Package ‘taRifx.geo’

February 20, 2015

Type Package
Title Collection of various spatial functions
Version 1.0.6
Date 2014-02-23
Author Ari B. Friedman
Maintainer Ari B. Friedman <abfriedman@gmail.com>
Description A collection of miscellaneous spatial object manipulation functions
License GPL (>= 2)
LazyLoad yes
Suggests gdata, fields, maps, geoR, maptools, shapefiles, spatial, mapproj, akima, mgcv, caTools, pspline
Imports taRifx, sp, rgdal, RJSONIO, rgeos, RCurl, methods
Collate 'Contributed.R' 'R-GIS.R' 'data.R'
NeedsCompilation no
Repository CRAN
Date/Publication 2014-05-31 08:13:41

R topics documented:

as.SpatialPolygons.bbox ......................................................... 2
cleanLatLon ................................................................. 3
clipToExtent ................................................................. 4
closestPoint ................................................................. 5
countPointsInPolys .......................................................... 5
cumDist ....................................................................... 6
geocode ...................................................................... 6
georoute ..................................................................... 8
gGeoCode ................................................................. 9
IDs ....................................................................... 9
interpolateAndApplyWithinSpatial ....................................... 10
interpolatePathpoints ...................................................... 11
as.SpatialPolygons.bbox

Convert a bounding box to a SpatialPolygons object Bounding box is first created (in lat/lon) then projected if specified

Description

Convert a bounding box to a SpatialPolygons object Bounding box is first created (in lat/lon) then projected if specified

Usage

as.SpatialPolygons.bbox(bbox,
proj4stringFrom = sp::CRS("+proj=longlat +datum=WGS84"),
proj4stringTo = NULL, interpolate = 0)

Arguments

bbox Bounding box: a 2x2 numerical matrix of lat/lon coordinates (rownames must be c('lat','lon') and colnames must be c('min','max'))
proj4stringFrom Projection string for the current bbox coordinates (defaults to lat/lon, WGS84)
proj4stringTo Projection string, or NULL to not project
interpolate If nonzero, the number of nodes per side to add in (helps maintain coverage if you're projecting)
**cleanLatLon**

**Value**

A SpatialPolygons object of the bounding box

**See Also**

`clipToExtent` which uses the output of this to clip to a bounding box

**Examples**

```r
bb <- matrix(c(3,2,5,4),nrow=2)
rownames(bb) <- c("lon","lat")
colnames(bb) <- c("min","max")
as.SpatialPolygons.bbox(bb)
```

---

**cleanLatLon**

**Description**

Standardize latitude/longitude coordinates Cleans up character representations of lat/lon coordinates

**Usage**

`cleanLatLon(vec)`

**Arguments**

vec  Character vector of lat/lon coordinates

**Value**

Numeric decimal lat/lon
clipToExtent

Restrict to extent of a polygon Currently does the sloppy thing and returns any object that has any area inside the extent polygon

Description

Restrict to extent of a polygon Currently does the sloppy thing and returns any object that has any area inside the extent polygon

Usage

clipToExtent(sp, extent)

Arguments

sp Spatial object
extent a SpatialPolygons object - any part of sp not within a polygon will be discarded

Value

A spatial object of the same type

See Also

as.SpatialPolygons.bbox to create a SP from a bbox

Examples

```r
set.seed(1)
require(rgdal)
require(sp)
P4S.latlon <- sp::CRS("+proj=longlat +datum=WGS84")
ply <- sp::SpatialPolygons(list(
    sp::Polygons(list(Polygon(cbind(c(2,4,4,1,2),c(2,3,5,4,2)))), "s1"),
    sp::Polygons(list(Polygon(cbind(c(5,4,2,5),c(2,3,2,2)))), "s2"))
    }, proj4string=P4S.latlon)
pnt <- sp::SpatialPoints( matrix(rnorm(100),ncol=2), proj4string=P4S.latlon )
# Make bounding box as Spatial Polygon
bb <- matrix(c(3,2,5,4),nrow=2)
rownames(bb) <- c("lon","lat")
colnames(bb) <- c('min','max')
bbSP <- as.SpatialPolygons.bbox(bb, proj4stringTo=P4S.latlon )
# Clip to extent
plyClip <- clipToExtent( ply, bbSP )
pntClip <- clipToExtent( pnt, bbSP )
# Plot
plot( ply )
plot( pnt, add=TRUE )
plot( bbSP, add=TRUE, border="blue" )
```
closestPoint

Find closest point to a given point’s coordinates (closestPoint).

Usage

closestPoint(point, points)

Arguments

points points is an Nx2 matrix, where columns are x,y and rows are the two points
point the point from which you want to find the closest match in points

Value

Distance, in same units as input

countPointsInPolys

Count points within a polygon

Description

Overlays points on polygons and create a new polygon dataset with the count of the points in that polygon

Usage

countPointsInPolys(points, polys, density = FALSE,
by = NULL)

Arguments

points SpatialPoints
polys SpatialPolygonsDataFrame
density Return a density (point count divided by area) instead of a point count
by Factor to return counts by. For instance, if by is a factor with two levels, instead of a single count variable being returned, two variables will be returned– the count of point type A in the polygon, and the count of point type B. The by factor must be of length length(points).
Value

SpatialPolygonsDataFrame

See Also

See Also as overlay

---

cumDist

*Calculate cumulative distance along a matrix of x,y coordinates*

Description

Calculate cumulative distance along a matrix of x,y coordinates

Usage

cumDist(coords)

Arguments

coords an [n,2] matrix of coordinates

Value

A single value that represents the distance along the path

See Also

See Also reshapeSLDF, SLDFtoLine

---

geocode

*Geocode character vectors using online services*

Description

Geocode character vectors (or data.frames) using Google or Bing’s API
Usage

geocode(x, verbose = FALSE, service = "google",
         returntype = "coordinates", ...)

## Default S3 method:
geocode(x, verbose = FALSE,
         service = "google", returntype = "coordinates", ...)

## S3 method for class 'data.frame'
geocode(x, verbose = FALSE,
         service = "google", returntype = "coordinates",
         addresscol = "address", ...)

Arguments

  x          A vector or data.frame
  verbose    Whether to display each address as it is submitted to Google or not
  service    API to use. Current options are "bing" or "google". To add your Bing API key,
             set options(BingMapsKey='yourkeygoeshere').
  returntype What to return. Options include "coordinates" and "zip".
  addresscol A (character) name of the column in a data.frame which contains the addresses
  ...        Other items to pass along

Value

geocode.default returns a numeric vector of length 2 containing the latitudes and longitudes. geocode.data.frame
returns the original data.frame with two additional columns for the longitude and latitudes.

Author(s)

Error handling, object orientation, and Bing capabilities by Ari Friedman. Google REST algorithm
by Tony Breyal (http://stackoverflow.com/a/3259537/636656).

Examples

## Not run:
geocode("3817 Spruce St, Philadelphia, PA 19104")
geocode("Philadelphia, PA")
dat <- data.frame(  
  value=runif(3),  
  address=c("3817 Spruce St, Philadelphia, PA 19104", "Philadelphia, PA", "Neverneverland")
)
geocode(dat)

## End(Not run)
georoute  

Find driving routes using online services

Description

Find transit routes using Google or Bing’s API

Usage

```r
georoute(x, verbose = FALSE, service = "bing", returntype = "all", ...)
```

## Default S3 method:
```r
georoute(x, verbose = FALSE, service = "bing", returntype = "all", ...)
```

Arguments

- **x**  
  A character vector of length>=2, where each element is a (starting, ending, or intermediate) waypoint, or a numeric matrix with columns c('lat', 'lon') where each row is a waypoint
- **verbose**  
  Provide additional information
- **returntype**  
  What information to return. Currently, the options are "all", "distance", "distanceUnit", "path", "time", and/or "timeUnit". Can be combined, as in returntype=c("time","distance").
- **service**  
  API to use. Currently the only option is "bing"
- ...  
  Other items to pass along

Value

Route information (see the returntype argument)

Author(s)

Ari B. Friedman

Examples

```r
## Not run:
georoute( c("3817 Spruce St, Philadelphia, PA 19104", "9000 Rockville Pike, Bethesda, Maryland 20892"), verbose=TRUE )
georoute( c("3817 Spruce St, Philadelphia, PA 19104", "Tulum, MX","9000 Rockville Pike, Bethesda, Maryland 20892"), returntype="distance" )
georoute( c("3817 Spruce St, Philadelphia, PA 19104", "9000 Rockville Pike, Bethesda, Maryland 20892"), verbose=TRUE, returntype="path" )
georoute( c("3817 Spruce St, Philadelphia, PA 19104", "9000 Rockville Pike, Bethesda, Maryland 20892"), verbose=TRUE, returntype="time" )
```
gGeoCode

# Using lat/lon
xmat <- rbind(
  geocode("3817 Spruce St, Philadelphia, PA 19104"),
  geocode("9000 Rockville Pike, Bethesda, Maryland 20892")
)
colnames(xmat) <- c('lat', 'lon')
georoute(xmat, verbose=TRUE, returntype = c("distance","distanceUnit"))

## End(Not run)

---

**gGeoCode**

*Geocode using Google Maps API (deprecated) This has been generalized to the taRifx::geocode() function*

### Description

Geocode using Google Maps API (deprecated) This has been generalized to the taRifx::geocode() function

### Usage

`gGeoCode(...)`

### Arguments

... Ignored

### IDs

*Get sp feature IDs*

### Description

Get sp feature IDs

Assign sp feature IDs

### Usage

`IDs(x, ...)`

## Default S3 method:

`IDs(x, ...)`

## S3 method for class 'SpatialPolygonsDataFrame'

`IDs(x, ...)`

`IDs(x) <- value`

`IDs(x) <- value`
**Arguments**

- `x` The object to get the IDs from or assign to
- `value` The character vector to assign to the IDs
- `...` Pass-alongs

**Author(s)**

Ari B. Friedman

---

### interpolateAndApplyWithinSpatial

_Generic function to interpolate from a polygon to points lying inside it_ This function was designed to solve the following problem. Suppose you have counts of the number of entities inside a polygon (N). To compute distances to a point, you might take the distance from the polygon centroid. But this is too simplistic—it discards the positional uncertainty inherent in not knowing the exact location of each entity which makes up the count. Instead, we repeatedly sample N points from the census block group centroids which lie within our polygon, weight them by their population, and compute distances from there.

### Description

Generic function to interpolate from a polygon to points lying inside it This function was designed to solve the following problem. Suppose you have counts of the number of entities inside a polygon (N). To compute distances to a point, you might take the distance from the polygon centroid. But this is too simplistic—it discards the positional uncertainty inherent in not knowing the exact location of each entity which makes up the count. Instead, we repeatedly sample N points from the census block group centroids which lie within our polygon, weight them by their population, and compute distances from there.

### Usage

```r
test <- interpolateAndApplyWithinSpatial(crude, fine, FUN = mean, nsamplecol = "population", samplesize = 30, simplify = FALSE, ...)
```

### Arguments

- `crude` A `SpatialPolygonsDataFrame` containing the variable of interest
- `fine` An `sp` object whose points or polygons lie inside the polygons of `crude`
- `FUN` An function taking at least two arguments (a single polygon of `crude` in the first position, and the within-polygon elements of `fine` in the second)
- `nsamplecol` Column name in `crude` containing number of elements of `fine` to sample per polygon
- `samplesize` The number of replicates per element of `crude` to draw
- `simplify` Whether to simplify to an array or not
- `...` Arguments to be passed to `FUN`
interpolatePathpoints

Value

list of length length(crude) where each element is a list of length samplesize containing the results of FUN for that crude-element/sample

Examples

# Not run because too time-consuming
## Not run:
require(fields)
require(rgdal)
distanceMatrix <- function(points1, points2, distFn=rdist.earth) {
  cat("Generating distance matrix for ", length(points1), ", by ", length(points2), " matrix.\n")
  if(!is.na(proj4string(points1))) {
    points1 <- sp::spTransform(points1, sp::CRS("+proj=longlat +datum=WGS84") )
  }
  if(!is.na(proj4string(points2))) {
    points2 <- sp::spTransform(points2, sp::CRS("+proj=longlat +datum=WGS84") )
  }
  distFn(coordinates(points1), coordinates(points2))
}

# One option: Use the apply functionality
dist <- interpolateAndApplyWithinSpatial(
  crude=poysP, fine=poysP, 
  FUN=distanceMatrix, 
  nSampleCol="z", samplesize=25, 
  points2=poysP2, simplify=TRUE
)

# Dist now is a list of S matrices, each with dimensions: n x length(pointSP2) x samplesize
# Each matrix represents N entities imputed from a single polygon,
# so we can actually simplify further
library(abind)
distmat <- do.call(Curry(abind, along=1), dist)
mindist <- apply(distmat, 3, function(x) { # For each realization of the 'world'
  apply(x, )
})

## End(Not run) # end of dontrun

interpolatePathpoints Interpolate points along a path

Description

Interpolate points along a path

Usage

interpolatePathpoints(pathpoints, dens, 
tolerance.min = 1.2, tolerance.max = 50)
Arguments

- **pathpoints**: Path points as they currently exist
- **dens**: inverse density and is in the units of the x and y in pathpoints (e.g. 1 point per density meters)
- **tolerance.min**: The proportion of the density (e.g. 1.2 means we’ll fill in gaps 20 size)
- **tolerance.max**: Max tolerance (see **tolerance.min**)

Value

- path with points interpolated

See Also

- See Also `reshapeSLDF, SLDfToLine`

---

**interpolatePolyPoint**

Interpolate points from polygon SPDF. This function returns (weighted) sample points in fine for every polygon in crude. Thus running it repeatedly gives you useful variation that reflects the interpolation uncertainty.

Description

Interpolate points from polygon SPDF. This function returns (weighted) sample points in fine for every polygon in crude. Thus running it repeatedly gives you useful variation that reflects the interpolation uncertainty.

Usage

```r
interpolatePolyPoint(crudeL, fineL, weightCol = NULLL,
nsampleCol = 1L, replace = TRUEL, verbose = FALSEL)
```

Arguments

- **crude**: A SpatialPolygonsDataFrame.
- **fine**: A SpatialPointsDataFrame.
- **weightCol**: A column name in fine to weight the point sampling by, or NULL if no weighting is required
- **nsampleCol**: Either a column name in crude containing number of elements of fineWithin to sample per polygon, or a number of points to sample per polygon
- **replace**: A logical indicating whether to sample points from fine with replacement or not within each polygon of crude
- **verbose**: Whether to output detailed error messages
Value

A SpatialPointsDataFrame containing

Examples

```r
## Not run:
replicate( 10, interpolatePolyPoint(
  crude=polySP, fine=pointSP,
  weightCol="pop", nSampleCol="z",
  replace=TRUE, verbose=TRUE ),
  simplify=FALSE )

## End(Not run)
```

**interpolateWithinSingleSpatial**

*Interpolate and sample within a single polygon (Called by interpolateWithinSpatial)*

**Description**

Interpolate and sample within a single polygon (Called by interpolateWithinSpatial)

**Usage**

```r
interpolateWithinSingleSpatial(crudeSingle, fineWithin, 
  FUN, nSampleCol, samplesize, simplify = FALSE, ...)
```

**Arguments**

- `crudeSingle`: A single polygon
- `fineWithin`: A points or polygon sp object whose elements lie within crudeSingle
- `FUN`: A function to be applied to them after sampling
- `nSampleCol`: Column name containing number of elements of fineWithin to sample
- `samplesize`: The number of replicates per element of crude to draw
- `simplify`: Whether to simplify to an array or not
- `...`: Arguments to FUN

**Value**

List of FUN’s results for each sampling
lineDist

Line distance in SpatialLinesDataFrame Stores length of each line segment in a SpatialLinesDataFrame’s data.frame

Description

Line distance in SpatialLinesDataFrame Stores length of each line segment in a SpatialLinesDataFrame’s data.frame

Usage

lineDist(SLDF, varname = "distances")

Arguments

SLDF SpatialLinesDataFrame
varname Character string containing name of variable to hold line distances.

Value

Returns a SpatialLinesDataFrame

merge

Merge a SpatialPolygonsDataFrame with a data.frame

Description

Merge a SpatialPolygonsDataFrame with a data.frame

Arguments

SPDF A SpatialPolygonsDataFrame
df A data.frame
... Parameters to pass to merge.data.frame
**overlayPolyPoly**


**Description**


**Usage**

`overlayPolyPoly(poly1, poly2)`

**Arguments**

- `poly1` : SPDF
- `poly2` : SPDF

**Value**

A SPDF with nested polygons of each

---

**pointDistPairwise**

| pointDistPairwise | Create all pairwise distances of points from a SpatialPointsDataFrame |

**Description**

Create all pairwise distances of points from a SpatialPointsDataFrame

**Usage**

`pointDistPairwise(SPDF, names = "name")`

**Arguments**

- `SPDF` : SpatialPointsDataFrame
- `names` : variable name in the SPDF’s dataframe used to label each point in the resulting matrix

**Value**

matrix
pointgrid2SpatialPolygons

Take a grid of regularly spaced points (such as those output by the centroids of Arc’s fishnet function) and convert it to various grid data types.

Description

Take a grid of regularly spaced points (such as those output by the centroids of Arc’s fishnet function) and convert it to various grid data types.

Usage

pointgrid2SpatialPolygons(df, type)

Arguments

df  SpatialPointsDataFrame

type  "SpatialGrid" or "SpatialPolygons" or "SpatialPolygonsDataFrame"

Value

SpatialGrid, SpatialPolygons, SpatialPolygonsDataFrame

rbind.SpatialPolygonsDataFrame

rbind SpatialPolygonsDataFrames together, fixing IDs if duplicated

Description

rbind SpatialPolygonsDataFrames together, fixing IDs if duplicated.

Usage

## S3 method for class 'SpatialPolygonsDataFrame'

rbind(...,

fix.duplicated.IDs = TRUE)

Arguments

...  SpatialPolygonsDataFrame(s) to rbind together

fix.duplicated.IDs  Whether to de-duplicate polygon IDs or not
**reshapeSLDF**

**Value**

SpatialPolygonsDataFrame

**Author(s)**

Ari B. Friedman, with key functionality by csfowler on StackExchange

---

**reshapeSLDF**

Reshape a spatialLinesDataFrame into a series of points with associated information

---

**Description**

Reshape a spatialLinesDataFrame into a series of points with associated information (less efficient because all the segment data gets replicated over each point)

**Usage**

```r
reshapeSLDF(SLDF, shape = "long")
```

**Arguments**

- **SLDF**: spatialLinesDataFrame
- **shape**: Do not change. Must be "long". For forward compatibility.

**Value**

data.frame

**See Also**

See Also `SLDFtoLine`

---

**simpledist**

Cartesian distance between points

---

**Description**

Cartesian distance between points

**Usage**

```r
simpledist(points)
```
**SLDFtoLine**

Convert a SpatialLinesDataFrame to a single line matrix with associated segment information.

**Description**

Convert a SpatialLinesDataFrame to a single line matrix with associated segment information.

**Usage**

```
SLDFtoLine(lineDF, orderXY = FALSE, segments = TRUE)
```

**Arguments**

- `lineDF`: SpatialLinesDataFrame object
- `orderXY`: ordering
- `segments`: segments to associate

**Value**

data.frame

**See Also**

See Also reshapeSLDF
smoothLines

Smooth the line segments in a SpatialLinesDataFrame

Description
Smooth the line segments in a SpatialLinesDataFrame. Takes a jittery SLDF (such as a GPS produces) and smooths it.

Usage
smoothLines(lineDF)

Arguments

linedf SpatialLinesDataFrame

Value
SpatialLinesDataFrame

See Also
See Also as SLDFile, ~~~

spdfareas

Return areas of polygons in a SpatialPolygonsDataFrame

Description
Get the areas stored in the polygons and return them in the dataframe slot

Usage
spdfareas(SPDF, colname = "AREA")

Arguments

SPDF SpatialPolygonsDataFrame
colname Name of the column in the data frame component of the SpatialPolygonsDataFrame in which to store the polygon areas

Value
SpatialPolygonsDataFrame
SPDFtoPointsDF

Convert SpatialPointsDataFrame to a regular data.frame with the coordinates as "x" and "y"

Description

Convert SpatialPointsDataFrame to a regular data.frame with the coordinates as "x" and "y"

Usage

SPDFtoPointsDF(SPDF)

Arguments

SPDF SpatialPointsDataFrame

Value

data.frame with the coordinates as "x" and "y"

Srs1

Entirely fabricated spatial data for taRifx.geo examples

Description

Entirely fabricated spatial data for taRifx.geo examples

states

U.S. State names, abbreviations, and FIPS codes

Description

U.S. State names, abbreviations, and FIPS codes
subsetSPDF

**Description**

Subset SpatialPolygonsDataFrame or SpatialPointsDataFrame

**Usage**

```r
subsetSPDF(x, tf, ...)
```

**Arguments**

- `x` SpatialPolygonsDataFrame or SpatialPointsDataFrame
- `tf` Boolean on which to subset
- `...` Arguments to pass to SpatialPolygonsDataFrame() or SpatialPointsDataFrame() when reconstructing objects

**Value**

SpatialPolygonsDataFrame or SpatialPointsDataFrame
Index

+Topic datasets
  Srs1, 20
  states, 20
+Topic data
  Srs1, 20
  states, 20
  as.SpatialPolygons.bbox, 2, 4
  box (Srs1), 20
  cleanLatLon, 3
  clipToExtent, 3, 4
  closestPoint, 5
  countPointsInPolys, 5
  cumDist, 6
  geocode, 6
  georoute, 8
  gGeoCode, 9
  IDs, 9
  IDs<- (IDs), 9
  interpolateAndApplyWithinSpatial, 10
  interpolatePathpoints, 11
  interpolatePolyPoint, 12
  interpolateWithinSingleSpatial, 13
  lineDist, 14
  merge, 14
  merge, SpatialPolygonsDataFrame, data.frame-method
    (merge), 14
  overlay, 6
  overlayPolyPoly, 15
  pointDistPairwise, 15
  pointgrid2SpatialPolygons, 16
  pointSP (Srs1), 20
  pointSP2 (Srs1), 20
  polySP (Srs1), 20
  rbind.SpatialPolygonsDataFrame, 16
  reshapeSLDF, 6, 12, 17, 18
  simpledist, 17
  SLDToLine, 6, 12, 17, 18, 19
  smoothLines, 19
  SPDFareas, 19
  SPDToPointsDF, 20
  Srs1, 20
  Srs2 (Srs1), 20
  Srs3 (Srs1), 20
  Srs4 (Srs1), 20
  states, 20
  subset.SpatialPointsDataFrame
    (subsetSPDF), 21
  subset.SpatialPolygonsDataFrame
    (subsetSPDF), 21
  subsetSPDF, 21