Package 'SpatialPosition'

June 14, 2021

Title Spatial Position Models

Version 2.1.1

Description Computes spatial position models: the potential model as defined by Stewart (1941) <doi:10.1126/science.93.2404.89> and catchment areas as defined by Reilly (1931) or Huff (1964) <doi:10.2307/1249154>.

Depends R (>= 3.5.0)

License GPL-3

LazyData true

Imports sf, sp, grDevices, graphics, methods, isoband, raster

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

URL https://github.com/riatelab/SpatialPosition

BugReports https://github.com/riatelab/SpatialPosition/issues

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.1.1

NeedsCompilation no

Author Timothée Giraud [cre, aut] (<https://orcid.org/0000-0002-1932-3323>), Hadrien Commenges [aut], Joël Boulier [ctb]

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

Repository CRAN

Date/Publication 2021-06-14 13:50:29 UTC

R topics documented:

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

Description

This function creates a distance matrix between two spatial objects (sp or sf objects).

Usage

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

Arguments

`knownpts`	sp or sf object; rows of the distance matrix.
`unknownpts`	sp or sf 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. 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 grid of paris extent and 200 meters
# resolution
data(hospital)
mygrid <- CreateGrid(w = paris, resolution = 200, returnclass = "sf")
# Create a distance matrix between known hospital and mygrid
mymat <- CreateDistMatrix(knownpts = hospital, unknownpts = mygrid,
                          longlat = FALSE)
mymat[1:5,1:5]
nrow(paris)
nrow(mygrid)
dim(mymat)
```

CreateGrid *Create a Regularly Spaced Points Grid*

Description

This function creates a regular grid of points from the extent of a given spatial object and a given resolution.

Usage

```
CreateGrid(w, resolution, returnclass = "sp")
```

Arguments

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

Value

The output of the function is a regularly spaced points grid with the extent of w.

See Also

CreateDistMatrix

Examples

```
# Create a grid of paris extent and 200 meters
# resolution
library(SpatialPosition)
library(sf)
data(hospital)
mygrid <- CreateGrid(w = paris, resolution = 200, returnclass = "sf")
plot(st_geometry(mygrid), cex = 0.1, pch = ".")
plot(st_geometry(paris), border="red", lwd = 2, add = TRUE)
```

---

hospital *Public Hospitals*

Description

An sf POINT data frame of 18 public hospitals with their capacity ("capacity" = number of beds).

---

huff *Huff Catchment Areas*

Description

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

Usage

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

Arguments

knownpts	sp or sf object; this is the set of known observations to estimate the catchment areas from.
unknownpts	sp or sf object; 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 not set, the distance matrix is automaticly 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 grid (in map units). If resolution is not set, the grid will contain around 7000 points. (optional)
mask	sp or sf object; the spatial extent of this object is used to create the regularly spaced points 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.
returnclass	"sp" or "sf"; class of the returned object.

Value

Point object 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 grid of paris extent and 200 meters
# resolution
data(hospital)
mygrid <- CreateGrid(w = paris, resolution = 200, returnclass = "sf")
# Create a distance matrix between known points (hospital) and mygrid
mymat <- CreateDistMatrix(knownpts = hospital, unknownpts = mygrid,
                          longlat = FALSE)
# Compute Huff catchment areas from known points (hospital) on a given
# grid (mygrid) using a given distance matrix (mymat)
myhuff <- huff(knownpts = hospital, unknownpts = mygrid,
               matdist = mymat, varname = "capacity",
               typefct = "exponential", span = 1250,
               beta = 3, mask = paris, returnclass = "sf")
# Compute Huff catchment areas from known points (hospital) on a
# grid defined by its resolution
myhuff2 <- huff(knownpts = hospital, varname = "capacity",
                typefct = "exponential", span = 1250, beta = 3,
                resolution = 200, mask = paris, returnclass= "sf")
# The two methods have the same result
identical(myhuff, myhuff2)
# the function output an sf object
class(myhuff)
```

`isopoly` *Create Spatial Polygons Contours from a Raster*

Description

This function creates spatial polygons of contours from a raster.

Usage

```
isopoly(
  x,
  nclass = 8,
  breaks,
  mask,
  xcoords = "COORDX",
  ycoords = "COORDY",
  var = "OUTPUT",
  returnclass = "sp"
)
```

Arguments

`x`	sf POINT data.frame; must contain X, Y and OUTPUT fields.
`nclass`	numeric; a number of class.

breaks	numeric; a vector of break values.
mask	sf POLYGON data.frame; mask used to clip contour shapes.
xcoords	character; name of the X coordinates field in x.
ycoords	character; name of the Y coordinates field in x.
var	character; name of the OUTPUT field in x.
returnclass	"sp" or "sf"; class of the returned object.

Value

The output is an sf POLYGON data.frame. The data frame 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.

Examples

```
data(hospital)
# Compute Stewart potentials
mystewart <- stewart(knownpts = hospital, varname = "capacity",
                     typefct = "exponential", span = 1000, beta = 3,
                     mask = paris, returnclass = "sf")
# Create contour
contourpoly <- isopoly(x = mystewart,
                       nclass = 6,
                       mask = paris, returnclass = "sf")
library(sf)
plot(st_geometry(contourpoly))
if(require(cartography)){
  # Created breaks
  bks <- sort(unique(c(contourpoly$min, contourpoly$max)))
  opar <- par(mar = c(0,0,1.2,0))
  # Display the map
  choroLayer(x = contourpoly,
             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(st_geometry(paris), add = TRUE)
  propSymbolsLayer(x = hospital, var = "capacity",
                   legend.pos = "right",
                   legend.title.txt = "Number of beds",
                   col = "#ff000020")
  layoutLayer(title = "Global Accessibility to Public Hospitals",
              sources = "", author = "")
  par(opar)
}
```

`mcStewart` *Stewart Potentials Parallel*

Description

This function computes Stewart potentials using parallel computation.

Usage

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

Arguments

`knownpts`	sp or sf object; this is the set of known observations to estimate the potentials from.
`unknownpts`	sp or sf object; 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 or sf object; the spatial extent of this object is used to create the regularly spaced points 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.
returnclass	"sp" or "sf"; class of the returned object.

Details

The parallel implementation splits potentials computations along chunks of unknownpts (or chunks of the grid defined using resolution).

Value

Point object with the computed potentials in a new field named OUTPUT.

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, returnclass = "sf")
  )
  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,
                    returnclass = "sf")
  )
  identical(s1, s2)
  cat("Elapsed time\n", "stewart:", t1[3], "\n mcStewart:",t2[3])

  iso <- isopoly(x = s2,
                 breaks = c(0,1000000,2000000, 5000000, 10000000, 20000000,
                            200004342),
                 mask = nuts3.spdf, returnclass = "sf")
  # cartography
  opar <- par(mar = c(0,0,1.2,0))
  bks <- sort(unique(c(iso$min, iso$max)))
  choroLayer(x = iso, var = "center", breaks = bks, border = NA,
             legend.title.txt = "pop")
  layoutLayer("potential population", "","", scale = NULL)
  par(opar)
```

```
}

## End(Not run)
```

`paris` *Paris Polygon*

Description

An sf POLYGON data frame of the Paris perimeter.

`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.

Examples

```
data(hospital)
# Compute Huff catchment areas from known points (hospital) on a
# grid defined by its resolution
myhuff <- huff(knownpts = hospital, varname = "capacity",
               typefct = "exponential", span = 750, beta = 2,
               resolution = 100, mask = paris, returnclass = "sf")
# Create a raster of huff values
myhuffraster <- rasterHuff(x = myhuff, mask = paris)
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.

Examples

```
data(hospital)
# Compute Reilly catchment areas from known points (hospital) on a
# grid defined by its resolution
myreilly <- reilly(knownpts = hospital, varname = "capacity",
                   typefct = "exponential", span = 1250, beta = 3,
                   resolution = 200, mask = paris, returnclass = 'sf')
# Create a raster of reilly values
myreillyraster <- rasterReilly(x = myreilly, mask = paris)
# 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, CreateGrid, CreateDistMatrix.

Examples

```
data(hospital)
# Compute Stewart potentials from known points (hospital) on a
# grid defined by its resolution
mystewart <- stewart(knownpts = hospital, varname = "capacity",
                     typefct = "exponential", span = 1000, beta = 3,
                     resolution = 100, mask = paris)
# Create a raster of potentials values
mystewartraster <- rasterStewart(x = mystewart, mask = paris)
# 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 Polygons of Potentials Contours*

Description

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

Usage

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

Arguments

x	sp or sf object; this is the set of known observations to estimate the potentials from.
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	sp or sf object; the spatial extent of this object is used to create the regularly spaced points output. (optional)
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).
returnclass	"sp" or "sf"; class of the returned object.

Details

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

Value

A polyfon object is returned ("sp" or "sf", see isopoly Value).

See Also

stewart, isopoly

Examples

```
# load data
data("hospital")
# Compute potentials
pot <- quickStewart(x = hospital,
                    var = "capacity",
                    span = 1000,
                    beta = 2, mask = paris,
                    returnclass = "sf")
# cartography
if(require("cartography")){
  breaks <- sort(c(unique(pot$min), max(pot$max)), decreasing = FALSE)
  choroLayer(x = pot,
             var = "center", breaks = breaks,
             legend.pos = "topleft",
             legend.title.txt = "Nb. of Beds")
}

# Compute a ratio of potentials
hospital$dummy <- hospital$capacity + c(rep(50, 18))
pot2 <- quickStewart(x = hospital,
                     var = "capacity",
                     var2 = "dummy",
                     span = 1000,
                     beta = 2,
                     mask = paris,
                     returnclass = "sf")
# cartography
if(require("cartography")){
  breaks <- sort(c(unique(pot2$min), max(pot2$max)), decreasing = FALSE)
  choroLayer(x = pot2,
             var = "center", breaks = breaks,
             legend.pos = "topleft",legend.values.rnd = 3,
             legend.title.txt = "Nb. of DummyBeds")
}
```

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

Description

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

Usage

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

Arguments

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

Value

Raster of catchment areas values.

See Also

huff, plotHuff.

Examples

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

---

rasterReilly *Create a Raster from a Reilly Regular Grid*

Description

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

Usage

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

Arguments

x	sp or sf object; output of the `reilly` function.
mask	sp or sf object; 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, plotReilly.

Examples

```
library(raster)
data(hospital)
# Compute Reilly catchment areas from known points (hospital) on a
# grid defined by its resolution
myreilly <- reilly(knownpts = hospital, varname = "capacity",
                   typefct = "exponential", span = 1250, beta = 3,
                   resolution = 200, mask = paris, returnclass = "sf")
# Create a raster of reilly values
myreillyraster <- rasterReilly(x = myreilly, mask = paris)
plot(myreillyraster, col = rainbow(18))
# Correspondance between raster values and reilly areas
head(unique(levels(myreillyraster)[[1]]))
```

---

rasterStewart *Create a Raster from a Stewart Regular Grid*

Description

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

Usage

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

Arguments

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

Value

Raster of potential values.

See Also

stewart, quickStewart, plotStewart, CreateGrid, CreateDistMatrix.

Examples

```
library(raster)
data(hospital)
# Compute Stewart potentials from known points (hospital) on a
# grid defined by its resolution
mystewart <- stewart(knownpts = hospital, varname = "capacity",
                     typefct = "exponential", span = 1000, beta = 3,
                     resolution = 100, mask = paris)
# Create a raster of potentials values
```

```
mystewartraster <- rasterStewart(x = mystewart, mask = paris)
plot(mystewartraster)
```

`reilly` *Reilly Catchment Areas*

Description

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

Usage

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

Arguments

`knownpts`	sp or sf object; this is the set of known observations to estimate the catchment areas from.
`unknownpts`	sp or sf object; 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 not set, 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 grid (in map units). If resolution is not set, the grid will contain around 7250 points. (optional)
mask	sp or sf object; the spatial extent of this object is used to create the regularly spaced points 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.
returnclass	"sp" or "sf"; class of the returned object.

Value

Point object 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 grid of paris extent and 200 meters
# resolution
data(hospital)
mygrid <- CreateGrid(w = hospital, resolution = 200, returnclass = "sf")
# Create a distance matrix between known points (hospital) and mygrid
mymat <- CreateDistMatrix(knownpts = hospital, unknownpts = mygrid)
# Compute Reilly catchment areas from known points (hospital) on a given
# grid (mygrid) using a given distance matrix (mymat)
myreilly2 <- reilly(knownpts = hospital, unknownpts = mygrid,
                    matdist = mymat, varname = "capacity",
                    typefct = "exponential", span = 1250,
                    beta = 3, mask = paris, returnclass = "sf")
# Compute Reilly catchment areas from known points (hospital) on a
# grid defined by its resolution
myreilly <- reilly(knownpts = hospital, varname = "capacity",
                   typefct = "exponential", span = 1250, beta = 3,
                   resolution = 200, mask = paris, returnclass = "sf")
# The function output an sf object
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,
  matdist,
  varname,
  typefct = "exponential",
  span,
  beta,
  resolution,
  mask,
  bypassctrl = FALSE,
  longlat = TRUE,
  returnclass = "sp"
)
```

Arguments

knownpts	sp or sf object; this is the set of known observations to estimate the potentials from.
unknownpts	sp or sf object; 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 grid (in map units). If resolution is not set, the grid will contain around 7250 points. (optional)
mask	sp or sf object; the spatial extent of this object is used to create the regularly spaced points 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.
returnclass	"sp" or "sf"; class of the returned object.

Value

Point object with the computed distance weighted mean in a new field named OUTPUT.

See Also

stewart.

Examples

```
# Create a grid of paris extent and 200 meters
# resolution
data(hospital)
mygrid <- CreateGrid(w = paris, resolution = 200, returnclass = "sf")
# Create a distance matrix between known points (hospital) and mygrid
mymat <- CreateDistMatrix(knownpts = hospital, unknownpts = mygrid)
# Compute  distance weighted mean from known points (hospital) on a given
# grid (mygrid) using a given distance matrix (mymat)
mysmoothy <- smoothy(knownpts = hospital, unknownpts = mygrid,
                     matdist = mymat, varname = "capacity",
                     typefct = "exponential", span = 1250,
                     beta = 3, mask = paris, returnclass = "sf")
# Compute  distance weighted mean from known points (hospital) on a
# grid defined by its resolution
mysmoothy2 <- smoothy(knownpts = hospital, varname = "capacity",
                      typefct = "exponential", span = 1250, beta = 3,
                      resolution = 200, mask = paris, returnclass = "sf")
# The two methods have the same result
identical(mysmoothy, mysmoothy2)
# Computed values
summary(mysmoothy$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.

spatMask *Paris Perimeter*

Description

A SpatialPolygonsDataFrame of the Paris perimeter.

Details

This is a deprecated dataset.

spatPts *Public Hospitals*

Description

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

Details

This is a deprecated dataset.

spatUnits *Spatial Units of Paris*

Description

A SpatialPolygonsDataFrame of the 20 spatial arrondissements of the Paris.

Details

This is a deprecated dataset.

stewart *Stewart Potentials*

Description

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

Usage

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

Arguments

knownpts	sp or sf object; this is the set of known observations to estimate the potentials from.
unknownpts	sp or sf object; 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 missing, 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 grid (in map units). If resolution is not set, the grid will contain around 7250 points. (optional)
mask	sp or sf object; the spatial extent of this object is used to create the regularly spaced points 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.
returnclass	"sp" or "sf"; class of the returned object.

Value

Point object 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, isopoly, CreateGrid, CreateDistMatrix.

Examples

```
# Create a grid of paris extent and 200 meters
# resolution
data(hospital)
mygrid <- CreateGrid(w = paris, resolution = 200, returnclass = "sf")
# Create a distance matrix between known points (spatPts) and mygrid
mymat <- CreateDistMatrix(knownpts = hospital, unknownpts = mygrid)
# Compute Stewart potentials from known points (spatPts) on a given
# grid (mygrid) using a given distance matrix (mymat)
mystewart <- stewart(knownpts = hospital, unknownpts = mygrid,
                     matdist = mymat, varname = "capacity",
                     typefct = "exponential", span = 1250,
                     beta = 3, mask = paris, returnclass = "sf")
# Compute Stewart potentials from known points (spatPts) on a
```

```
# grid defined by its resolution
mystewart2 <- stewart(knownpts = hospital, varname = "capacity",
                      typefct = "exponential", span = 1250, beta = 3,
                      resolution = 200, mask = paris, returnclass = "sf")
# The two methods have the same result
identical(mystewart, mystewart2)
# the function output a sf data.frame
class(mystewart)
# Computed values
summary(mystewart$OUTPUT)
```

Index