness per cell, where cell values are counts of species
richnessRaster <- app(rangeStack, fun=sum, na.rm = TRUE)
```

```
# set values of 0 to NA
richnessRaster[richnessRaster == 0] <- NA

#plot
ramp <- colorRampPalette(c('blue','yellow','red'))
plot(richnessRaster, col=ramp(100))

# to add a basic coastline, you can use the internal map
# world <- rangeBuilder:::loadWorldMap()
# plot(world, add = TRUE, lwd = 0.5)


## End(Not run)
```

`standardizeCountry` *Standardize country name*

Description

Standardizes country names to the list of countries used internally by this package.

Usage

```
standardizeCountry(country, fuzzyDist = 1, nthreads = 1, progressBar = TRUE)
```

Arguments

`country` character vector of country names or ISO codes

`fuzzyDist` for fuzzy searching, the maximum string distance allowed for a match; if 0, fuzzy searching is disabled.

`nthreads` number of threads to use for parallelization of the function. The R package `parallel` must be loaded for `nthreads` > `1`.

`progressBar` if `FALSE`, progress bar will be suppressed.

Details

This package interacts with data from the Global Invasive Species Database (GISD), the Reptile Database, as well as global maps that were used to generate the internal dataset used by `closestCountry`. Efforts have been made to make country names consistent across these separate datasets. This function can be used to convert the user's `Country` field to the same standardized set.

Fuzzy matching uses the function `adist`.

Parallelization with `nthreads` becomes more time-efficient only if the input vector is of multiple thousands of country names.

Value

Character vector of the standardized country names. If no match found, "" is returned.

Author(s)

Pascal Title

Examples

```
standardizeCountry(c("Russian Federation", "USA", "Plurinational State of Bolivia", "Brezil"))
```

transparentColor *Define colors with transparency*

Description

Converts a named color and opacity and returns the proper RGB code.

Usage

```
transparentColor(namedColor, alpha = 0.8)
```

Arguments

namedColor	a color name
alpha	a transparency value between 0 and 1, where 0 is fully transparent

Value

Returns the transparent color in RGB format.

Author(s)

Pascal Title

Index