

# Package ‘SpatialPosition’

August 29, 2016

**Title** Spatial Position Models

**Version** 1.1.1

**Date** 2016-06-07

**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** cartography, knitr, rmarkdown

**URL** <https://github.com/Groupe-ElementR/SpatialPosition>

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

**VignetteBuilder** knitr

**Encoding** UTF-8

**RoxygenNote** 5.0.1

**NeedsCompilation** no

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

**Maintainer** Timothée Giraud <[timothee.giraud@ums-riate.fr](mailto:timothee.giraud@ums-riate.fr)>

**Repository** CRAN

**Date/Publication** 2016-06-07 14:43:42

## R topics documented:

contourStewart	2
CreateDistMatrix	4
CreateGrid	5
huff	6

plotHuff . . . . .	8
plotReilly . . . . .	8
plotStewart . . . . .	9
quickStewart . . . . .	10
rasterHuff . . . . .	12
rasterReilly . . . . .	13
rasterStewart . . . . .	14
rasterToContourPoly . . . . .	15
reilly . . . . .	16
SpatialPosition . . . . .	18
spatMask . . . . .	18
spatPts . . . . .	18
spatUnits . . . . .	19
stewart . . . . .	19

<b>Index</b>	<b>21</b>
--------------	-----------

---

contourStewart	<i>Create a SpatialPolygonsDataFrame or a SpatialLinesDataFrame from a Stewart Raster</i>
----------------	---

---

## 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 <a href="#">rasterStewart</a> 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". <b>WARNING:</b> the poly option is experimental (see details). It needs the rgeos package.

## Details

To obtain a correct SpatialPolygonsDataFrame of potentials follow these 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 output of the function is a SpatialPolygonsDataFrame (type = "poly") or a SpatialLinesDataFrame (type = "line"). The data frame of the outputted 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)
```

### 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).

**Details**

The function returns a full matrix of distances in meters using the Great Circle distance (WGS84 ellipsoid) method. This is a wrapper for the [spDists](#) function.

.

If the matrix to compute is too large (more than 100,000,000 cells or more than 10,000,000 origins or 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,
                          bypassctrl = FALSE)

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**

<code>w</code>	<code>sp</code> object; the spatial extent of this object is used to create the regular <code>SpatialPointsDataFrame</code> .
<code>resolution</code>	numeric; resolution of the grid (in map units).

**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)
```

**Arguments**

<code>knownpts</code>	sp object ( <code>SpatialPointsDataFrame</code> or <code>SpatialPolygonsDataFrame</code> ); this is the set of known observations to estimate the catchment areas from.
<code>unknownpts</code>	sp object ( <code>SpatialPointsDataFrame</code> or <code>SpatialPolygonsDataFrame</code> ); this is the set of unknown units for which the function computes the estimates. Not used when resolution is set up. (optional)
<code>matdist</code>	matrix; distance matrix between known observations and unknown units for which the function computes the estimates. Row names match the row names of <code>knownpts</code> and column names match the row names of <code>unknownpts</code> . <code>matdist</code> can contain any distance metric (time distance or euclidean distance for example). If <code>matdist</code> is <code>NULL</code> , Great Circle distances are used (with <a href="#">CreateDistMatrix</a> ). (optional)
<code>varname</code>	character; name of the variable in the <code>knownpts</code> dataframe from which values are computed. Quantitative variable with no negative values.
<code>typefct</code>	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).
mask	sp object; the spatial extent of this object is used to create the regularly spaced SpatialPointsDataFrame output. (optional)

### Details

If unknownpts is NULL then resolution must be used.

### 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)
```

---

`plotHuff`*Plot a Huff Raster*

---

**Description**

This function plots the raster produced by the [rasterHuff](#) function.

**Usage**

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

**Arguments**

<code>x</code>	raster; output of the <a href="#">rasterHuff</a> function.
<code>add</code>	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 <a href="#">rasterReilly</a> 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 <a href="#">colorRampPalette</a> .

**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 <a href="#">rasterStewart</a> 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 <a href="#">colorRampPalette</a> .

**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)
```

**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).
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.

**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 ( <code>SpatialPointsDataFrame</code> ); output of the <code>reilly</code> function.
mask	sp object ( <code>SpatialPolygonsDataFrame</code> ); 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 catchment areas values. Use `unique(levels(rasterName)[[1]])` to see the correspondance table.

**See Also**

[reilly](#), [rasterReilly](#), [plotReilly](#), [CreateGrid](#), [CreateDistMatrix](#).



```
# 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 output of the function is a SpatialPolygonsDataFrame. The data frame of the outputted 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)
```

## 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, Great Circle distances are used (with <a href="#">CreateDistMatrix</a> ). (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).
mask	sp object; the spatial extent of this object is used to create the regularly spaced SpatialPointsDataFrame output. (optional)

### 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))
```

---

SpatialPosition	<i>Spatial Position Package</i>
-----------------	---------------------------------

---

### 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").

### See Also

[stewart](#), [rasterStewart](#), [plotStewart](#), [quickStewart](#), [rasterToContourPoly](#), [huff](#), [rasterHuff](#), [plotHuff](#), [reilly](#), [rasterReilly](#), [plotReilly](#), [CreateGrid](#), [CreateDistMatrix](#).

---

spatMask	<i>Paris Perimeter</i>
----------	------------------------

---

### Description

A SpatialPolygonsDataFrame of the Paris perimeter.

---

spatPts	<i>Public Hospitals</i>
---------	-------------------------

---

### Description

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

---

spatUnits	<i>Spatial Units of Paris</i>
-----------	-------------------------------

---

**Description**

A SpatialPolygonsDataFrame of the 20 spatial arrondissements of the Paris.

---

stewart	<i>Stewart Potentials</i>
---------	---------------------------

---

**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)
```

**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, Great Circle distances are used (with <a href="#">CreateDistMatrix</a> ). (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).
mask	sp object; the spatial extent of this object is used to create the regularly spaced SpatialPointsDataFrame output. (optional)

**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

colorRampPalette, [9](#), [10](#)  
contourStewart, [2](#)  
CreateDistMatrix, [4](#), [6–11](#), [13–20](#)  
CreateGrid, [5](#), [5](#), [7–11](#), [13–15](#), [17](#), [18](#), [20](#)

huff, [6](#), [7](#), [8](#), [12](#), [13](#), [18](#)

plotHuff, [7](#), [8](#), [8](#), [13](#), [18](#)  
plotReilly, [8](#), [9](#), [13](#), [17](#), [18](#)  
plotStewart, [9](#), [11](#), [14](#), [15](#), [18](#), [20](#)

quickStewart, [10](#), [10](#), [14](#), [15](#), [18](#), [20](#)

rasterHuff, [7](#), [8](#), [12](#), [13](#), [18](#)  
rasterReilly, [8](#), [9](#), [13](#), [13](#), [17](#), [18](#)  
rasterStewart, [2](#), [9–11](#), [14](#), [15](#), [18](#), [20](#)  
rasterToContour, [2](#)  
rasterToContourPoly, [2](#), [3](#), [10](#), [11](#), [14](#), [15](#),  
[18](#), [20](#)  
ratify, [13](#)  
reilly, [9](#), [13](#), [16](#), [17](#), [18](#)

SpatialPosition, [18](#)  
SpatialPosition-package  
(SpatialPosition), [18](#)  
spatMask, [18](#)  
spatPts, [18](#)  
spatUnits, [19](#)  
spDists, [5](#)  
stewart, [10](#), [11](#), [14](#), [15](#), [18](#), [19](#)