Package 'SpatialPosition'

September 6, 2017

Title Spatial Position Models

Version 1.2.0

Date 2017-09-06

Description Computes spatial position models: Stewart potentials, Reilly catchment areas, Huff catchment areas.

Depends R (>= 3.0.0), raster

License GPL-3

LazyData true

Imports sp, grDevices, graphics, methods, rgeos, rgdal

Suggests parallel, doParallel, foreach, cartography, knitr, rmarkdown

URL https://github.com/Groupe-ElementR/SpatialPosition

BugReports https://github.com/Groupe-ElementR/SpatialPosition/issues

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 6.0.1

NeedsCompilation no

Author Timothée Giraud [cre, aut],
Hadrien Commenges [aut],
Joël Boulier [ctb]

Maintainer Timothée Giraud <timothee.giraud@cnrs.fr>

Repository CRAN

Date/Publication 2017-09-06 12:50:29 UTC

R topics documented:

`contourStewart` *Create a SpatialPolygonsDataFrame or a SpatialLinesDataFrame from a Stewart Raster*

Description

`contourStewart` is deprecated.
To obtain contour lines use `rasterToContour` from raster package.
To obtain contour polygons use `rasterToContourPoly` from SpatialPosition package.

This function creates a SpatialPolygonsDataFrame or SpatialLinesDataFrame contour from the Stewart raster.

Usage

```
contourStewart(x, breaks, mask, type = "line")
```

Arguments

`x`	raster; output of the `rasterStewart` function. The raster must contain only positive values.
`breaks`	numeric; a vector of break values.
`mask`	SpatialPolygonsDataFrame; mask used to clip contour shapes.
`type`	character; "poly" or "line". WARNING: the poly option is experimental (see details). It needs the rgeos package.

Details

To obtain a correct SpatialPolygonsDataFrame of potentials follow theses steps:

- Step 1: Create a SpatialPointsDataFrame of potentials with the stewart function. Do not enter an unknownpts layer, set a resolution, and set a SpatialPolygonsDataFrame (spmask) as mask.
- Step 2: Create a raster from the SpatialPointsDataFrame of potentials with the rasterStewart function without using a mask.
- Step 3: Create the SpatialPolygonsDataFrame of potentials with the contourStewart function and use the same spmask SpatialPolygonsDataFrame (Step1) as mask.

See also the second example in the examples section.

Value

The ouput of the function is a SpatialPolygonsDataFrame (`type = "poly"`) or a SpatialLinesDataFrame (`type = "line"`). The data frame of the outputed SpatialPolygonsDataFrame contains four fields: id (id of each polygon), min and max (minimum and maximum breaks of the polygon), mean (center value of the class)

See Also

rasterToContourPoly.

Examples

```
data("spatData")
## Not run:
#### Example with type = "line"
mystewart <- stewart(knownpts = spatPts, varname = "Capacite",
                     typefct = "exponential", span = 1000, beta = 3,
                     resolution = 50,
                     mask = spatMask)
# Create a raster of potentials values
mystewartraster <- rasterStewart(x = mystewart, mask = spatMask)
# Display the raster and get break values
break.values <- plotStewart(x = mystewartraster)
# Create contour SpatialLinesDataFrame
mystewartcontourpoly <- contourStewart(x = mystewartraster,
                                       breaks = break.values,
                                       type = "line")
# Display the Map
plot(spatMask, add=TRUE)
plot(mystewartcontourpoly, border = "grey40",add = TRUE)
plot(spatPts, cex = 0.8, pch = 20, col  = "black", add = TRUE)



#### Example with type = "poly"
mystewart <- stewart(knownpts = spatPts, varname = "Capacite",
                     typefct = "exponential", span = 1000, beta = 3,
                     resolution = 50, longlat = FALSE,
```

```
                    mask = spatMask)
# Create a raster of potentials valuesn, no mask
mystewartraster <- rasterStewart(x = mystewart)
# Display the raster and get break values
break.values <- plotStewart(x = mystewartraster)
# Create contour SpatialLinesDataFrame
mystewartcontourpoly <- contourStewart(x = mystewartraster,
                                       breaks = break.values,
                                       mask = spatMask,
                                       type = "poly")
# Display the map
library(cartography)
opar <- par(mar = c(0,0,1.1,0))
choroLayer(spdf = mystewartcontourpoly,
           df = mystewartcontourpoly@data,
           var = "mean", legend.pos = "topleft",
           breaks = break.values, border = "grey90",
           lwd = 0.2,
           legend.title.txt = "Potential number\nof beds in the\nneighbourhood",
           legend.values.rnd = 0)
plot(spatMask, add = TRUE)
propSymbolsLayer(spdf = spatPts, df = spatPts@data, var = "Capacite",
                 legend.title.txt = "Number of beds",
                 col = "#ff000020")
layoutLayer(title = "Global Accessibility to Public Hospitals",
            south = TRUE, sources = "", author = "")
par(opar)

## End(Not run)
```

`CreateDistMatrix` *Create a Distance Matrix Between Two Sp Objects*

Description

This function creates a distance matrix between two sp objects (SpatialPointsDataFrame or SpatialPolygonsDataFrame).

Usage

```
CreateDistMatrix(knownpts, unknownpts, bypassctrl = FALSE, longlat = TRUE)
```

Arguments

`knownpts`	sp object; rows of the distance matrix.
`unknownpts`	sp object; columns of the distance matrix.
`bypassctrl`	logical; bypass the distance matrix size control (see Details).
`longlat`	logical; if FALSE, Euclidean distance, if TRUE Great Circle (WGS84 ellipsoid) distance.

Details

The function returns a full matrix of distances in meters. This is a wrapper for the `spDists` function.

If the matrix to compute is too large (more than 100,000,000 cells, more than 10,000,000 origins or more than 10,000,000 destinations) the function sends a confirmation message to warn users about the amount of RAM mobilized. Use `bypassctrl` = TRUE to skip this control.

Value

A distance matrix, row names are `knownpts` row names, column names are `unknownpts` row names.

See Also

CreateGrid

Examples

```
# Create a SpatialPointsDataFrame grid of spatMask extent and 200 meters
# resolution
data(spatData)
mygrid <- CreateGrid(w = spatMask, resolution = 200)
# Create a distance matrix between known spatPts and mygrid
mymat <- CreateDistMatrix(knownpts = spatPts, unknownpts = mygrid)
mymat[1:5,1:5]
nrow(spatPts)
nrow(mygrid)
dim(mymat)
```

`CreateGrid` *Create a Regularly Spaced SpatialPointsDataFrame*

Description

This function creates a regular grid of SpatialPointsDataFrame from the extent of a given sp object and a given resolution.

Usage

```
CreateGrid(w, resolution)
```

Arguments

`w`	sp object; the spatial extent of this object is used to create the regular SpatialPointsDataFrame.
`resolution`	numeric; resolution of the grid (in map units). If resolution is not set, the grid will contain around 7500 points. (optional)

Value

The output of the function is a SpatialPointsDataFrame of regularly spaced points with the same extent as w.

See Also

CreateDistMatrix

Examples

```
# Create a SpatialPointsDataFrame grid of spatMask extent and 200 meters
# resolution
data(spatData)
mygrid <- CreateGrid(w = spatMask, resolution = 200)
plot(mygrid, cex = 0.1, pch = ".")
plot(spatMask, border="red", lwd = 2, add = TRUE)
```

---

huff *Huff Catchment Areas*

Description

This function computes the catchment areas as defined by D. Huff (1964).

Usage

```
huff(knownpts, unknownpts = NULL, matdist = NULL, varname,
  typefct = "exponential", span, beta, resolution = NULL, mask = NULL,
  bypassctrl = FALSE, longlat = TRUE)
```

Arguments

knownpts	sp object (SpatialPointsDataFrame or SpatialPolygonsDataFrame); this is the set of known observations to estimate the catchment areas from.
unknownpts	sp object (SpatialPointsDataFrame or SpatialPolygonsDataFrame); this is the set of unknown units for which the function computes the estimates. Not used when resolution is set up. (optional)
matdist	matrix; distance matrix between known observations and unknown units for which the function computes the estimates. Row names match the row names of knownpts and column names match the row names of unknownpts. matdist can contain any distance metric (time distance or euclidean distance for example). If matdist is NULL, the distance matrix is built with CreateDistMatrix. (optional)
varname	character; name of the variable in the knownpts dataframe from which values are computed. Quantitative variable with no negative values.

typefct	character; spatial interaction function. Options are "pareto" (means power law) or "exponential". If "pareto" the interaction is defined as: (1 + alpha * mDistance) ^ (-beta). If "exponential" the interaction is defined as: exp(- alpha * mDistance ^ beta). The alpha parameter is computed from parameters given by the user (beta and span).
span	numeric; distance where the density of probability of the spatial interaction function equals 0.5.
beta	numeric; impedance factor for the spatial interaction function.
resolution	numeric; resolution of the output SpatialPointsDataFrame (in map units). If resolution is not set, the grid will contain around 7250 points. (optional)
mask	sp object; the spatial extent of this object is used to create the regularly spaced SpatialPointsDataFrame output. (optional)
bypassctrl	logical; bypass the distance matrix size control (see CreateDistMatrix Details).
longlat	logical; if FALSE, Euclidean distance, if TRUE Great Circle (WGS84 ellipsoid) distance.

Value

SpatialPointsDataFrame with the computed catchment areas in a new field named OUTPUT.

References

HUFF D. (1964) Defining and Estimating a Trading Area. Journal of Marketing, 28: 34-38.

See Also

huff, rasterHuff, plotHuff, CreateGrid, CreateDistMatrix.

Examples

```
# Create a SpatialPointsDataFrame grid of spatMask extent and 200 meters
# resolution
data(spatData)
mygrid <- CreateGrid(w = spatMask, resolution = 200)
# Create a distance matrix between known points (spatPts) and mygrid
mymat <- CreateDistMatrix(knownpts = spatPts, unknownpts = mygrid)
# Compute Huff catchment areas from known points (spatPts) on a given
# grid (mygrid) using a given distance matrix (mymat)
myhuff <- huff(knownpts = spatPts, unknownpts = mygrid,
               matdist = mymat, varname = "Capacite",
               typefct = "exponential", span = 1250,
               beta = 3, mask = spatMask)
# Compute Huff catchment areas from known points (spatPts) on a
# grid defined by its resolution
myhuff2 <- huff(knownpts = spatPts, varname = "Capacite",
                     typefct = "exponential", span = 1250, beta = 3,
                     resolution = 200, mask = spatMask)
# The two methods have the same result
```

```
identical(myhuff, myhuff2)
# the function output a SpatialPointsDataFrame
class(myhuff)
```

---

mcStewart *Stewart Potentials Parallel*

Description

This function computes Stewart potentials using parallel computation.

Usage

```
mcStewart(knownpts, unknownpts = NULL, varname, typefct = "exponential",
  span, beta, resolution = NULL, mask = NULL, cl = NULL, size = 1000,
  longlat = TRUE)
```

Arguments

knownpts	sp object (SpatialPointsDataFrame or SpatialPolygonsDataFrame); this is the set of known observations to estimate the potentials from.
unknownpts	sp object (SpatialPointsDataFrame or SpatialPolygonsDataFrame); this is the set of unknown units for which the function computes the estimates. Not used when resolution is set up. (optional)
varname	character; name of the variable in the knownpts dataframe from which potentials are computed. Quantitative variable with no negative values.
typefct	character; spatial interaction function. Options are "pareto" (means power law) or "exponential". If "pareto" the interaction is defined as: (1 + alpha * mDistance) ^ (-beta). If "exponential" the interaction is defined as: exp(- alpha * mDistance ^ beta). The alpha parameter is computed from parameters given by the user (beta and span).
span	numeric; distance where the density of probability of the spatial interaction function equals 0.5.
beta	numeric; impedance factor for the spatial interaction function.
resolution	numeric; resolution of the output SpatialPointsDataFrame (in map units). If resolution is not set, the grid will contain around 7250 points. (optional)
mask	sp object; the spatial extent of this object is used to create the regularly spaced SpatialPointsDataFrame output. (optional)
cl	numeric; number of clusters. By default cl is determined using parallel::detectCores().
size	numeric; mcStewart splits unknownpts in chunks, size indicates the size of each chunks.
longlat	logical; if FALSE, Euclidean distance, if TRUE Great Circle (WGS84 ellipsoid) distance.

Details

The parallel implementation splits potentials computations along chunks of unknownpts (or chunks of the grid defined using resolution). It only uses Great Cercle distances (with `CreateDistMatrix`).

Value

A Spatial*DataFrame with the computed potentials in a new field named `OUTPUT` is returned.

See Also

stewart.

Examples

```
## Not run:
if(require(cartography)){
  nuts3.spdf@data <- nuts3.df
  t1 <- system.time(
    s1 <- stewart(knownpts = nuts3.spdf,resolution = 40000,
                  varname = "pop2008",
                  typefct = "exponential", span = 100000,
                  beta = 3, mask = nuts3.spdf)
  )

  t2 <- system.time(
    s2 <- mcStewart(knownpts = nuts3.spdf, resolution = 40000,
                    varname = "pop2008",
                    typefct = "exponential", span = 100000,
                    beta = 3, mask = nuts3.spdf, cl = 3, size = 500)
  )
  identical(s1, s2)
  cat("Elapsed time\n", "stewart:", t1[3], "\n parStewart:",t2[3])

  r2 <- rasterStewart(s2)
  c2 <- rasterToContourPoly(r = r2, breaks = c(0,1000000,2000000, 5000000,
                                               10000000, 20000000, 200004342),
                            mask = nuts3.spdf)
  # cartography
  opar <- par(mar = c(0,0,1.2,0))
  bks <- sort(unique(c(c2$min, c2$max)))
  choroLayer(spdf = c2, var = "center", breaks = bks, border = NA,
             legend.title.txt = "pop")
  layoutLayer("potential population", "","", scale = NULL)
  par(opar)
}

## End(Not run)
```

plotHuff *Plot a Huff Raster*

Description

This function plots the raster produced by the rasterHuff function.

Usage

```
plotHuff(x, add = FALSE)
```

Arguments

x	raster; output of the rasterHuff function.
add	logical; if TRUE the raster is added to the current plot, if FALSE the raster is displayed in a new plot.

Value

Display the raster nicely.

See Also

huff, rasterHuff, plotHuff, CreateGrid, CreateDistMatrix.

Examples

```
data(spatData)
# Compute Huff catchment areas from known points (spatPts) on a
# grid defined by its resolution
myhuff <- huff(knownpts = spatPts, varname = "Capacite",
               typefct = "exponential", span = 750, beta = 2,
               resolution = 100, mask = spatMask)
# Create a raster of huff values
myhuffraster <- rasterHuff(x = myhuff, mask = spatMask)
plotHuff(myhuffraster)
```

plotReilly *Plot a Reilly Raster*

Description

This function plots the raster produced by the rasterReilly function.

Usage

```
plotReilly(x, add = FALSE, col = rainbow)
```

Arguments

x	raster; output of the rasterReilly function.
add	logical; if TRUE the raster is added to the current plot, if FALSE the raster is displayed in a new plot.
col	function; color ramp function, such as colorRampPalette.

Details

Display the raster nicely.

See Also

reilly, rasterReilly, plotReilly, CreateGrid, CreateDistMatrix.

Examples

```
data(spatData)
row.names(spatPts) <- spatPts$CodHop
# Compute Reilly catchment areas from known points (spatPts) on a
# grid defined by its resolution
myreilly <- reilly(knownpts = spatPts, varname = "Capacite",
                   typefct = "exponential", span = 750, beta = 2,
                   resolution = 100, mask = spatMask)
# Create a raster of reilly values
myreillyraster <- rasterReilly(x = myreilly, mask = spatMask)
# Plot the raster nicely
plotReilly(x = myreillyraster)
```

---

plotStewart *Plot a Stewart Raster*

Description

This function plots the raster produced by the rasterStewart function.

Usage

```
plotStewart(x, add = FALSE, breaks = NULL, typec = "equal", nclass = 5,
  legend.rnd = 0, col = colorRampPalette(c("#FEA3A3", "#980000")))
```

Arguments

x	raster; output of the rasterStewart function.
add	logical; if TRUE the raster is added to the current plot, if FALSE the raster is displayed in a new plot.
breaks	numeric; vector of break values to map. If used, this parameter overrides typec and nclass parameters

typec	character; either "equal" or "quantile", how to discretize the values.
nclass	numeric (integer), number of classes.
legend.rnd	numeric (integer); number of digits used to round the values displayed in the legend.
col	function; color ramp function, such as colorRampPalette.

Value

Display the raster nicely and return the list of break values (invisible).

See Also

stewart, rasterStewart, quickStewart, rasterToContourPoly, CreateGrid, CreateDistMatrix.

Examples

```
data(spatData)
# Compute Stewart potentials from known points (spatPts) on a
# grid defined by its resolution
mystewart <- stewart(knownpts = spatPts, varname = "Capacite",
                     typefct = "exponential", span = 1000, beta = 3,
                     resolution = 100, mask = spatMask)
# Create a raster of potentials values
mystewartraster <- rasterStewart(x = mystewart, mask = spatMask)
# Plot stewart potentials nicely
plotStewart(x = mystewartraster, add = FALSE, nclass = 5)
# Can be used to obtain break values
break.values <- plotStewart(x = mystewartraster, add = FALSE, nclass = 5)
break.values
```

---

quickStewart *Create a SpatialPolygonsDataFrame of Potentials Contours*

Description

This function is a wrapper around stewart, rasterStewart and rasterToContourPoly functions. Providing only the main parameters of these functions, it simplifies a lot the computation of potentials. This function creates a SpatialPolygonsDataFrame of potential values. It also allows to compute directly the ratio between the potentials of two variables.

Usage

```
quickStewart(spdf, df, spdfid = NULL, dfid = NULL, var, var2 = NULL,
  typefct = "exponential", span, beta, resolution = NULL, mask = NULL,
  nclass = 8, breaks = NULL, bypassctrl = FALSE)
```

Arguments

spdf	a SpatialPolygonsDataFrame.
df	a data frame that contains the values to compute
spdfid	name of the identifier field in spdf, default to the first column of the spdf data frame. (optional)
dfid	name of the identifier field in df, default to the first column of df. (optional)
var	name of the numeric field in df used to compute potentials.
var2	name of the numeric field in df used to compute potentials. This field is used for ratio computation (see Details).
typefct	character; spatial interaction function. Options are "pareto" (means power law) or "exponential". If "pareto" the interaction is defined as: (1 + alpha * mDistance) ^ (-beta). If "exponential" the interaction is defined as: exp(- alpha * mDistance ^ beta). The alpha parameter is computed from parameters given by the user (beta and span).
span	numeric; distance where the density of probability of the spatial interaction function equals 0.5.
beta	numeric; impedance factor for the spatial interaction function.
resolution	numeric; resolution of the output SpatialPointsDataFrame (in map units). If resolution is not set, the grid will contain around 7250 points. (optional)
mask	SpatialPolygonsDataFrame; mask used to clip contours of potentials.
nclass	numeric; a targeted number of classes (default to 8). Not used if breaks is set.
breaks	numeric; a vector of values used to discretize the potentials.
bypassctrl	logical; bypass the distance matrix size control (see CreateDistMatrix Details).

Details

If var2 is provided the ratio between the potentials of var (numerator) and var2 (denominator) is computed.

Value

A SpatialPolygonsDataFrame is returned (see rasterToContourPoly Value).

See Also

stewart, rasterStewart, plotStewart, rasterToContourPoly, CreateGrid, CreateDistMatrix.

Examples

```
# load data
data("spatData")
# Compute a SpatialPolygonsDataFrame of potentials
pot.spdf <- quickStewart(spdf = spatPts,
                         df = spatPts@data,
```

```
                          var = "Capacite",
                          span = 1000,
                          beta = 2, mask = spatMask)
plot(pot.spdf)
# cartography
if(require("cartography")){
  breaks <- sort(c(unique(pot.spdf$min), max(pot.spdf$max)), decreasing = FALSE)
  cartography::choroLayer(spdf = pot.spdf, df = pot.spdf@data,
                          var = "center", breaks = breaks,
                          legend.pos = "topleft",
                          legend.title.txt = "Nb. of Beds")
}
pot.spdf@data


# Compute a SpatialPolygonsDataFrame of a ratio of potentials
spatPts$dummy <- spatPts$Capacite + c(rep(50, 18))
pot2.spdf <- quickStewart(spdf = spatPts,
                          df = spatPts@data,
                          var = "Capacite",
                          var2 = "dummy",
                          span = 1000,
                          beta = 2, mask = spatMask)
# cartography
if(require("cartography")){
  breaks <- sort(c(unique(pot2.spdf$min), max(pot2.spdf$max)), decreasing = FALSE)
  cartography::choroLayer(spdf = pot2.spdf, df = pot2.spdf@data,
                          var = "center", breaks = breaks,
                          legend.pos = "topleft",legend.values.rnd = 3,
                          legend.title.txt = "Nb. of Beds")
}
```

`rasterHuff` *Create a Raster from a Huff SpatialPointsDataFrame*

Description

This function creates a raster from a regularly spaced Huff SpatialPointsDataFrame (output of the `huff` function).

Usage

```
rasterHuff(x, mask = NULL)
```

Arguments

`x`	sp object (SpatialPointsDataFrame); output of the `huff` function.
`mask`	sp object (SpatialPolygonsDataFrame); this object is used to clip the raster. (optional)

Value

Raster of catchment areas values.

See Also

huff, rasterHuff, plotHuff, CreateGrid, CreateDistMatrix.

Examples

```
data(spatData)
# Compute Huff catchment areas from known points (spatPts) on a
# grid defined by its resolution
myhuff <- huff(knownpts = spatPts, varname = "Capacite",
               typefct = "exponential", span = 750, beta = 2,
               resolution = 100, mask = spatMask)
# Create a raster of huff values
myhuffraster <- rasterHuff(x = myhuff, mask = spatMask)
plot(myhuffraster)
```

---

rasterReilly *Create a Raster from a Reilly SpatialPointsDataFrame*

Description

This function creates a raster from a regularly spaced Reilly SpatialPointsDataFrame (output of the `reilly` function).

Usage

```
rasterReilly(x, mask = NULL)
```

Arguments

`x`	sp object (SpatialPointsDataFrame); output of the `reilly` function.
`mask`	sp object (SpatialPolygonsDataFrame); this object is used to clip the raster. (optional)

Value

Raster of catchment areas values. The raster uses a RAT (`ratify`) that contains the correspondance between raster values and catchement areas values. Use `unique(levels(rasterName)[[1]])` to see the correpondance table.

See Also

reilly, rasterReilly, plotReilly, CreateGrid, CreateDistMatrix.

Examples

```
data(spatData)
row.names(spatPts) <- spatPts$CodHop
# Compute Reilly catchment areas from known points (spatPts) on a
# grid defined by its resolution
myreilly <- reilly(knownpts = spatPts, varname = "Capacite",
                   typefct = "exponential", span = 750, beta = 2,
                   resolution = 100, mask = spatMask)
# Create a raster of reilly values
myreillyraster <- rasterReilly(x = myreilly, mask = spatMask)
plot(myreillyraster, col = rainbow(18))
# Correspondance between raster values and reilly areas
head(unique(levels(myreillyraster)[[1]]))
```

---

rasterStewart *Create a Raster from a Stewart SpatialPointsDataFrame*

Description

This function creates a raster from a regularly spaced Stewart SpatialPointsDataFrame (output of the `stewart` function).

Usage

```
rasterStewart(x, mask = NULL)
```

Arguments

`x`	sp object (SpatialPointsDataFrame); output of the `stewart` function.
`mask`	sp object (SpatialPolygonsDataFrame); this object is used to clip the raster. (optional)

Value

Raster of potential values.

See Also

stewart, quickStewart, plotStewart, rasterToContourPoly, CreateGrid, CreateDistMatrix.

Examples

```
data(spatData)
# Compute Stewart potentials from known points (spatPts) on a
# grid defined by its resolution
mystewart <- stewart(knownpts = spatPts, varname = "Capacite",
                     typefct = "exponential", span = 1000, beta = 3,
                     resolution = 100, mask = spatMask)
```

```
# Create a raster of potentials values
mystewartraster <- rasterStewart(x = mystewart, mask = spatMask)
plot(mystewartraster)
```

`rasterToContourPoly` *Create a SpatialPolygonsDataFrame from a Raster*

Description

This function creates a contour SpatialPolygonsDataFrame from a raster.

Usage

```
rasterToContourPoly(r, nclass = 8, breaks = NULL, mask = NULL)
```

Arguments

`r`	raster; the raster must contain only positive values.
`nclass`	numeric; a number of class.
`breaks`	numeric; a vector of break values.
`mask`	SpatialPolygonsDataFrame; mask used to clip contour shapes. The mask should have a smaller extent than r.

Details

This function uses the rgeos package.

Value

The ouput of the function is a SpatialPolygonsDataFrame. The data frame of the outputed SpatialPolygonsDataFrame contains four fields: id (id of each polygon), min and max (minimum and maximum breaks of the polygon), center (central values of classes)

See Also

stewart, rasterStewart, plotStewart, quickStewart, CreateGrid, CreateDistMatrix.

Examples

```
data("spatData")
## Not run:
mystewart <- stewart(knownpts = spatPts, varname = "Capacite",
                     typefct = "exponential", span = 1000, beta = 3,
                     resolution = 50, mask = spatMask)
# Create a raster of potentials values
mystewartraster <- rasterStewart(x = mystewart)
# Create contour SpatialLinesDataFrame
contourpoly <- rasterToContourPoly(r = mystewartraster,
```

```
                                    nclass = 6,
                                    mask = spatMask)
# Created breaks
bks <- sort(unique(c(contourpoly$min, contourpoly$max)))
# Display the map
library(cartography)
opar <- par(mar = c(0,0,1.2,0))
choroLayer(spdf = contourpoly,
           df = contourpoly@data,
           var = "center", legend.pos = "topleft",
           breaks = bks, border = "grey90",
           lwd = 0.2,
           legend.title.txt = "Potential number\nof beds in the\nneighbourhood",
           legend.values.rnd = 0)
plot(spatMask, add = TRUE)
propSymbolsLayer(spdf = spatPts, df = spatPts@data, var = "Capacite",
                 legend.title.txt = "Number of beds",
                 col = "#ff000020")
layoutLayer(title = "Global Accessibility to Public Hospitals",
            south = TRUE, sources = "", author = "")
par(opar)

## End(Not run)
```

`reilly` *Reilly Catchment Areas*

Description

This function computes the catchment areas as defined by W.J. Reilly (1931).

Usage

```
reilly(knownpts, unknownpts = NULL, matdist = NULL, varname,
  typefct = "exponential", span, beta, resolution = NULL, mask = NULL,
  bypassctrl = FALSE, longlat = TRUE)
```

Arguments

`knownpts`	sp object (SpatialPointsDataFrame or SpatialPolygonsDataFrame); this is the set of known observations to estimate the catchment areas from.
`unknownpts`	sp object (SpatialPointsDataFrame or SpatialPolygonsDataFrame); this is the set of unknown units for which the function computes the estimates. Not used when `resolution` is set up. (optional)
`matdist`	matrix; distance matrix between known observations and unknown units for which the function computes the estimates. Row names match the row names of `knownpts` and column names match the row names of `unknownpts`. `matdist` can contain any distance metric (time distance or euclidean distance for example). If `matdist` is NULL, the distance matrix is built with `CreateDistMatrix`. (optional)

varname	character; name of the variable in the knownpts dataframe from which values are computed. Quantitative variable with no negative values.
typefct	character; spatial interaction function. Options are "pareto" (means power law) or "exponential". If "pareto" the interaction is defined as: (1 + alpha * mDistance) ^ (-beta). If "exponential" the interaction is defined as: exp(- alpha * mDistance ^ beta). The alpha parameter is computed from parameters given by the user (beta and span).
span	numeric; distance where the density of probability of the spatial interaction function equals 0.5.
beta	numeric; impedance factor for the spatial interaction function.
resolution	numeric; resolution of the output SpatialPointsDataFrame (in map units). If resolution is not set, the grid will contain around 7250 points. (optional)
mask	sp object; the spatial extent of this object is used to create the regularly spaced SpatialPointsDataFrame output. (optional)
bypassctrl	logical; bypass the distance matrix size control (see CreateDistMatrix Details).
longlat	logical; if FALSE, Euclidean distance, if TRUE Great Circle (WGS84 ellipsoid) distance.

Value

SpatialPointsDataFrame with the computed catchment areas in a new field named OUTPUT. Values match the row names of knownpts.

References

REILLY, W. J. (1931) The law of retail gravitation, W. J. Reilly, New York.

See Also

reilly, rasterReilly, plotReilly, CreateGrid, CreateDistMatrix.

Examples

```
# Create a SpatialPointsDataFrame grid of spatMask extent and 200 meters
# resolution
data(spatData)
mygrid <- CreateGrid(w = spatMask, resolution = 200)
# Create a distance matrix between known points (spatPts) and mygrid
mymat <- CreateDistMatrix(knownpts = spatPts, unknownpts = mygrid)
# Compute Reilly catchment areas from known points (spatPts) on a given
# grid (mygrid) using a given distance matrix (mymat)
myreilly2 <- reilly(knownpts = spatPts, unknownpts = mygrid,
               matdist = mymat, varname = "Capacite",
               typefct = "exponential", span = 1250,
               beta = 3, mask = spatMask)
row.names(spatPts) <- spatPts$CodHop
# Compute Reilly catchment areas from known points (spatPts) on a
```

```
# grid defined by its resolution
myreilly <- reilly(knownpts = spatPts, varname = "Capacite",
                typefct = "exponential", span = 1250, beta = 3,
                resolution = 200, mask = spatMask)
# The function output a SpatialPointsDataFrame
class(myreilly)
# The OUTPUT field values match knownpts row names
head(unique(myreilly$OUTPUT))
```

---

smoothy *Stewart Smooth*

Description

This function computes a distance weighted mean. It offers the same parameters as stewart: user defined distance matrix, user defined impedance function (power or exponential), user defined exponent.

Usage

```
smoothy(knownpts, unknownpts = NULL, matdist = NULL, varname,
  typefct = "exponential", span, beta, resolution = NULL, mask = NULL,
  bypassctrl = FALSE, longlat = TRUE)
```

Arguments

knownpts	sp object (SpatialPointsDataFrame or SpatialPolygonsDataFrame); this is the set of known observations to estimate the potentials from.
unknownpts	sp object (SpatialPointsDataFrame or SpatialPolygonsDataFrame); this is the set of unknown units for which the function computes the estimates. Not used when resolution is set up. (optional)
matdist	matrix; distance matrix between known observations and unknown units for which the function computes the estimates. Row names match the row names of knownpts and column names match the row names of unknownpts. matdist can contain any distance metric (time distance or euclidean distance for example). If matdist is NULL, the distance matrix is built with CreateDistMatrix. (optional)
varname	character; name of the variable in the knownpts dataframe from which potentials are computed. Quantitative variable with no negative values.
typefct	character; spatial interaction function. Options are "pareto" (means power law) or "exponential". If "pareto" the interaction is defined as: (1 + alpha * mDistance) ^ (-beta). If "exponential" the interaction is defined as: exp(- alpha * mDistance ^ beta). The alpha parameter is computed from parameters given by the user (beta and span).
span	numeric; distance where the density of probability of the spatial interaction function equals 0.5.

beta	numeric; impedance factor for the spatial interaction function.
resolution	numeric; resolution of the output SpatialPointsDataFrame (in map units). If resolution is not set, the grid will contain around 7250 points. (optional)
mask	sp object; the spatial extent of this object is used to create the regularly spaced SpatialPointsDataFrame output. (optional)
bypassctrl	logical; bypass the distance matrix size control (see CreateDistMatrix Details).
longlat	logical; if FALSE, Euclidean distance, if TRUE Great Circle (WGS84 ellipsoid) distance.

Value

SpatialPointsDataFrame with the computed potentials in a new field named OUTPUT

See Also

stewart.

Examples

```
# Create a SpatialPointsDataFrame grid of spatMask extent and 200 meters
# resolution
data(spatData)
mygrid <- CreateGrid(w = spatMask, resolution = 200)
# Create a distance matrix between known points (spatPts) and mygrid
mymat <- CreateDistMatrix(knownpts = spatPts, unknownpts = mygrid)
# Compute Stewart potentials from known points (spatPts) on a given
# grid (mygrid) using a given distance matrix (mymat)
mystewart <- smoothy(knownpts = spatPts, unknownpts = mygrid,
                     matdist = mymat, varname = "Capacite",
                     typefct = "exponential", span = 1250,
                     beta = 3, mask = spatMask)
# Compute Stewart potentials from known points (spatPts) on a
# grid defined by its resolution
mystewart2 <- smoothy(knownpts = spatPts, varname = "Capacite",
                      typefct = "exponential", span = 1250, beta = 3,
                      resolution = 200, mask = spatMask)
# The two methods have the same result
identical(mystewart, mystewart2)
# the function output a SpatialPointsDataFrame
class(mystewart)
# Computed values
summary(mystewart$OUTPUT)
```

SpatialPosition *Spatial Position Package*

Description

Computes spatial position models:

- Stewart potentials,
- Reilly catchment areas,
- Huff catchment areas.

An introduction to the package conceptual background and usage:
- `vignette(topic = "SpatialPosition")`
A Stewart potentials use case:
- `vignette(topic = "StewartExample")`.

References

COMMENGES H., GIRAUD, T., LAMBERT, N. (2016) "ESPON FIT: Functional Indicators for Spatial-Aware Policy-Making", Cartographica: The International Journal for Geographic Information and Geovisualization, 51(3): 127-136.

See Also

stewart, rasterStewart, plotStewart, quickStewart, mcStewart, smoothy, rasterToContourPoly, huff, rasterHuff, plotHuff, reilly, rasterReilly, plotReilly, CreateGrid, CreateDistMatrix.

spatMask *Paris Perimeter*

Description

A SpatialPolygonsDataFrame of the Paris perimeter.

spatPts *Public Hospitals*

Description

A SpatialPointsDataFrame of 18 public hospitals with their capacity (Capacite field = number of beds).

spatUnits *Spatial Units of Paris*

Description

A SpatialPolygonsDataFrame of the 20 spatial arrondissements of the Paris.

stewart *Stewart Potentials*

Description

This function computes the potentials as defined by J.Q. Stewart (1942).

Usage

```
stewart(knownpts, unknownpts = NULL, matdist = NULL, varname,
  typefct = "exponential", span, beta, resolution = NULL, mask = NULL,
  bypassctrl = FALSE, longlat = TRUE)
```

Arguments

knownpts	sp object (SpatialPointsDataFrame or SpatialPolygonsDataFrame); this is the set of known observations to estimate the potentials from.
unknownpts	sp object (SpatialPointsDataFrame or SpatialPolygonsDataFrame); this is the set of unknown units for which the function computes the estimates. Not used when resolution is set up. (optional)
matdist	matrix; distance matrix between known observations and unknown units for which the function computes the estimates. Row names match the row names of knownpts and column names match the row names of unknownpts. matdist can contain any distance metric (time distance or euclidean distance for example). If matdist is NULL, the distance matrix is built with CreateDistMatrix. (optional)
varname	character; name of the variable in the knownpts dataframe from which potentials are computed. Quantitative variable with no negative values.
typefct	character; spatial interaction function. Options are "pareto" (means power law) or "exponential". If "pareto" the interaction is defined as: (1 + alpha * mDistance) ^ (-beta). If "exponential" the interaction is defined as: exp(- alpha * mDistance ^ beta). The alpha parameter is computed from parameters given by the user (beta and span).
span	numeric; distance where the density of probability of the spatial interaction function equals 0.5.
beta	numeric; impedance factor for the spatial interaction function.

resolution	numeric; resolution of the output SpatialPointsDataFrame (in map units). If resolution is not set, the grid will contain around 7250 points. (optional)
mask	sp object; the spatial extent of this object is used to create the regularly spaced SpatialPointsDataFrame output. (optional)
bypassctrl	logical; bypass the distance matrix size control (see CreateDistMatrix Details).
longlat	logical; if FALSE, Euclidean distance, if TRUE Great Circle (WGS84 ellipsoid) distance.

Value

SpatialPointsDataFrame with the computed potentials in a new field named OUTPUT

References

STEWART J.Q. (1942) "Measure of the influence of a population at a distance", Sociometry, 5(1): 63-71.

See Also

rasterStewart, plotStewart, quickStewart, rasterToContourPoly, CreateGrid, CreateDistMatrix.

Examples

```
# Create a SpatialPointsDataFrame grid of spatMask extent and 200 meters
# resolution
data(spatData)
mygrid <- CreateGrid(w = spatMask, resolution = 200)
# Create a distance matrix between known points (spatPts) and mygrid
mymat <- CreateDistMatrix(knownpts = spatPts, unknownpts = mygrid)
# Compute Stewart potentials from known points (spatPts) on a given
# grid (mygrid) using a given distance matrix (mymat)
mystewart <- stewart(knownpts = spatPts, unknownpts = mygrid,
                     matdist = mymat, varname = "Capacite",
                     typefct = "exponential", span = 1250,
                     beta = 3, mask = spatMask)
# Compute Stewart potentials from known points (spatPts) on a
# grid defined by its resolution
mystewart2 <- stewart(knownpts = spatPts, varname = "Capacite",
                      typefct = "exponential", span = 1250, beta = 3,
                      resolution = 200, mask = spatMask)
# The two methods have the same result
identical(mystewart, mystewart2)
# the function output a SpatialPointsDataFrame
class(mystewart)
# Computed values
summary(mystewart$OUTPUT)
```

Index