Package 'gdistance'

July 2, 2014

Type Package

Title distances and routes on geographical grids

Version 1.1-5

Date 18-February-2014

Author Jacob van Etten

Maintainer Jacob van Etten <jacobvanetten@yahoo.com>

Depends R (>= 2.8.0), raster (>= 1.9-19), igraph (>= 0.7.0), Matrix,methods, sp

Description Calculate distances and routes on geographic grids.

License GPL (>= 2)

LazyLoad yes

NeedsCompilation no

Repository CRAN

Date/Publication 2014-02-18 22:16:51

R topics documented:

`gdistance-package` *gdistance: geographic distance calculations*

Description

Calculate distances and routes on geographic grids.

Details

Package:	gdistance
Type:	Package
Version:	1.1-1
Date:	2011-01-04
License:	GPL (>=3)
LazyLoad:	yes

Distances can be calculated following these steps.

1. Read spatial grid data into the R environment, using the raster package. Function: `raster`.

2. Construct an object of the class TransitionLayer or TransitionStack. Function: `transition`.

3. Correct diagonal connections and projection distortions. Function: `geoCorrection`.

4. Get coordinates for the starting and end points of routes.

5. Calculate distances and routes. Functions: `accCost`, `costDistance`, `commuteDistance`, `rSPDistance`, `shortestPath`, `passage`, `pathInc`.

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>

References

Ray, N. 2005. PATHMATRIX: a geographical information system tool to compute effective distances among samples. *Molecular Ecology Notes* 5, 177 - 180. http://cmpg.unibe.ch/software/pathmatrix/

McRae, B.H. 2006. Isolation by resistance. *Evolution* 60(8), 1551 - 1561.

McRae B.H., B.G. Dickson, and T. Keitt. 2008. Using circuit theory to model connectivity in ecology, evolution, and conservation. *Ecology* 89:2712-2724. http://www.circuitscape.org

See Also

raster

---

accCost *Accumulated Cost Surface*

Description

Calculates the accumulated cost surface from one or more origins.

Usage

```
accCost(x, fromCoords)
```

Arguments

x	object of class TransitionLayer
fromCoords	origin point locations (SpatialPoints, matrix or numeric class)

Details

If more than one coordinate is supplied in fromCoords, the function calculates the minimum least-cost distance from any origin point.

The function uses Dijkstra's algorithm (as implemented in the igraph package).

Value

RasterLayer

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>

References

E.W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1, 269 - 271.

See Also

shortest.paths, geoCorrection, costDistance

Examples

```
#example equivalent to that in the documentation on r.cost in GRASS
r <- raster(nrows=6, ncols=7, xmn=0, xmx=7, ymn=0, ymx=6, crs="+proj=utm +units=m")

r[] <- c(2, 2, 1, 1, 5, 5, 5,
         2, 2, 8, 8, 5, 2, 1,
         7, 1, 1, 8, 2, 2, 2,
         8, 7, 8, 8, 8, 8, 5,
         8, 8, 1, 1, 5, 3, 9,
         8, 1, 1, 2, 5, 3, 9)

T <- transition(r, function(x) 1/mean(x), 8)
# 1/mean: reciprocal to get permeability
T <- geoCorrection(T)

c1 <- c(5.5,1.5)
c2 <- c(1.5,5.5)

A <- accCost(T, c1)
plot(A)
text(A)
```

---

adjacencyFromTransition

Adjacent cells

Description

Identify pairs of cells that are adjacent.

Usage

```
adjacencyFromTransition(x)
```

Arguments

x	TransitionLayer

Details

Extracts the indices of those cells that are connected (e.g. cells i,j that have a non-zero value in the transition matrix).

Cell numbers are unique indices of cells in the original grid. By convention, cell numbers start with 1 in the upper-left corner of the grid and increase from left to right and from top to bottom.

Value

A two column matrix with each row containing a pair of adjacent cells.

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>

See Also

adjacent

Examples

```
r <- raster(nrows=6, ncols=7, xmn=0, xmx=7, ymn=0, ymx=6, crs="+proj=utm +units=m")
r[] <- runif(6*7)
T <- transition(r, function(x) 1/mean(x), 8)
adjacencyFromTransition(T)
```

ArithMath-methods *Arithmetic and mathematical operations with objects of Transition* classes*

Description

Standard arithmetic operators for computations with `Transition*` objects and numeric values. Transition objects must have the same extent and resolution. All arithmetic and mathematical operations that work on the sparse matrices are available for `Transition*` objects.

Value

`Transition*` object or numeric.

Author(s)

Jacob van Etten

Examples

```
#create a new raster and set all its values to unity.
raster <- raster(nrows=18, ncols=36)
raster <- setValues(raster,rep(1,ncell(raster)))

#create TransitionLayer objects
tr1 <- transition(raster,mean,4)
tr2 <- tr1

#arithmetic operations
tr3 <- tr1 * tr2
tr4 <- tr3 * 4
```

```
#mathematical operations
tr5 <- sqrt(tr4)
```

`commuteDistance` *Commute-time distance*

Description

Calculates the resistance distance between points.

Usage

```
commuteDistance (x, coords)
```

Arguments

`x`	`TransitionLayer` object)
`coords`	point locations coordinates (of SpatialPoints, matrix or numeric class)

Details

This function calculates the expected random-walk commute time between nodes in a graph. It is defined as the effective distance (resistance distance) between the selected nodes multiplied by the volume of the graph, which is the sum of the conductance weights of all the edges in the graph (Chandra et al. 1997). The result represents the average number of steps that is needed to commute between the nodes during a random walk.

The function implements the algorithm given by Fouss et al. (2007).

Before calculating commute-time distances from a `TransitionLayer` object, see if you need to apply the function `geoCorrection`.

Value

distance matrix (S3 class dist or matrix)

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>

References

Chandra, A.K., Raghavan, P., Ruzzo, W.L., Smolensy, R. & Tiwari, P. 1996. The electrical resistance of a graph captures its commute and cover times. *Computational Complexity*, 6(4), 312-340.

Fouss, F., Pirotte, A., Renders, J.-M. & Saerens, M. 2007. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 19(3), 355-369.

McRae, B.H. 2006. Isolation by resistance. *Evolution* 60(8), 1551-1561.

http://www.circuitscape.org

See Also

geoCorrection

Examples

```
#Create a new raster and set all its values to unity.
r <- raster(nrows=18, ncols=36)
r <- setValues(r,rep(1,ncell(raster)))

#Create a Transition object from the raster
tr <- transition(r,function(x) 1/mean(x),4)

#Create two sets of coordinates
sP1 <- SpatialPoints(cbind(c(65,5,-65),c(55,35,-35)))
sP2 <- SpatialPoints(cbind(c(50,15,-40),c(80,20,-5)))

#Calculate the resistance distance between the points
commuteDistance(tr,sP1)
```

Coords class *Coords class*

Description

This is a class union, providing a convenient class for coordinates in several formats. The class accepts coordinates in any of the following formats: 1. SpatialPoints 2. two-columned matrix 3. vector of length 2

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>

Examples

```
showClass("Coords")
```

costDistance *Cost distance (least-cost distance)*

Description

Calculate the least-cost distance between points.

Usage

```
costDistance(x, fromCoords, toCoords)
```

Arguments

x	object of class TransitionLayer
fromCoords	first set of point locations (of SpatialPoints, matrix or numeric class)
toCoords	second, optional set of point locations (of SpatialPoints, matrix or numeric class)

Details

Cost units between cells are defined as the reciprocal of the values in the transition matrix.

The function uses Dijkstra's algorithm, as implemented in the igraph package.

A projection correction is needed for accuracy in the case of grid data for a longlat raster (see function geoCorrection).

Value

distance matrix (S3 class dist or matrix)

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>

References

E.W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1, 269-271.

See Also

accCost, geoCorrection

Examples

```
#create a new raster and set all its values to unity.
r <- raster(nrows=18, ncols=36)
r <- setValues(r,runif(ncell(r),0,1))

#create a Transition object from the raster
tr <- transition(r,function(x) 1/mean(x),8)

#asymmetric
ncf <- function(x) max(x) - x[1] + x[2]
tr2 <- transition(r,ncf,8, symm=FALSE)

#create two sets of coordinates
sP1 <- cbind(c(65,5,-65),c(55,35,-35))
sP2 <- cbind(c(50,15,-40),c(80,20,-5))

#from and to identical
costDistance(tr,sP1)
costDistance(tr2,sP1)

#from and to different
costDistance(tr,sP1,sP2)
costDistance(tr2,sP1,sP2)
```

---

genDist *Genetic distances and coordinates of haplogroup R1b1b2 populations in Europe*

Description

genDist: Genetic distances between 23 populations of Y-chromosome haplogroup R1b1b2 in Europe, calculated from haplotype microsatellite data (see source). The distance measure used is the negative logarithm of the shared proportion of alleles.

popCoord: Geographic coordinates of the same populations.

Usage

```
genDist
```

Format

dist object (genDist) and dataframe (popCoord)

Source

Distances calculated from Table S1 in Balaresque et al. (2010). Coordinates from Table 1.

References

Balaresque P., et al. 2010. A predominantly Neolithic origin for European paternal lineages. *PLoS Biology* 8(1): e1000285.

`geoCorrection` *Geographic Correction*

Description

Correct Transition* objects taking into account local distances

Usage

```
geoCorrection(x, type, ...)
```

Arguments

`x`	object of class Transition*
`type`	type of correction: "c", "r", or missing (only required for lonlat, see below)
`...`	multpl: matrix with correction factor (TRUE) or corrected values (FALSE, the default); scl: scale the correction values (default is FALSE)

Details

Geographic correction is necessary for all objects of the class Transition that are either: (1) based on a grid in a geographic (lonlat) projection and covering a large area; (2) made with directions > 4.

The function will correct for map distortion, as well as for diagonal connections between grid cells (which cover a longer distance than vertical or horizontal connections).

When working with lonlat grids, users should also anticipate whether they will use methods based on either least-cost or random walks, and set the `type` argument accordingly. In the case of least-cost distances, the correction is only done in East-West direction. In the case of random walks there should be an additional correction which reduces the conductance in North-South direction (`type="r"`).

The correction is done by dividing conductance values by the inter-cell distance. Distances are calculated as great-circle distances for lonlat grids (see function isLonLat()) and Euclidean distances for all other grids.

In the case of randomised shortest paths, the need for correction is somewhat in between these two correction methods. We have not developed an analytical solution for this problem yet. With very low values for theta, you may choose the correction for random walks, and for high values the one for least-cost paths. Users who want to work with intermediate values of theta are encouraged to experiment with different solutions.

The values are scaled to get values near 1 if the argument `scl` is set to TRUE. This is desirable for subsequent calculations involving random walk calculations. Values are divided by the W-E inter-cell distance (at the centre of the grid).

Value

Transition* object

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>

Examples

```
r <- raster(ncol=36,nrow=18)
r <- setValues(r,rep(1,times=ncell(r)))
tr <- transition(r, mean, directions=8)

#directly
tr1 <- geoCorrection(tr, type="c", multpl=FALSE)

#the same, but with a separate correction matrix
trCorr <- geoCorrection(tr, type="c", multpl=TRUE)
tr2 <- tr * trCorr
```

---

normalize *normalize*

Description

Normalize the transition matrix.

Usage

```
normalize(x, ...)
```

Arguments

x	object of class Transition*)
...	optional argument method (see below)

Details

Normalization of the weighted adjacency matrix in the Transition* object. Matrix values are divided by their respective row-sums, column-sums, or the product of the square-roots of both (symmetric normalization). The default method is row-normalization. To use the other normalization methods, users can set the optional method argument to either "col" or "symm". For random walk calculations a symmetric matrix is needed (method = "symm").

Value

TransitionLayer object

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>

References

von Luxburg, U. 2007. A tutorial on spectral clustering. *Statistics and Computing* 17(4), 395-416. http://arxiv.org/PS_cache/arxiv/pdf/0711/0711.0189v1.pdf

Chung, F. 1997. *Spectral Graph Theory*. Conference Board of the Mathematical Sciences, Washington.

Examples

```
r <- raster(ncol=36,nrow=18)
r <- setValues(r,rep(1,times=ncell(r)))
tr <- transition(r, mean, directions=8)

tr1 <- normalize(tr)
tr2 <- normalize(tr, method="symm")
```

---

overlap *Overlap and nonoverlap of trajectories*

Description

Special functions to calculate the degree of overlap and nonoverlap between trajectories

Usage

```
overlap(a, b)
nonoverlap(a, b)
```

Arguments

a	matrix object
b	matrix object

Details

These functions are used by the pathInc() as defaults.

Value

matrix

Note

The functions are provided here to assist the user in defining alternative measures of overlap / nonoverlap.

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>

`passage` *Probability of passage*

Description

Calculates for each cell the number of passages of a random-walk or randomised shortest paths with given origin(s) and destination(s). Either the total or the net number of passages can be calculated. In the case of multiple origins or destinations, each receives equal weight.

Usage

```
passage(x, origin, goal, theta, ...)
```

Arguments

`x`	Object of class Transition
`origin`	SpatialPoints, matrix or numeric object with coordinates or RasterLayer object with origin cells set to TRUE
`goal`	SpatialPoints, matrix or numeric object with coordinates or RasterLayer object with origin cells set to TRUE
`theta`	If zero or missing, a random walk results. If a numeric value 0 < theta < 20 is given, randomised shortest paths are calculated; theta is the degree from which the path randomly deviates from the shortest path
`...`	Additional arguments: totalNet ("total" or "net"), and output ("RasterLayer" or "Transition")

Details

The net number of passages between i and j is defined as: abs(passages from i to j - passages from j to i).

Defaults for additional argument `totalNet` is "net" and for `output` it is "RasterLayer".

Random walk requires a symmetric transition matrix.

Value

RasterLayer or Transition object, depending on the output argument.

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>. Implementation of randomised shortest paths based on Matlab code by Marco Saerens.

References

McRae B.H., B.G. Dickson, and T. Keitt. 2008. Using circuit theory to model connectivity in ecology, evolution, and conservation. *Ecology* 89:2712-2724.

Saerens M., L. Yen, F. Fouss, and Y. Achbany. 2009. Randomized shortest-path problems: two related models. *Neural Computation*, 21(8):2363-2404.

See Also

commuteDistance, pathInc

Examples

```
#create a new raster and set all its values to unity.
raster <- raster(nrows=18, ncols=36)
raster <- setValues(raster,rep(1,ncell(raster)))

#create a Transition object from the raster
tr <- transition(raster,mean,4)
trC <- geoCorrection(tr, type="c", scl=TRUE)
trR <- geoCorrection(tr, type="r", scl=TRUE)

#create two coordinates
sP1 <- SpatialPoints(cbind(-105,55))
sP2 <- SpatialPoints(cbind(105,-55))

#randomised shortest paths with theta = 2
rSPraster <- passage(trC, sP1, sP2, 2)
plot(rSPraster)
points(sP1)
points(sP2)

#randomised shortest paths with theta = 0.05
rSPraster <- passage(trC, sP1, sP2, 0.05)
plot(rSPraster)
points(sP1)
points(sP2)

#randomised shortest paths with theta = 0.05
#and total
rSPraster <- passage(trC, sP1, sP2, 0.05, totalNet = "total")
plot(rSPraster)
points(sP1)
points(sP2)

#random walk
rwraster <- passage(trR, sP1, sP2)
plot(rwraster)
points(sP1)
points(sP2)
```

pathInc *Incidence of paths from a common origin: overlap and non-overlap*

Description

Calculate the overlap and non-overlap of paths departing from a common origin. Two algorithms are available: random walk and randomised shortest paths.

Usage

```
pathInc(x, origin, from, to, theta, weight, ...)
```

Arguments

x	transition matrix (class Transition)
origin	coordinates of the origin (one point location, SpatialPoints, matrix or numeric class)
from	coordinates of the destinations (SpatialPoints, matrix or numeric class)
to	second set of coordinates of the destinations (can be missing)
theta	value > 0 and < 20 (randomised shortest paths) or missing (random walk)
weight	weight matrix – Reciprocals of the non-zero values are used as weights. If missing, reciprocals of the transition matrix are used.
...	an extra argument for functions to be defined manually (see below)

Details

This is a complex wrapper function that calculates to what extent dispersal routes overlap or do not overlap.

First, the function calculates the trajectories for all "from" and "to" locations, starting from a single "origin" location. These trajectories can either be based on random walks or randomised shortest paths (giving a value to theta).

Then, for all unique pairs of trajectories, it calculates the extent to which these trajectories overlap or diverge. These values are given back to the user as a list of (distance) matrices.

If only "from" coordinates are given, the function calculates symmetric distance matrices for all combinations of points in "from". If both "from" and "to" coordinates are given, the function calculates a matrix of values with rows for all locations in "from" and columns for all locations in "to".

Overlap is currently calculated as the minimum values of each pair of trajectories compared. Non-overlap uses the following formula: Nonoverlap = max(0,max(a,b)*(1-min(a,b))-min(a,b)) (see van Etten and Hijmans 2010).

Value

list of dist objects or list of matrices

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>. Implementation of randomised shortest paths based on Matlab code by Marco Saerens.

References

McRae B.H., B.G. Dickson, and T. Keitt. 2008. Using circuit theory to model connectivity in ecology, evolution, and conservation. *Ecology* 89:2712-2724.

Saerens M., L. Yen, F. Fouss, and Y. Achbany. 2009. Randomized shortest-path problems: two related models. *Neural Computation*, 21(8):2363-2404.

van Etten, J., and R.J. Hijmans. 2010. A geospatial modelling approach integrating archaeobotany and genetics to trace the origin and dispersal of domesticated plants. *PLoS ONE* 5(8): e12060.

Examples

```
#Create TransitionLayer
r <- raster(ncol=36,nrow=18)
r <- setValues(r,rep(1,times=ncell(r)))
tr <- transition(r,mean,directions=4)

#Two different types of correction are required
trR <- geoCorrection(tr, type="r", multpl=FALSE)
trC <- geoCorrection(tr, type="c", multpl=FALSE)

#Create TransitionStack
ts <- stack(trR, trR)

#Points for origin and coordinates between which to calculate path (non)overlaps
sP0 <- SpatialPoints(cbind(0,0))
sP1 <- SpatialPoints(cbind(c(65,5,-65),c(-55,35,-35)))

#Randomised shortest paths
#rescaling is needed: exp(-theta * trC) should give reasonable values
trC <- trC / median(transitionMatrix(trC)@x) #divide by median of the non-zero values
pathInc(trC, origin=sP0, from=sP1, theta=2)

#Random walk
pathInc(trR, origin=sP0, from=sP1)

#TransitionStack as weights
pathInc(trR, origin=sP0, from=sP1, weight=ts)

#Demonstrate use of an alternative function
#The current default is to take the minimum of each pair of layers

altoverlap <- function(a, b)
{
aV <- as.vector(a[,rep(1:ncol(a), each=ncol(b))])
bV <- as.vector(b[,rep(1:ncol(b), times=ncol(a))])
result <- matrix(aV * bV, nrow = nrow(a), ncol=ncol(a)*ncol(b))
return(result)
```

```
}

pathInc(trR, origin=sP0, from=sP1, weight=ts, functions=list(altoverlap))
```

---

raster-methods *RasterLayer from TransitionLayer object*

Description

Create a RasterLayer from a TransitionLayer with a call to the generic function raster.

The n x n transition matrix of the TransitionLayer is transformed to form the values n cells of a raster.

The following methods to 'reduce' the transition matrix are available with the optional argument reduceMethod):

- colSums
- rowSums
- colMeans
- rowMeans
- NZcolMeans
- NZrowMeans

The latter two methods only take into account the non-zero entries in the transition matrix.

The default is NZcolMeans.

Value

RasterLayer

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>

Examples

```
#create a new raster and set all its values to unity.
r <- raster(nrows=18, ncols=36)
r <- setValues(r,runif(ncell(r),0,1))

#create a Transition object from the raster
tr1 <- transition(r,mean,8)

#asymmetric
asf <- function(x) max(x) - x[1] + x[2]
tr2 <- transition(r,asf,8, symm=FALSE)
```

```
#create RasterLayer objects
r1 <- raster(tr1)
r2 <- raster(tr2)
r3 <- raster(tr1, "colMeans")
```

`rSPDistance` *Randomized shortest path distance*

Description

Calculates the randomized shortest path distance between points.

Usage

```
rSPDistance (x, from, to, theta, totalNet = "net", method=1)
```

Arguments

`x`	`TransitionLayer` object)
`from`	point locations coordinates (of SpatialPoints, matrix or numeric class)
`to`	point locations coordinates (of SpatialPoints, matrix or numeric class)
`theta`	theta is the degree from which the path randomly deviates from the shortest path, 0 < theta < 20
`totalNet`	total or net movements between cells
`method`	method 1 (as defined in Saerens et al.) or method 2 (a modified version, see below in Details)

Details

The function implements the algorithm given by Saerens et al. (2009).

Method 1 implements the method as it is. Method 2 uses W = exp(-theta * ln(P)).

Value

distance matrix (S3 class dist or matrix)

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>

References

Saerens M., L. Yen, F. Fouss, and Y. Achbany. 2009. Randomized shortest-path problems: two related models. *Neural Computation*, 21(8):2363-2404.

See Also

geoCorrection

Examples

```
#Create a new raster and set all its values to unity.
r <- raster(nrows=18, ncols=36)
r <- setValues(r,rep(1,ncell(raster)))

#Create a Transition object from the raster
tr <- transition(r,mean,4)

#Create two sets of coordinates
sP1 <- SpatialPoints(cbind(c(65,5,-65),c(55,35,-35)))
sP2 <- SpatialPoints(cbind(c(50,15,-40),c(80,20,-5)))

#Calculate the RSP distance between the points
rSPDistance(tr, sP1, sP2, 1)
```

---

shortestPath *Shortest path*

Description

Calculates the shortest path from an origin to a goal.

Usage

```
shortestPath(x, origin, goal, ...)
```

Arguments

x	TransitionLayer object
origin	SpatialPoints, vector or matrix with coordinates, at the moment only the first cell is taken into account
goal	SpatialPoints, vector or matrix with coordinates
...	Additional argument: output

Details

As an additional argument output, the desired output object can be specified: either "TransitionLayer" (default), "TransitionStack" or "SpatialLines".

If there is more than one path either (1) transition values in the TransitionLayer get values of 1 / number of paths or (2) the SpatialLines object will contain more than one line.

Value

Transition object.

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>.

See Also

costDistance, accCost

Examples

```
#example equivalent to that in the documentation on r.cost/r.drain in GRASS
r <- raster(nrows=6, ncols=7, xmn=0, xmx=7, ymn=0, ymx=6, crs="+proj=utm +units=m")

r[] <- c(2, 2, 1, 1, 5, 5, 5,
         2, 2, 8, 8, 5, 2, 1,
         7, 1, 1, 8, 2, 2, 2,
         8, 7, 8, 8, 8, 8, 5,
         8, 8, 1, 1, 5, 3, 9,
         8, 1, 1, 2, 5, 3, 9)

T <- transition(r, function(x) 1/mean(x), 8)
# 1/mean: reciprocal to get permeability
T <- geoCorrection(T)

c1 <- c(5.5,1.5)
c2 <- c(1.5,5.5)

#make a SpatialLines object for visualization
sPath1 <- shortestPath(T, c1, c2, output="SpatialLines")
plot(r)
lines(sPath1)

#make a TransitionLayer for further calculations
sPath2 <- shortestPath(T, c1, c2)
plot(raster(sPath2))
```

Summary-methods *Summary methods*

Description

The following summary methods are available:

mean, Median, max, min, range, prod, sum, any, all

Value

a TransitionLayer

Note

These methods compute a summary statistic based on cell values of layers in a TransitionStack. The result of these methods is always a single TransitionLayer.

Author(s)

Jacob van Etten

Examples

```
#Create a new raster and set all its values to unity.
raster <- raster(nrows=18, ncols=36)
raster <- setValues(raster,rep(1,ncell(raster)))

#Create a Transition object from the raster
tr <- transition(raster,mean,4)

trS <- stack(tr, tr*2)

#Apply a Summary method
trSum <- sum(trS)

#plot(raster(trMean))
```

`sumReciprocal` *Reciprocal of the sum of the reciprocals of conductance values in Transition* objects*

Description

Reciprocal of the sum of the reciprocals of conductance Transition* objects

Usage

```
sumReciprocal(x1, x2)
```

Arguments

`x1`	`TransitionLayer` object
`x2`	`TransitionLayer` object

Details

To calculate the total resistance of two resistors that are serially connected, we should add their resistance values. However, if we work with conductance values, we need to take the reciprocal of the summed reciprocals of the conductance values. This function does that when adding two TransitionLayers with conductance values (`matrixValues(tr) == "conductance"`).

For a TransitionLayer with resistance values (`matrixValues(tr) == "resistance"`), the function will not take reciprocals for that object, but will still take a reciprocal for the final product (which will consequently have conductance values).

Value

TransitionLayer object containing conductance values.

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>

Examples

```
#Create a new raster and set all its values to unity.
raster <- raster(nrows=18, ncols=36)
raster <- setValues(raster,rep(1,ncell(raster)))

#Create TransitionLayer objects
tr1 <- transition(raster,mean,4)
tr2 <- tr1
matrixValues(tr1)

#Set one to resistance
matrixValues(tr2) <- "resistance"

#Sum the two objects
sumReciprocal(tr1,tr2)
```

---

transition *Create an object of the class Transition*

Description

Create a Transition object from a RasterLayer or RasterBrick object. Transition values are calculated with a user-defined function from the grid values.

Usage

```
transition (x, transitionFunction, directions, ...)
```

Arguments

x	RasterLayer or RasterBrick (raster package)
transitionFunction	Function to calculate transition values from grid values
directions	Directions in which cells are connected (4, 8, 16, or other), see adjacent
...	additional arguments, see methods 1 and 3 below

Details

Users may use one of three methods to construct a Transition* object with this function.

1) TransitionLayer from RasterLayer

```
transition(x, transisitonFunction, directions, symm)
```

When a symmetric transition matrix is required, the user should supply a transitionFunction f that obeys f(i,j) = f(j,i) (a commutative function). The function transition does no commutativity check. To obtain an asymmetric transition matrix, a non-commutative function should be supplied and an additional argument 'symm' should be set to FALSE.

2) TransitionLayer from RasterBrick

```
transition(x, transitionFunction="mahal", directions)
```

This method serves to summarize several layers of data in a single distance measure. The distance between adjacent cells is the normalized reciprocal of the Mahalanobis distance (mean distance / (mean distance + distance ij).

3) TransitionStack from RasterLayer

In contrast with the above methods, this method produces resistance matrices by default.

a) Continuous variables - barriers

```
transition(x, transitionFunction="barriers", directions, symm, intervalBreaks)
```

This method creates a TransitionStack with each layer containing a discrete boundary between areas in x. Areas are defined by intervals in x. The argument intervalBreaks is a vector of interval breaks corresponding to the values in x. If between a pair of cells i and j, min(i,j) < break AND max(i,j) > break, then the value ij in the transition matrix becomes 1. All other values in the transition matrix remain 0. The package classInt offers several methods to define intervals. If symm is changed from the default (TRUE) to "up" or "down", it will give either only the upslope (symm="up") or downslope (symm="down") barriers.

b) Categorical variables - barriers

```
transition(x, transitionFunction="barriers", directions)
```

In this case, areas are defined as categories in the input raster. A raster with a categorical variable can be created with asFactor(). The layers of the resulting TransitionStack contain all possible combinations of categories. Which layer contains the combination of categories i and j out of n categories, can be determined with these formulae:

if symm is TRUE: layer(i,j) = n*(j-1) - j*(j-1)/2 + i-j. if symm is FALSE and i>j: layer(i,j) = ((n*(j-1) - j*(j-1)/2 + i-j) * 2) - 1. if symm is FALSE and i<j: layer(i,j) = (n*(j-1) - j*(j-1)/2 + i-j) * 2.

c) Categorical variables - areas

```
transition(x, transitionFunction="areas", directions)
```

Here, areas are also a categorical variable (see under 3b). The layers in the resulting TransitionStack represent each one area. Connections between two cells which are each inside the area are set to 1. Connections between a cell inside and a cell outside the area are set to 0.5. Connections between two cells outside the area are set to 0.

Value

Transition object

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>

Examples

```
#Create a new raster and set all its values to unity.
r <- raster(nrows=18, ncols=36)
r <- setValues(r, runif(ncell(r)))

#Create a Transition object from the raster
tr <- transition(r, transitionFunction=mean, directions=4)
tr #the sparse matrix is of class dsCMatrix (symmetric)

#Create an asymmetric transition matrix
#first, an arbitrary non-commutative function
f <- function(x) max(x) - x[1] + x[2]
tr2 <- transition(r, f, 4, symm=FALSE)
tr2 #the sparse matrix is of class dgCMatrix (=asymmetric)

#Barriers - interval breaks
tr3 <- transition(r, "barriers", 8, intervalBreaks=c(0.25, 0.5, 0.75))
nlayers(tr3)

#Barriers - categories
r <- round(r * 2)
r <- asFactor(r)
tr4 <- transition(r, "barriers", 8)
nlayers(tr4)
plot(raster(tr4[[2]]))

#Areas
r <- round(r * 2)
r <- asFactor(r)
tr5 <- transition(r, "areas", 8)
nlayers(tr5)
plot(raster(tr5[[2]]))
```

Transition slots *Extract or change elements of Transition* objects*

Description

These functions are to be used to access slots of Transition* objects.

Usage

```
transitionMatrix(x, inflate)
transitionCells(x)
matrixValues(x)
```

Arguments

x	object of class Transition*
inflate	logical (default is TRUE)

Transition* classes *Transition* classes*

Description

TransitionLayer and TransitionStack (or Transition*) are the core classes of the package gdistance. They are the main input into the functions to calculate distances and routes.

An object of the class TransitionLayer contains two main elements: a. a transition matrix with transition values between connected cells in a raster - an object of class sparseMatrix (package Matrix); b. information on the extent, resolution and projection of the underlying raster - an object of class Raster (package raster).

All slots belong to these two elements from other package, except two additional slots: 1. slot transitionCells, which is only used internally in the package; 2. slot matrixValues indicates if the nonzero values of the transition matrix contains conductance or resistance values.

Class TransitionStack contains one or more transition matrices.

Class Transition is the union of TransitionLayer and TransitionStack.

Objects from the Class

Objects can be created by calls of the form `new("Transition", nrows, ncols, xmin, xmax, ymin, ymax, projection)`

Slots

`transitionMatrix`: Object of class `"sparseMatrix"`

`transitionCells`: Object of class `"integer"`

`matrixValues`: Object of class `"character"`

`ncols`: Object of class `"integer"`

`nrows`: Object of class `"integer"`

`crs`: Object of class `"CRS"` (sp package)

`extent`: Object of class `"Extent"`

`layernames`: Object of class `"vector"`

Extends

Class `"Raster"`.

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>

Examples

```
showClass("TransitionLayer")

tr <- new("TransitionLayer",nrows=as.integer(36),ncols=as.integer(18),extent=extent(c(xmin=-180,xmax=180,
          ymin=-90,ymax=90)),crs=CRS("+proj=longlat +datum=WGS84"))

tr <- new("TransitionLayer",nrows=as.integer(36),ncols=as.integer(18),extent=extent(c(xmin=-180,xmax=180,
          ymin=-90,ymax=90)),crs=CRS(""))
```

Transition* methods *Extracting and replacing: class Transition*

Description

Methods for functions `[` and `[<-` for object of the class TransitionLayer. Methods for functions `[[` and `[[<-` for object of the class TransitionStack.

Also see `adjacencyFromTransition`.

Examples

```
#Create a new raster and set all its values to unity.
r <- raster(nrows=18, ncols=36)
r <- setValues(r,rep(1,ncell(r)))

#Create TransitionLayer objects
tr1 <- transition(r,mean,4)
tr2 <- tr1
```

```
#Extracting and replacing
tr1[cbind(1:9,1:9)] <- tr2[cbind(1:9,1:9)]
tr1[1:9,1:9] <- tr2[1:9,1:9]
tr1[1:5,1:5]
```

Index