Package ‘spdep’

January 29, 2022

Version 1.2-2
Date 2022-01-28
Title Spatial Dependence: Weighting Schemes, Statistics
Encoding UTF-8
Depends R (>= 3.3.0), methods, sp (>= 1.0), spData (>= 0.2.6.0), sf
Imports stats, deldir, boot (>= 1.3-1), graphics, utils, grDevices, units, s2, e1071
Suggests spatialreg (>= 1.2-1), Matrix, parallel, dbscan, rgeos, RColorBrewer, lattice, xtable, maptools (>= 0.5-4), foreign, igraph, RSpectra, knitr, rgdal, classInt, tmap, spam, ggplot2, markdown, tinytest, rgeoda
BugReports https://github.com/r-spatial/spdep/issues/
for global and local 'Moran's I' ('Bivand et al.' 2009)
<doi:10.1016/j.csda.2008.07.021> and 'LOSH' local indicators
of spatial heteroscedasticity ('Ord' and 'Getis')
<doi:10.1007/s00168-011-0492-y>. The implementation of most of
the measures is described in 'Bivand' and 'Wong' (2018)
<doi:10.1007/s11749-018-0599-x>.
From 'spdep' and 'spatialreg' versions >= 1.2-1, the model fitting functions
previously present in this package are defunct in 'spdep' and may be found
in 'spatialreg'.

License GPL (>= 2)

VignetteBuilder knitr

RoxygenNote RoxygenNote: 6.1.1

NeedsCompilation yes

Author Roger Bivand [cre, aut] (<https://orcid.org/0000-0003-2392-6140>),
Micah Altman [ctb],
Luc Anselin [ctb],
Renato Assunção [ctb],
Olaf Berke [ctb],
Guillaume Blanchet [ctb],
Marília Carvalho [ctb],
Bjarke Christensen [ctb],
Yongwan Chun [ctb],
Carsten Dormann [ctb],
Stéphane Dray [ctb],
Virgilio Gómez-Rubio [ctb],
Elias Krainski [ctb],
Pierre Legendre [ctb],
Nicholas Lewin-Koh [ctb],
Angela Li [ctb],
Giovanni Millo [ctb],
Werner Mueller [ctb],
Hisaji Ono [ctb],
Josiah Parry [ctb] (<https://orcid.org/0000-0001-9910-865X>),
Pedro Peres-Neto [ctb],
Gianfranco Piras [ctb],
Markus Reder [ctb],
Jeff Sauer [ctb],
Michael Tiefelsdorf [ctb],
René Westerholt [ctb],
Levi Wolf [ctb],
Danlin Yu [ctb]

Maintainer Roger Bivand <Roger.Bivand@nhh.no>

Repository CRAN

Date/Publication 2022-01-28 23:10:02 UTC
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregate.nb</td>
<td>5</td>
</tr>
<tr>
<td>airdist</td>
<td>6</td>
</tr>
<tr>
<td>autocov_dist</td>
<td>6</td>
</tr>
<tr>
<td>bhicv</td>
<td>8</td>
</tr>
<tr>
<td>card</td>
<td>9</td>
</tr>
<tr>
<td>cell2nb</td>
<td>10</td>
</tr>
<tr>
<td>choynowski</td>
<td>12</td>
</tr>
<tr>
<td>columbus</td>
<td>13</td>
</tr>
<tr>
<td>diffnb</td>
<td>14</td>
</tr>
<tr>
<td>dnearneigh</td>
<td>15</td>
</tr>
<tr>
<td>droplinks</td>
<td>17</td>
</tr>
<tr>
<td>EBest</td>
<td>19</td>
</tr>
<tr>
<td>EBImoran.mc</td>
<td>20</td>
</tr>
<tr>
<td>EBllocal</td>
<td>22</td>
</tr>
<tr>
<td>edit.nb</td>
<td>24</td>
</tr>
<tr>
<td>eire</td>
<td>25</td>
</tr>
<tr>
<td>geary</td>
<td>25</td>
</tr>
<tr>
<td>geary.mc</td>
<td>26</td>
</tr>
<tr>
<td>geary.test</td>
<td>28</td>
</tr>
<tr>
<td>globalG.test</td>
<td>30</td>
</tr>
<tr>
<td>Graph Components</td>
<td>32</td>
</tr>
<tr>
<td>graphneigh</td>
<td>33</td>
</tr>
<tr>
<td>grid2nb</td>
<td>36</td>
</tr>
<tr>
<td>include.self</td>
<td>37</td>
</tr>
<tr>
<td>is.symmetric.nb</td>
<td>38</td>
</tr>
<tr>
<td>joincount.mc</td>
<td>40</td>
</tr>
<tr>
<td>joincount.multi</td>
<td>41</td>
</tr>
<tr>
<td>joincount.test</td>
<td>43</td>
</tr>
<tr>
<td>knearneigh</td>
<td>45</td>
</tr>
<tr>
<td>knn2nb</td>
<td>47</td>
</tr>
<tr>
<td>lag.listw</td>
<td>48</td>
</tr>
<tr>
<td>lee</td>
<td>49</td>
</tr>
<tr>
<td>lee.mc</td>
<td>50</td>
</tr>
<tr>
<td>lee.test</td>
<td>52</td>
</tr>
<tr>
<td>listw2sn</td>
<td>54</td>
</tr>
<tr>
<td>lm.LMtests</td>
<td>55</td>
</tr>
<tr>
<td>lm.morantest</td>
<td>57</td>
</tr>
<tr>
<td>lm.morantest.exact</td>
<td>59</td>
</tr>
<tr>
<td>lm.morantest.sad</td>
<td>61</td>
</tr>
<tr>
<td>localC</td>
<td>63</td>
</tr>
<tr>
<td>localG</td>
<td>71</td>
</tr>
<tr>
<td>localmoran</td>
<td>73</td>
</tr>
<tr>
<td>localmoran.exact</td>
<td>76</td>
</tr>
<tr>
<td>localmoran.sad</td>
<td>79</td>
</tr>
<tr>
<td>LOSH</td>
<td>82</td>
</tr>
<tr>
<td>LOSH.cs</td>
<td>84</td>
</tr>
</tbody>
</table>
### R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOSH.mc</td>
<td>85</td>
</tr>
<tr>
<td>mat2listw</td>
<td>87</td>
</tr>
<tr>
<td>moran</td>
<td>88</td>
</tr>
<tr>
<td>moran.mc</td>
<td>90</td>
</tr>
<tr>
<td>moran.plot</td>
<td>91</td>
</tr>
<tr>
<td>moran.test</td>
<td>93</td>
</tr>
<tr>
<td>mstree</td>
<td>95</td>
</tr>
<tr>
<td>nb.set.operations</td>
<td>97</td>
</tr>
<tr>
<td>nb2blocknb</td>
<td>98</td>
</tr>
<tr>
<td>nb2INLA</td>
<td>100</td>
</tr>
<tr>
<td>nb2lines</td>
<td>101</td>
</tr>
<tr>
<td>nb2listw</td>
<td>102</td>
</tr>
<tr>
<td>nb2listwdist</td>
<td>104</td>
</tr>
<tr>
<td>nb2mat</td>
<td>107</td>
</tr>
<tr>
<td>nb2WB</td>
<td>108</td>
</tr>
<tr>
<td>nbcosts</td>
<td>109</td>
</tr>
<tr>
<td>nbdists</td>
<td>110</td>
</tr>
<tr>
<td>nblag</td>
<td>111</td>
</tr>
<tr>
<td>oldcol</td>
<td>113</td>
</tr>
<tr>
<td>p.adjustSP</td>
<td>114</td>
</tr>
<tr>
<td>plot.mst</td>
<td>115</td>
</tr>
<tr>
<td>plot.nb</td>
<td>116</td>
</tr>
<tr>
<td>plot.skater</td>
<td>117</td>
</tr>
<tr>
<td>poly2nb</td>
<td>118</td>
</tr>
<tr>
<td>probmap</td>
<td>120</td>
</tr>
<tr>
<td>prunecost</td>
<td>121</td>
</tr>
<tr>
<td>prunemst</td>
<td>123</td>
</tr>
<tr>
<td>read.gal</td>
<td>124</td>
</tr>
<tr>
<td>read.gwt2nb</td>
<td>125</td>
</tr>
<tr>
<td>Rotation</td>
<td>127</td>
</tr>
<tr>
<td>set.mcOption</td>
<td>128</td>
</tr>
<tr>
<td>set.spChkOption</td>
<td>130</td>
</tr>
<tr>
<td>skater</td>
<td>131</td>
</tr>
<tr>
<td>sp.correlogram</td>
<td>135</td>
</tr>
<tr>
<td>sp.mantel.mc</td>
<td>137</td>
</tr>
<tr>
<td>spddep</td>
<td>139</td>
</tr>
<tr>
<td>spddep-defunct</td>
<td>140</td>
</tr>
<tr>
<td>spweights.constants</td>
<td>143</td>
</tr>
<tr>
<td>ssw</td>
<td>144</td>
</tr>
<tr>
<td>subset.listw</td>
<td>145</td>
</tr>
<tr>
<td>subset.nb</td>
<td>146</td>
</tr>
<tr>
<td>summary.nb</td>
<td>147</td>
</tr>
<tr>
<td>tolerance.nb</td>
<td>148</td>
</tr>
<tr>
<td>tri2nb</td>
<td>150</td>
</tr>
<tr>
<td>write.nb.gal</td>
<td>151</td>
</tr>
</tbody>
</table>

### Index

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>153</td>
</tr>
</tbody>
</table>
**aggregate.nb**

**Aggregate a spatial neighbours object**

**Description**

The method aggregates a spatial neighbours object, creating a new object listing the neighbours of the aggregates.

**Usage**

```r
## S3 method for class 'nb'
aggregate(x, IDs, remove.self = TRUE, ...)
```

**Arguments**

- **x**: an nb neighbour object
- **IDs**: a character vector of IDs grouping the members of the neighbour object
- **remove.self**: default TRUE: remove self-neighbours resulting from aggregation
- **...**: unused - arguments passed through

**Value**

an nb neighbour object, with empty aggregates dropped.

**Note**

Method suggested by Roberto Patuelli

**Author(s)**

Roger Bivand <Roger.Bivand@nhh.no>

**Examples**

```r
data(used.cars, package="spData")
data(state)
cont_st <- match(attr(usa48.nb, "region.id"), state.abb)
cents <- as.matrix(as.data.frame(state.center))[cont_st,]
par <- par(mfrow=c(2,1))
plot(usa48.nb, cents, xlim=c(-125, -65), ylim=c(25, 50))
IDs <- as.character(state.division[cont_st])
agg_cents <- aggregate(cents, list(IDs), mean)
agg_nb <- aggregate(usa48.nb, IDs)
plot(agg_nb, agg_cents[, 2:3], xlim=c(-125, -65), ylim=c(25, 50))
text(agg_cents[, 2:3], agg_cents[, 1], cex=0.6)
par(par)
```
Measure distance from plot

Description
Measure a distance between two points on a plot using locator; the function checks par("plt") and par("usr") to try to ensure that the aspect ratio y/x is 1, that is that the units of measurement in both x and y are equivalent.

Usage
airdist(ann=FALSE)

Arguments
ann annotate the plot with line measured and distance

Value
a list with members:
dist distance measured
coords coordinates between which distance is measured

Author(s)
Roger Bivand <Roger.Bivand@nhh.no>

See Also
locator

Distance-weighted autocovariate

Description
Calculates the autocovariate to be used in autonormal, autopoisson or autologistic regression. Three distance-weighting schemes are available.

Usage
autocov_dist(z, xy, nbs = 1, type = "inverse", zero.policy = NULL, style = "B", longlat=NULL)
Arguments

z the response variable

xy a matrix of coordinates or a SpatialPoints, sf or sfc points object

nbs neighbourhood radius; default is 1

type the weighting scheme: "one" gives equal weight to all data points in the neighbourhood; "inverse" (the default) weights by inverse distance; "inverse.squared" weights by the square of "inverse"

zero.policy default NULL, use global option value; if FALSE stop with error for any empty neighbour sets, if TRUE permit the weights list to be formed with zero-length weights vectors

style default “B” (changed from “W” 2015-01-27); style can take values “W”, “B”, “C”, “U”, and “S”

longlat TRUE if point coordinates are longitude-latitude decimal, in which case distances are measured in kilometers; if xy is a SpatialPoints object, the value is taken from the object itself

Value

A numeric vector of autocovariate values

Note

The validity of this approach strongly hinges on the correct choice of the neighbourhood scheme! Using ‘style=”B”’ ensures symmetry of the neighbourhood matrix (i.e. \( w_{nm} = w_{mn} \)). Please see Bardos et al. (2015) for details.

Author(s)

Carsten F. Dormann and Roger Bivand

References


See Also

nb2listw
Examples

columbus <- st_read(system.file("shapes/columbus.shp"., package="spData")[[1], quiet=TRUE)
#xy <- cbind(columbus$X, columbus$Y)
xy <- st_coordinates(st_centroid(st_geometry(columbus),
of_largest_polygon=TRUE))
ac1a <- autocov_dist(columbus$CRIME, xy, nbs=10, style="B",
type="one")
acinv <- autocov_dist(columbus$CRIME, xy, nbs=10, style="B",
type="inverse")
acinv2a <- autocov_dist(columbus$CRIME, xy, nbs=10, style="B",
type="inverse.squared")
plot(ac1a ~ columbus$CRIME, pch=16, ylim=c(0,9000))
points(acinv ~ columbus$CRIME, pch=16, col="red")
points(acinv2a ~ columbus$CRIME, pch=16, col="blue")
legend("topleft", legend=c("one", "inverse", "inverse.squared"),
col=c("black", "red", "blue"), bty="n", pch=16)

nb <- dnearneigh(xy, 0, 10)
lw <- nb2listw(nb, style="B")
ac1b <- lag(lw, columbus$CRIME)
all.equal(ac1b, ac1a)

nb <- nbdistst(nbdists, nb, xy)
gl <- lapply(nb, function(x) 1/x)
lw <- nblistw(nb, glist=gl, style="B")
acinvb <- lag(lw, columbus$CRIME)
all.equal(acinvb, acinv)

nb <- dnearealh(xy, 0, 10)
lw <- nbdistst(nbdists, nb, xy)
gl2 <- lapply(nb, function(x) 1/(x^2))
lw <- nb2listw(nb, glist=gl2, style="B")
acinv2b <- lag(lw, columbus$CRIME)
all.equal(acinv2b, acinv2a)

bhicv

Data set with 4 life condition indices of Belo Horizonte region

Description

The data are collected in the Atlas of condition indices published by the Joao Pinheiro Foundation
and UNDP.
Format

A shape polygon object with seven variables:

- **id**  The identifier
- **Name**  Name of city
- **Population**  The population of city
- **HLCI**  Health Life Condition Index
- **ELCI**  Education Life Condition Index
- **CLCI**  Children Life Condition Index
- **ELCI**  Economic Life Condition Index

Examples

```r
bh <- st_read(system.file("etc/shapes/bhicv.shp", 
package="spdep")[1])
```

---

**card**  
*Cardinalities for neighbours lists*

Description

The function tallies the numbers of neighbours of regions in the neighbours list.

Usage

```r
card(nb)
```

Arguments

- **nb**  a neighbours list object of class nb

Details

“nb” objects are stored as lists of integer vectors, where the vectors contain either the indices in the range 1:n for n as length(nb) of the neighbours of region i, or as.integer(0) to signal no neighbours. The function `card(nb)` is used to extract the numbers of neighbours from the “nb” object.

Value

An integer vector of the numbers of neighbours of regions in the neighbours list.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>
References


See Also

summary.nb

Examples

col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[[1]])
table(card(col.gal.nb))

cell2nb

Generate neighbours list for grid cells

Description

The function generates a list of neighbours for a grid of cells. Helper functions are used to convert to and from the vector indices for row and column grid positions, and rook (shared edge) or queen (shared edge or vertex) neighbour definitions are applied by type. If torus is TRUE, the grid is mapped onto a torus, removing edge effects.

Usage

cell2nb(nrow, ncol, type="rook", torus=FALSE, legacy=FALSE, x=NULL)
vi2mrc(i, nrow, ncol)

Arguments

nrow number of rows in the grid, may also be an object inheriting from class "SpatialGrid" or "GridTopology" only in cell2nb
ncol number of columns in the grid; if nrow or x is an object inheriting from class "SpatialGrid" or "GridTopology", it may be omitted
type default rook, may also be queen
torus default FALSE, if TRUE map grid onto torus
legacy default FALSE, nrow/ncol reversed, if TRUE wrong col/row directions (see https://github.com/r-spatial/spdep/issues/20)
x if given, an object inheriting from class "SpatialGrid" or "GridTopology". and replaces nrow and ncol
i vector of vector indices corresponding to rowcol, a matrix with two columns of row, column indices
Value

The function returns an object of class nb with a list of integer vectors containing neighbour region number ids. See card for details of “nb” objects.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

See Also

summary.nb, card

Examples

```r
nb7rt <- cell2nb(7, 7)
summary(nb7rt)
xyc <- attr(nb7rt, "region.id")
xy <- matrix(as.integer(unlist(strsplit(xyc, ":"))), ncol=2, byrow=TRUE)
plot(nb7rt, xy)
nb7rt <- cell2nb(7, 7, torus=TRUE)
summary(nb7rt)
# https://github.com/r-spatial/spdep/issues/20
GT <- GridTopology(c(1, 1), c(1, 1), c(10, 50))
SPix <- as(SpatialGrid(GT), "SpatialPixels")
nb_rook_cont <- poly2nb(as(SPix, "SpatialPolygons"), queen=FALSE)
nb_rook_dist <- dnearneigh(coordinates(SPix), 0, 1.01)
all.equal(nb_rook_cont, nb_rook_dist, check.attributes=FALSE)
## [1] TRUE
t.nb <- cell2nb(GT, type='rook', legacy=TRUE)
isTRUE(all.equal(nb_rook_cont, t.nb, check.attributes=FALSE))
## [1] FALSE
t.nb <- cell2nb(GT, type='rook')
isTRUE(all.equal(nb_rook_cont, t.nb, check.attributes=FALSE))
## [1] TRUE
# https://github.com/r-spatial/spdep/issues/55
# problem reported in issue caused by rep() cycling in unexpected order
GT <- GridTopology(c(1, 1), c(1, 1), c(22, 11))
SPix <- as(SpatialGrid(GT), "SpatialPixels")
nb_rook_cont <- poly2nb(as(SPix, "SpatialPolygons"), queen=FALSE)
nb_rook_dist <- dnearneigh(coordinates(SPix), 0, 1.01)
all.equal(nb_rook_cont, nb_rook_dist, check.attributes=FALSE)
t.nb <- cell2nb(GT, type='rook', legacy=TRUE)
isTRUE(all.equal(nb_rook_cont, t.nb, check.attributes=FALSE))
## [1] FALSE
t.nb <- cell2nb(GT, type='rook', legacy=FALSE)
isTRUE(all.equal(nb_rook_cont, t.nb, check.attributes=FALSE))
## [1] TRUE
```
choynowski

Choynowski probability map values

Description

Calculates Choynowski probability map values.

Usage

choynowski(n, x, row.names=NULL, tol = .Machine$double.eps^0.5, legacy=FALSE)

Arguments

n
a numeric vector of counts of cases

x
a numeric vector of populations at risk

row.names
row names passed through to output data frame

tol
accumulate values for observed counts >= expected until value less than tol

legacy
default FALSE using vectorised alternating side ppois version, if true use original version written from sources and iterating down to tol

Value

A data frame with columns:

pmap
Poisson probability map values: probability of getting a more “extreme” count than actually observed, one-tailed with less than expected and more than expected folded together

type
logical: TRUE if observed count less than expected

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References


See Also

probmap
Examples

```r
columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[[1]])
```

Description

The data set is now part of the spData package

Usage

data(columbus)

Examples

```r
columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[[1]])```
**diffnb**

*Differences between neighbours lists*

**Description**

The function finds differences between lists of neighbours, returning a nb neighbour list of those found.

**Usage**

```r
diffnb(x, y, verbose=NULL)
```

**Arguments**

- `x` an object of class `nb`
- `y` an object of class `nb`
- `verbose` default NULL, use global option value; report regions ids taken from object attribute "region.id" with differences

**Value**

A neighbours list with class `nb`

**Author(s)**

Roger Bivand <Roger.Bivand@nhh.no>

**Examples**

```r
columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
coords <- st_centroid(st_geometry(columbus), of_largest_polygon=TRUE)
rn <- row.names(columbus)
knn1 <- knearneigh(coords, 1)
knn2 <- knearneigh(coords, 2)
b1 <- knn2nb(knn1, row.names=rn)
b2 <- knn2nb(knn2, row.names=rn)
diffs <- diffnb(b2, b1)
par <- par(no.readonly=TRUE)
plot(st_geometry(columbus), border="grey", reset=FALSE,
     main="Plot of first (black) and second (red)\nnearest neighbours")
plot(b1, coords, add=TRUE)
plot(diffs, coords, add=TRUE, col="red", lty=2)
par(opar)
```
**dneareigh**  
*Neighbourhood contiguity by distance*

**Description**

The function identifies neighbours of region points by Euclidean distance between lower (greater than or equal to (changed from version 1.1-7)) and upper (less than or equal to) bounds, or with longlat = TRUE, by Great Circle distance in kilometers.

**Usage**

```r
dneareigh(x, d1, d2, row.names = NULL, longlat = NULL, bounds=c("GE", "LE"), use_kd_tree=TRUE, symtest=FALSE, use_s2=packageVersion("s2") > "1.0.7", k=200, dwithin=FALSE)
```

**Arguments**

- **x**: matrix of point coordinates, an object inheriting from SpatialPoints or an "sf" or "sfc" object; if the "sf" or "sfc" object geometries are in geographical coordinates (use_s2=FALSE, sf::st_is_longlat(x) == TRUE and sf::sf_use_s2() == TRUE), s2 will be used to find the neighbours because it will (we hope) use spatial indexing https://github.com/r-spatial/s2/issues/125 as opposed to the legacy method which uses brute-force (at present s2 also uses brute-force)
- **d1**: lower distance bound
- **d2**: upper distance bound
- **row.names**: character vector of region ids to be added to the neighbours list as attribute region.id, default seq(1,nrow(x))
- **longlat**: TRUE if point coordinates are longitude-latitude decimal degrees, in which case distances are measured in kilometers; if x is a SpatialPoints object, the value is taken from the object itself, and overrides this argument if not NULL
- **bounds**: character vector of length 2, default c("GE", "LE"). (GE: greater than or equal to, LE: less than or equal to) that is the finite and closed interval [d1, d2], d1 <= x <= d2. The first element may also be "GT" (GT: greater than), the second "LT" (LT: less than) for finite, open intervals excluding the bounds; the first bound default was changed from "GT" to "GE" in release 1.1-7. When creating multiple distance bands, finite, half-open right-closed intervals may be used until the final interval to avoid overlapping on bounds: "GE", "LT", that is [d1,d2), d1 <= x < d2
- **use_kd_tree**: default TRUE, if TRUE, use dbscan frNN if available (permitting 3D distances).
- **symtest**: Default FALSE; before release 1.1-7, TRUE - run symmetry check on output object, costly with large numbers of points.
- **use_s2**: default=packageVersion("s2") > "1.0.7", as of s2 > 1.0-7, distance bound computations use spatial indexing so when sf::sf_use_s2() is TRUE, s2::s2_closest_edges() will be used for distances on the sphere for "sf" or "sfc" objects if s2 > 1.0-7.
**k**

Default 200, the number of closest points to consider when searching when using **s2**

**dwithin**

Default FALSE, if TRUE, use **s2::s2_dwithin_matrix()**, both if use_s2=TRUE, sf::st_is_longlat(x) == TRUE and sf::sf_use_s2() == TRUE; yields the same lists of neighbours as **s2::s2_closest_edges()**.

**Value**

The function returns a list of integer vectors giving the region id numbers for neighbours satisfying the distance criteria. See **card** for details of "nb" objects.

**Author(s)**

Roger Bivand <Roger.Bivand@nhh.no>

**See Also**

knearneigh, card

**Examples**

columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
coords <- st_centroid(st_geometry(columbus), of_largest_polygon=TRUE)
rn <- row.names(columbus)
k1 <- knn2nb(knearneigh(coords))
all.linked <- max(unlist(nbdists(k1, coords)))
col.nb.0.all <- dnearneigh(coords, 0, all.linked, row.names=rn)
summary(col.nb.0.all, coords)

(sfc_obj <- st_centroid(st_geometry(columbus)))
col.nb.0.all_sf <- dnearneigh(sfc_obj, 0, all.linked, row.names=rn)
all.equal(col.nb.0.all, col.nb.0.all_sf, check.attributes=FALSE)
data(state)
us48.fipsno <- read.geoda(system.file("etc/weights/us48.txt", package="spdep")[[1]])
if (as.numeric(paste(version$major, version$minor, sep="")) < 19) {
  m50.48 <- match(us48.fipsno$"State.name", state.name)
} else {
  m50.48 <- match(us48.fipsno$"State.name", state.name)
}
xy <- as.matrix(as.data.frame(state.center))[m50.48,]
lkl <- knn2nb(knearneigh(xy, k=1, longlat=FALSE))
(all.linked <- max(unlist(nbdists(lkl1, xy, longlat=FALSE))))
l1.nb <- dnearneigh(xy, 0, all.linked, longlat=FALSE)
summary(l1.nb, xy, longlat=TRUE, scale=0.5)
gck1 <- knn2nb(knearneigh(xy, k=1, longlat=TRUE))
(all.linked <- max(unlist(nbdists(gck1, xy, longlat=TRUE))))
gc.nb <- dnearneigh(xy, 0, all.linked, longlat=TRUE)
droplinks

Summary

Drops links to and from or just to a region from a neighbours list. The example corresponds to Fingleton’s Table 1, p. 6, for lattices 5 to 19.

Usage

droplinks(nb, drop, sym=TRUE)
droplinks

Arguments

- `nb`: a neighbours list object of class `nb`
- `drop`: either a logical vector the length of `nb`, or a character vector of named regions corresponding to `nb`'s region.id attribute, or an integer vector of region numbers
- `sym`: TRUE for removal of both "row" and "column" links, FALSE for only "row" links

Value

The function returns an object of class `nb` with a list of integer vectors containing neighbour region number ids.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References


See Also

- `is.symmetric.nb`

Examples

```r
rho <- c(0.2, 0.5, 0.95, 0.999, 1.0)
s <- c(5, 7, 9, 11, 13, 15, 17, 19)
ms <- matrix(0, nrow=length(ns), ncol=length(rho))
rownames(ms) <- ns
colnames(ms) <- rho
mxs <- matrix(0, nrow=length(ns), ncol=length(rho))
rownames(mxs) <- ns
colnames(mxs) <- rho
for (i in 1:length(ns)) {
  nblist <- cell2nb(ns[i], ns[i])
  nbdropped <- droplinks(nblist, ((ns[i]*ns[i])+1)/2, sym=FALSE)
  listw <- nb2listw(nbdropped, style="W", zero.policy=TRUE)
  wmat <- listw2mat(listw)
  for (j in 1:length(rho)) {
    mat <- diag(ns[i]*ns[i]) - rho[j] * wmat
    res <- diag(solve(t(mat) %*% mat))
    mns[i,j] <- mean(res)
    mxs[i,j] <- max(res)
  }
}
print(mns)
print(mxs)
```
**EBest**  
*Global Empirical Bayes estimator*

**Description**

The function computes global empirical Bayes estimates for rates "shrunk" to the overall mean.

**Usage**

```r
EBest(n, x, family="poisson")
```

**Arguments**

- `n` a numeric vector of counts of cases
- `x` a numeric vector of populations at risk
- `family` either "poisson" for rare conditions or "binomial" for non-rare conditions

**Details**

Details of the implementation for the "poisson" family are to be found in Marshall, p. 284–5, and Bailey and Gatrell p. 303–306 and exercise 8.2, pp. 328–330. For the "binomial" family, see Martuzzi and Elliott (implementation by Olaf Berke).

**Value**

A data frame with two columns:

- `raw` a numerical vector of raw (crude) rates
- `estmm` a numerical vector of empirical Bayes estimates

and a `parameters` attribute list with components:

- `a` global method of moments phi value
- `m` global method of moments gamma value

**Author(s)**

Roger Bivand <Roger.Bivand@nhh.no> and Olaf Berke, Population Medicine, OVC, University of Guelph, CANADA
References


See Also

EBlocal, probmap, EBImoran.mc

Examples

```r
auckland <- st_read(system.file("shapes/auckland.shp", package="spData")[[1]], quiet=TRUE)
res <- EBest(auckland$M77_85, 9*auckland$Und5_81)
attr(res, "parameters")
auckland$estmm000 <- res$estmm*1000
plot(auckland[,"estmm000"], breaks=c(0,2,2.5,3,3.5,5),
main="Infant mortality per 1000 per year")
data(huddersfield, package="spData")
res <- EBest(huddersfield$cases, huddersfield$total, family="binomial")
round(res[,1:2],4)*100
```

EBImoran.mc

Permutation test for empirical Bayes index

Description

An empirical Bayes index modification of Moran's I for testing for spatial autocorrelation in a rate, typically the number of observed cases in a population at risk. The index value is tested by using nsim random permutations of the index for the given spatial weighting scheme, to establish the rank of the observed statistic in relation to the nsim simulated values.

Usage

```r
EBImoran.mc(n, x, listw, nsim, zero.policy = NULL,
alternative = "greater", spChk=NULL, return_boot=FALSE,
subtract_mean_in_numerator=TRUE)
```

Arguments

- `n`: a numeric vector of counts of cases the same length as the neighbours list in `listw`
- `x`: a numeric vector of populations at risk the same length as the neighbours list in `listw`
- `listw`: a listw object created for example by `nb2listw`
- `nsim`: number of permutations
zero.policy  default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
alternative  a character string specifying the alternative hypothesis, must be one of "greater" (default), or "less"
spChk should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use get.spChkOption()
return_boot return an object of class boot from the equivalent permutation bootstrap rather than an object of class htest
subtract_mean_in_numerator default TRUE, if TRUE subtract mean of z in numerator of EBI equation on p. 2157 in reference (consulted with Renato Assunção 2016-02-19); until February 2016 the default was FALSE agreeing with the printed paper.

Details

The statistic used is (m is the number of observations):

$$EBI = \frac{m}{\sum_{i=1}^{m} \sum_{j=1}^{m} w_{ij}} \frac{\sum_{i=1}^{m} \sum_{j=1}^{m} w_{ij} z_i z_j}{\sum_{i=1}^{m} (z_i - \bar{z})^2}$$

where:

$$z_i = \frac{p_i - b}{\sqrt{v_i}}$$

and:

$$p_i = \frac{n_i}{x_i}$$
$$v_i = a + \left( b / x_i \right)$$
$$b = \sum_{i=1}^{m} n_i / \sum_{i=1}^{m} x_i$$
$$a = s^2 - b / \left( \sum_{i=1}^{m} x_i / m \right)$$
$$s^2 = \sum_{i=1}^{m} x_i (p_i - b)^2 / \sum_{i=1}^{m} x_i$$

Value

A list with class htest and mc.sim containing the following components:

 statistic  the value of the observed Moran’s I.
 parameter  the rank of the observed Moran’s I.
 p.value  the pseudo p-value of the test.
 alternative  a character string describing the alternative hypothesis.
 method  a character string giving the method used.
 data.name  a character string giving the name(s) of the data, and the number of simulations.
 res nsim simulated values of statistic, final value is observed statistic.
 z  a numerical vector of Empirical Bayes indices as z above
Author(s)
Roger Bivand <Roger.Bivand@nhh.no>

References

See Also
moran, moran.mc, EBest

Examples
nc.sids <- st_read(system.file("shapes/sids.shp", package="spData")[[1]], quiet=TRUE)
rn <- as.character(nc.sids$FIPS)
ncCC89_nb <- read.gal(system.file("weights/ncCC89.gal", package="spData")[[1]],
                      region.id=rn)
EBImoran.mc(nc.sids$SID74, nc.sids$BIR74,
            nb2listw(ncCC89_nb, style="B", zero.policy=TRUE), nsim=999, zero.policy=TRUE)
sids.p <- nc.sids$SID74 / nc.sids$BIR74
moran.mc(sids.p, nb2listw(ncCC89_nb, style="B", zero.policy=TRUE),
          nsim=999, zero.policy=TRUE)

EBlocal

Local Empirical Bayes estimator

Description
The function computes local empirical Bayes estimates for rates "shrunk" to a neighbourhood mean for neighbourhoods given by the nb neighbourhood list.

Usage
EBlocal(ri, ni, nb, zero.policy = NULL, spChk = NULL, geoda=FALSE)

Arguments
ri         a numeric vector of counts of cases the same length as the neighbours list in nb; if there are many zero counts, some estimates may be affected by division by zero, see https://stat.ethz.ch/pipermail/r-sig-geo/2022-January/028882.html
ni         a numeric vector of populations at risk the same length as the neighbours list in nb
nb         a nb object of neighbour relationships
zero.policy: default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA

spChk: should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use get.spChkOption()

geoda: default=FALSE, following Marshall’s algorithm as interpreted by Bailey and Gatrell, pp. 305-307, and exercise 8.2, pp. 328-330 for the definition of phi; TRUE for the definition of phi used in GeoDa (see discussion on OpenSpace mailing list June 2003: http://agec221.agecon.uiuc.edu/pipermail/openspace/2003-June/thread.html)

Details

Details of the implementation are to be found in Marshall, p. 286, and Bailey and Gatrell p. 307 and exercise 8.2, pp. 328–330. The example results do not fully correspond to the sources because of slightly differing neighbourhoods, but are generally close. If there are many zero counts, some estimates may be affected by division by zero, see https://stat.ethz.ch/pipermail/r-sig-geo/2022-January/028882.html.

Value

A data frame with two columns:

- raw: a numerical vector of raw (crude) rates
- est: a numerical vector of local empirical Bayes estimates

and a parameters attribute list with components (if both are zero, the estimate will be NaN, https://stat.ethz.ch/pipermail/r-sig-geo/2022-January/028882.html):

- a: a numerical vector of local phi values
- m: a numerical vector of local gamma values

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>, based on contributions by Marilia Carvalho

References


See Also

EBest, probmap
Examples

```r
auckland <- st_read(system.file("shapes/auckland.shp", package="spData")[[1]], quiet=TRUE)
auckland.nb <- poly2nb(auckland)
res <- EBlocal(auckland$M77_85, 9*auckland$Und5_81, auckland.nb)
auckland$est000 <- res$est*1000
plot(auckland[, "est000"], breaks=c(0,2,2.5,3,3.5,8),
     main="Infant mortality per 1000 per year")
```

**Description**

The function provides simple interactive editing of neighbours lists to allow unneeded links to be deleted, and missing links to be inserted. It uses `identify` to pick the endpoints of the link to be deleted or added, and asks for confirmation before committing. If the result is not assigned to a new object, the editing will be lost - as in `edit`.

This method relies on direct contact with the graphics device. Do not use in RStudio.

**Usage**

```r
## S3 method for class 'nb'
edit(name, coords, polys=NULL, ..., use_region.id=FALSE)
```

**Arguments**

- `name`: an object of class `nb`
- `coords`: matrix of region point coordinates; if missing and `polys` inherits from `SpatialPolygons`, the label points of that object are used
- `polys`: if polygon boundaries supplied, will be used as background; must inherit from `SpatialPolygons`
- `...`: further arguments passed to or from other methods
- `use_region.id`: default `FALSE`, in `identify` use 1-based observation numbers, otherwise use the `nb` region.id attribute values

**Value**

The function returns an object of class `nb` with the edited list of integer vectors containing neighbour region number ids, with added attributes tallying the added and deleted links.

**Author(s)**

Roger Bivand <Roger.Bivand@nhh.no>

**See Also**

`summary.nb`, `plot.nb`
Examples

```r
## Not run:
columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
class(columbus)
if (FALSE) nnb1 <- edit.nb(col.gal.nb, polys=as(columbus, "Spatial"))
## End(Not run)
```

eire  

### Eire data sets

Description

The data set is now part of the spData package

Usage

data(eire)

geary  

### Compute Geary’s C

Description

A simple function to compute Geary’s C, called by geary.test and geary.mc;

\[
C = \frac{(n - 1)}{2 \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} (x_i - x_j)^2} \sum_{i=1}^{n} (x_i - \bar{x})^2
\]

geary.intern is an internal function used to vary the similarity criterion.

Usage

geary(x, listw, n, n1, S0, zero.policy=NULL)

Arguments

- `x`: a numeric vector the same length as the neighbours list in listw
- `listw`: a listw object created for example by nb2listw
- `n`: number of zones
- `n1`: `n - 1`
- `S0`: global sum of weights
- `zero.policy`: default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
Value

A list with

- **C**: Geary’s C
- **K**: sample kurtosis of x

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References


See Also

- `geary.test`, `geary.mc`, `sp.mantel.mc`

Examples

```r
data(oldcol)
col.W <- nb2listw(COL.nb, style="W")
str(geary(COL.OLD$CRIME, col.W, length(COL.nb), length(COL.nb)-1,
Szero(col.W)))
```

---

### geary.mc

**Permutation test for Geary’s C statistic**

Description

A permutation test for Geary’s C statistic calculated by using nsim random permutations of x for the given spatial weighting scheme, to establish the rank of the observed statistic in relation to the nsim simulated values.

Usage

```r
geary.mc(x, listw, nsim, zero.policy=NULL, alternative="greater",
spChk=NULL, adjust.n=TRUE, return_boot=FALSE)
```

Arguments

- **x**: a numeric vector the same length as the neighbours list in listw
- **listw**: a listw object created for example by `nb2listw`
- **nsim**: number of permutations
- **zero.policy**: default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
alternative  a character string specifying the alternative hypothesis, must be one of "greater" (default), or "less"; this reversal corresponds to that on geary.test described in the section on the output statistic value, based on Cliff and Ord 1973, p. 21 (changed 2011-04-11, thanks to Daniel Garavito).

spChk     should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use get.spChkOption()

adjust.n  default TRUE, if FALSE the number of observations is not adjusted for no-neighbour observations, if TRUE, the number of observations is adjusted

return_boot return an object of class boot from the equivalent permutation bootstrap rather than an object of class htest

Value

A list with class htest and mc.sim containing the following components:

statistic  the value of the observed Geary's C.
parameter  the rank of the observed Geary's C.
p.value    the pseudo p-value of the test.
alternative a character string describing the alternative hypothesis.
method     a character string giving the method used.
data.name  a character string giving the name(s) of the data, and the number of simulations.
res        nsim simulated values of statistic, final value is observed statistic

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References


See Also

geary, geary.test

Examples

data(oldcol)
sim1 <- geary.mc(COL.OLD$CRIME, nb2listw(COL.nb, style="W"),
                 nsim=99, alternative="less")
sim1
mean(sim1$res)
var(sim1$res)
summary(sim1$res)
colold.lags <- nblag(COL.nb, 3)
sim2 <- geary.mc(COL.OLD$CRIME, nb2listw(colold.lags[[2]],
                 style="W"), nsim=99)
sim2
geary.test

Geary’s C test for spatial autocorrelation

Description

Geary’s test for spatial autocorrelation using a spatial weights matrix in weights list form. The assumptions underlying the test are sensitive to the form of the graph of neighbour relationships and other factors, and results may be checked against those of geary.mc permutations.

Usage

geary.test(x, listw, randomisation=TRUE, zero.policy=NULL, alternative="greater", spChk=NULL, adjust.n=TRUE)

Arguments

x 
a numeric vector the same length as the neighbours list in listw
listw 
a listw object created for example by nb2listw
randomisation 
variance of I calculated under the assumption of randomisation, if FALSE normality
zero.policy 
default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
alternative 
a character string specifying the alternative hypothesis, must be one of "greater" (default), "less" or "two.sided".
spChk 
should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use get.spChkOption()
adjust.n 
default TRUE, if FALSE the number of observations is not adjusted for no-neighbour observations, if TRUE, the number of observations is adjusted

Value

A list with class htest containing the following components:

statistic 
the value of the standard deviate of Geary’s C, in the order given in Cliff and Ord 1973, p. 21, which is (EC - C) / sqrt(VC), that is with the sign reversed with respect to the more usual (C - EC) / sqrt(VC); this means that the “greater” alternative for the Geary C test corresponds to the “greater” alternative for Moran’s I test.
p.value 
the p-value of the test.
estimate the value of the observed Geary’s C, its expectation and variance under the method assumption.

alternative a character string describing the alternative hypothesis.

method a character string giving the assumption used for calculating the standard deviate.

data.name a character string giving the name(s) of the data.

Note

The derivation of the test (Cliff and Ord, 1981, p. 18) assumes that the weights matrix is symmetric. For inherently non-symmetric matrices, such as k-nearest neighbour matrices, listw2U() can be used to make the matrix symmetric. In non-symmetric weights matrix cases, the variance of the test statistic may be negative (thanks to Franz Munoz I for a well documented bug report). Geary’s C is affected by non-symmetric weights under normality much more than Moran’s I. From 0.4-35, the sign of the standard deviate of C is changed to match Cliff and Ord (1973, p. 21).

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References


See Also

geary, geary.mc, listw2U

Examples

data(oldcol)
geary.test(COL.OLD$CRIME, nb2listw(COL.nb, style="W"))
geary.test(COL.OLD$CRIME, nb2listw(COL.nb, style="W"),
randomisation=FALSE)
colold.lags <- nblag(COL.nb, 3)
geary.test(COL.OLD$CRIME, nb2listw(colold.lags[[2]],
style="W"))
geary.test(COL.OLD$CRIME, nb2listw(colold.lags[[3]],
style="W"), alternative="greater")
print(is.symmetric.nb(COL.nb))
coords.OLD <- cbind(COL.OLD$X, COL.OLD$Y)
COL.k4.nb <- knn2nb(knearneigh(coords.OLD, 4))
print(is.symmetric.nb(COL.k4.nb))
geary.test(COL.OLD$CRIME, nb2listw(COL.k4.nb, style="W"))
geary.test(COL.OLD$CRIME, nb2listw(COL.k4.nb, style="W"),
randomisation=FALSE)
cat("Note non-symmetric weights matrix - use listw2U()
")
```r
geary.test(COL.OLD$CRIME, listw2U(nb2listw(COL.k4.nb, style="W")))
geary.test(COL.OLD$CRIME, listw2U(nb2listw(COL.k4.nb, style="W")), randomisation=FALSE)
```

---

### globalG.test

**Global G test for spatial autocorrelation**

**Description**

The global G statistic for spatial autocorrelation, complementing the local Gi LISA measures: `localG`

**Usage**

```r
globalG.test(x, listw, zero.policy=NULL, alternative="greater", spChk=NULL, adjust.n=TRUE, B1correct=TRUE, adjust.x=TRUE, Arc_all_x=FALSE)
```

**Arguments**

- **x**: a numeric vector the same length as the neighbours list in `listw`
- **listw**: a `listw` object created for example by `nb2listw`; if a sequence of distance bands is to be used, it is recommended that the weights style be binary (one of `c("B","C","U")`).
- **zero.policy**: default `NULL`, use global option value; if `TRUE` assign zero to the lagged value of zones without neighbours, if `FALSE` assign NA
- **alternative**: a character string specifying the alternative hypothesis, must be one of "greater" (default), "less" or "two.sided".
- **spChk**: should the data vector names be checked against the spatial objects for identity integrity, `TRUE`, or `FALSE`, default `NULL` to use `get.spChkOption()`
- **adjust.n**: default `TRUE`, if `FALSE` the number of observations is not adjusted for no-neighbour observations, if `TRUE` the number of observations is adjusted
- **B1correct**: default `TRUE`, if `TRUE`, the erratum referenced below: "On page 195, the coefficient of W2 in B1, (just below center of the page) should be 6, not 3." is applied; if `FALSE`, 3 is used (as in CrimeStat IV)
- **adjust.x**: default `TRUE`, if `TRUE`, x values of observations with no neighbours are omitted in the denominator of G
- **Arc_all_x**: default `FALSE`, if `Arc_all_x=TRUE` and `adjust.x=TRUE`, use the full x vector in part of the denominator term for G
Value

A list with class htest containing the following components:

- statistic: the value of the standard deviate of Moran's I.
- p.value: the p-value of the test.
- estimate: the value of the observed statistic, its expectation and variance.
- alternative: a character string describing the alternative hypothesis.
- data.name: a character string giving the name(s) of the data.

Author(s)

Hisaji ONO <hi-on@mn.xdsl.ne.jp> and Roger Bivand <Roger.Bivand@nhh.no>

References


See Also

localG

Examples

```r
nc.sids <- st_read(system.file("shapes/sids.shp", package="spData")[[1]], quiet=TRUE)
sids79rate <- (1000*nc.sids$SID79)/nc.sids$BIR79
dists <- c(10, 20, 30, 33, 40, 50, 60, 70, 80, 90, 100)
ndists <- length(dists)
ZG <- vector(mode="list", length=ndists)
names(ZG) <- as.character(dists)
milesxy <- cbind(nc.sids$east, nc.sids$north)
for (i in 1:ndists) {
  thisnb <- dnearneigh(milesxy, 0, dists[i])
  thislw <- nb2listw(thisnb, style="B", zero.policy=TRUE)
  ZG[[i]] <- globalG.test(sidsrate79, thislw, zero.policy=TRUE)
}
t(sapply(ZG, function(x) c(x$estimate[1], x$statistic, p.value=unname(x$p.value))))
for (i in 1:ndists) {
  thisnb <- dnearneigh(milesxy, 0, dists[i])
  thislw <- nb2listw(thisnb, style="B", zero.policy=TRUE)
  ZG[[i]] <- globalG.test(sidsrate79, thislw, zero.policy=TRUE, alternative="two.sided")
}
t(sapply(ZG, function(x) c(x$estimate[1], x$statistic, p.value=unname(x$p.value))))
```
Description

n.comp.nb() finds the number of disjoint connected subgraphs in the graph depicted by nb.obj - a spatial neighbours list object.

Usage

n.comp.nb(nb.obj)

Arguments

nb.obj a neighbours list object of class nb

Value

A list of:

nc number of disjoint connected subgraphs
comp.id vector with the indices of the disjoint connected subgraphs that the nodes in nb.obj belong to

Author(s)

Nicholas Lewin-Koh <nikko@hailmail.net>

See Also

plot.nb

Examples

columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[[1]])
coords <- st_coordinates(st_centroid(st_geometry(columbus)))
plot(col.gal.nb, coords, col="grey")
col2 <- droplinks(col.gal.nb, 21)
res <- n.comp.nb(col2)
table(res$comp.id)
plot(col2, coords, add=TRUE)
points(coords, col=res$comp.id, pch=16)
run <- FALSE
if (require(igraph, quietly=TRUE) && require(spatialreg, quietly=TRUE)) run <- TRUE
if (run) {
  B <- as(nb2listw(col2, style="B", zero.policy=TRUE), "CsparseMatrix")
g1 <- graph.adjacency(B, mode="undirected")
c1 <- clusters(g1)
print(c1$no == res$nc)

if (run) {
    print(all.equal(c1$membership, res$comp.id))
}

if (run) {
    print(all.equal(c1$csize, c(table(res$comp.id)), check.attributes=FALSE))
}

if (run) {
    W <- as(nb2listw(col2, style="W", zero.policy=TRUE), "CsparseMatrix")
g1W <- graph.adjacency(W, mode="directed", weighted="W")
c1W <- clusters(g1W)
    print(all.equal(c1W$membership, res$comp.id, check.attributes=FALSE))
}

if (run) {
    B1 <- get.adjacency(g1)
    print(all.equal(B, B1))
}

---

**graphneigh**

*Graph based spatial weights*

**Description**

Functions return a graph object containing a list with the vertex coordinates and the to and from indices defining the edges. Some/all of these functions assume that the coordinates are not exactly regularly spaced. The helper function graph2nb converts a graph object into a neighbour list. The plot functions plot the graph objects.

**Usage**

```r
# Gabriel
#```
```r
gabrielneigh(coords, nnmult=3)
relativeneigh(coords, nnmult=3)
```

```r
# SOI
```
```r
soi.graph(tri.nb, coords, quadsegs=10)
```

```r
# `graph2nb`
```
```r
graph2nb(gob, row.names=NULL, sym=FALSE)
```

```r
# S3 method for class 'Gabriel'
```
```r
plot(x, show.points=FALSE, add=FALSE, linecol=par(col), ...)
```

```r
# S3 method for class 'relative'
```
```r
plot(x, show.points=FALSE, add=FALSE, linecol=par(col), ...)
```

**Arguments**

- `coords`:
  - matrix of region point coordinates or SpatialPoints object or sfc points object
- `nnmult`:
  - scaling factor for memory allocation, default 3; if higher values are required, the function will exit with an error; example below thanks to Dan Putler
- `tri.nb`:
  - a neighbor list created from tri2nb
quadsegs  number of line segments making a quarter circle buffer, see the nQuadSegs argument in geos_unary

gob  a graph object created from any of the graph functions
row.names  character vector of region ids to be added to the neighbours list as attribute region.id, default seq(1,nrow(x))
sym  a logical argument indicating whether or not neighbors should be symmetric (if i->j then j->i)
x  object to be plotted
show.points  (logical) add points to plot
add  (logical) add to existing plot
linecol  edge plotting colour
...  further graphical parameters as in par(.)

Details

The graph functions produce graphs on a 2d point set that are all subgraphs of the Delaunay triangulation. The relative neighbor graph is defined by the relation, x and y are neighbors if
\[ d(x, y) \leq \min(\max(d(x, z), d(y, z)) | z \in S) \]
where \( d() \) is the distance, \( S \) is the set of points and \( z \) is an arbitrary point in \( S \). The Gabriel graph is a subgraph of the delaunay triangulation and has the relative neighbor graph as a sub-graph. The relative neighbor graph is defined by the relation \( x \) and \( y \) are Gabriel neighbors if
\[ d(x, y) \leq \min((d(x, z)^2 + d(y, z)^2)^{1/2} | z \in S) \]
where \( x, y, z \) and \( S \) are as before. The sphere of influence graph is defined for a finite point set \( S \), let \( r_x \) be the distance from point \( x \) to its nearest neighbor in \( S \), and \( C_x \) is the circle centered on \( x \). Then \( x \) and \( y \) are SOI neighbors iff \( C_x \) and \( C_y \) intersect in at least 2 places. From 2016-05-31, Computational Geometry in C code replaced by calls to functions in dbscan and sf; with a large quadsegs= argument, the behaviour of the function is the same, otherwise buffer intersections only closely approximate the original function.
See card for details of “nb” objects.

Value

A list of class Graph with the following elements

- \( np \)  number of input points
- \( from \)  array of origin ids
- \( to \)  array of destination ids
- \( nedges \)  number of edges in graph
- \( x \)  input x coordinates
- \( y \)  input y coordinates

The helper functions return an nb object with a list of integer vectors containing neighbor region number ids.
graphneigh

Author(s)
Nicholas Lewin-Koh <nikko@hailmail.net>

References

See Also
knearneigh, dnearneigh, knn2nb, card

Examples

columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
sf_obj <- st_centroid(st_geometry(columbus), of_largest_polygon)
sp_obj <- as(sf_obj, "Spatial")
coords <- st_coordinates(sf_obj)
suppressMessages(col.tri.nb <- tri2nb(coords))
col.gab.nb <- graph2nb(gabrielneigh(coords), sym=TRUE)
col.rel.nb <- graph2nb(relativeneigh(coords), sym=TRUE)
par(mfrow=c(2,2))
plot(st_geometry(columbus), border="grey")
plot(col.tri.nb,coords,add=TRUE)
title(main="Delaunay Triangulation", cex.main=0.6)
plot(st.geometry(columbus), border="grey")
plot(col.gab.nb, coords, add=TRUE)
title(main="Gabriel Graph", cex.main=0.6)
plot(st.geometry(columbus), border="grey")
plot(col.rel nb, coords, add=TRUE)
title(main="Relative Neighbor Graph", cex.main=0.6)
plot(st.geometry(columbus), border="grey")
if (require("dbscan", quietly=TRUE)) {
  col.soi.nb <- graph2nb(soi.graph(col.tri.nb,coords), sym=TRUE)
  plot(col.soi.nb, coords, add=TRUE)
title(main="Sphere of Influence Graph", cex.main=0.6)
}
par(mfrow=c(1,1))
col.tri.nb_sf <- tri2nb(sf_obj)
all.equal(col.tri.nb, col.tri.nb_sf, check.attributes=FALSE)
col.tri.nb_sp <- tri2nb(sp_obj)
all.equal(col.tri.nb, col.tri.nb_sp, check.attributes=FALSE)
if (require("dbscan", quietly=TRUE)) {
  col.soi nb_sf <- graph2nb(soi.graph(col.tri.nb, sf_obj), sym=TRUE)
  all.equal(col.soi.nb, col.soi.nb_sf, check.attributes=FALSE)
  col.soi nb_sp <- graph2nb(soi.graph(col.tri.nb, sp_obj), sym=TRUE)
  all.equal(col.soi.nb, col.soi.nb_sp, check.attributes=FALSE)
grid2nb

Construct neighbours for a GridTopology

Description

The function builds a neighbours list for a grid topology. It works for a k-dimensional grid topology, k>=1.

Usage

grid2nb(grid, d = grid@cells.dim,
       queen = TRUE, nb = TRUE, self = FALSE)

Arguments

grid
  An object of class GridTopology. One can avoid to supply this by just suplying the dimensions in the d argument.

d
  A scalar (for one dimensional grid) or a length k vector specifying the number of grid cells in each direction of the k dimensions.

queen
  Logical. Default is TRUE. To inform if the queen neighbourhood structure should be considered. If FALSE, only a hyper-cube with a common face will be considered neighbour. If TRUE, a single shared coordinate meets the contiguity condition.
include.self

nb Default TRUE. If TRUE, return the result as a neighbours list with class nb. If FALSE, the result is a matrix with $3^k$ columns if self = TRUE or $3^k-1$ if self = FALSE. Zeros are used for hyper-cubes at boundaries.

self Default FALSE, to indicate if the hyper-cube neighbour itself should be considered a neighbour.

Value

Either a matrix, if “nb” is FALSE or a neighbours list with class nb. See card for details of “nb” objects.

Note

This applies to a k-dimentional grid topology.

Author(s)

Elias T Krainski <eliaskrainski@gmail.com>

See Also

poly2nb, summary.nb, card

Examples

nb <- grid2nb(d = c(5L, 5L, 5L))
nb
summary(nb)

gt <- GridTopology(c(.125,.1), c(.25,.2), c(4L, 5L))
nb1 <- grid2nb(gt, queen = FALSE)
nb2 <- grid2nb(gt)

sg <- SpatialGrid(gt)
plot(sg, lwd=3)
plot(nb1, coordinates(sg), add=TRUE, lty=2, col=2, lwd=2)
plot(nb2, coordinates(sg), add=TRUE, lty=3, col=4)

str(grid2nb(d=5))

include.self Include self in neighbours list

Description

The function includes the region itself in its own list of neighbours, and sets attribute "self.included" to TRUE.
Usage

include.self(nb)

Arguments

nb input neighbours list of class nb

Value

The function returns an object of class nb with a list of integer vectors containing neighbour region number ids; attribute "self.included" is set to TRUE.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

See Also

summary.nb

Examples

columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[[1]])
coords <- coordinates(as(columbus, "Spatial"))
summary(col.gal.nb, coords)
summary(include.self(col.gal.nb), coords)

Description

Checks a neighbours list for symmetry/transitivity (if i is a neighbour of j, then j is a neighbour of i). This holds for distance and contiguity based neighbours, but not for k-nearest neighbours. The helper function sym.attr.nb() calls is.symmetric.nb() to set the sym attribute if needed, and make.sym.nb makes a non-symmetric list symmetric by adding neighbors. is.symmetric.glist checks a list of general weights corresponding to neighbours for symmetry for symmetric neighbours.

Usage

is.symmetric.nb(nb, verbose = NULL, force = FALSE)
sym.attr.nb(nb)
make.sym.nb(nb)
old.make.sym.nb(nb)
is.symmetric.glist(nb, glist)
is.symmetric.nb

Arguments

nb       an object of class nb with a list of integer vectors containing neighbour region number ids.
verbose  default NULL, use global option value; if TRUE prints non-matching pairs
force    do not respect a neighbours list sym attribute and test anyway
glist    list of general weights corresponding to neighbours

Value

TRUE if symmetric, FALSE if not; is.symmetric.glist returns a value with an attribute, "d", indicating for failed symmetry the largest failing value.

Note

A new version of make.sym.nb by Bjarke Christensen is now included. The older version has been renamed old.make.sym.nb, and their comparison constitutes a nice demonstration of vectorising speedup using sapply and lapply rather than loops. When any no-neighbour observations are present, old.make.sym.nb is used.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

See Also

read.gal

Examples

columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[[1]])
coords <- coordinates(as(columbus, "Spatial"))
ind <- row.names(as(columbus, "Spatial"))
print(is.symmetric.nb(col.gal.nb, verbose=TRUE, force=TRUE))
k4 <- knn2nb(knearneigh(coords, k=4), row.names=ind)
k4 <- sym.attr.nb(k4)
print(is.symmetric.nb(k4))
k4.sym <- make.sym.nb(k4)
print(is.symmetric.nb(k4.sym))
joincount.mc

Permutation test for same colour join count statistics

Description
A permutation test for same colour join count statistics calculated by using nsim random permutations of fx for the given spatial weighting scheme, to establish the ranks of the observed statistics (for each colour) in relation to the nsim simulated values.

Usage

joincount.mc(fx, listw, nsim, zero.policy=FALSE, alternative="greater", spChk=NULL)

Arguments

- fx: a factor of the same length as the neighbours and weights objects in listw
- listw: a listw object created for example by nb2listw
- nsim: number of permutations
- zero.policy: if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
- alternative: a character string specifying the alternative hypothesis, must be one of "greater" (default), or "less".
- spChk: should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use get.spChkOption()

Value
A list with class jclist of lists with class htest and mc.sim for each of the k colours containing the following components:

- statistic: the value of the observed statistic.
- parameter: the rank of the observed statistic.
- method: a character string giving the method used.
- data.name: a character string giving the name(s) of the data.
- p.value: the pseudo p-value of the test.
- alternative: a character string describing the alternative hypothesis.
- estimate: the mean and variance of the simulated distribution.
- res: nsim simulated values of statistic, the final element is the observed statistic

Author(s)
Roger Bivand <Roger.Bivand@nhh.no>
joincount.multi

References

See Also
joincount.test

Examples

data(oldcol)
HICRIME <- cut(COL.OLD$CRIME, breaks=c(0,35,80), labels=c("low","high"))
names(HICRIME) <- rownames(COL.OLD)
joincount.mc(HICRIME, nb2listw(COL.nb, style="B"), nsim=99)
joincount.test(HICRIME, nb2listw(COL.nb, style="B"))

Description
A function for tallying join counts between same-colour and different colour spatial objects, where neighbour relations are defined by a weights list. Given the global counts in each colour, expected counts and variances are calculated under non-free sampling, and a z-value reported. Since multiple tests are reported, no p-values are given, allowing the user to adjust the significance level applied. Jtot is the count of all different-colour joins.

Usage

joincount.multi(fx, listw, zero.policy = FALSE,
spChk = NULL, adjust.n=TRUE)
## S3 method for class 'jcmulti'
print(x, ...)

Arguments

fx
a factor of the same length as the neighbours and weights objects in listw

listw
a listw object created for example by nb2listw

zero.policy
if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA

adjust.n
default TRUE, if FALSE the number of observations is not adjusted for no-neighbour observations, if TRUE, the number of observations is adjusted consistently (up to and including spdep 0.3-28 the adjustment was inconsistent - thanks to Tomoki NAKAYA for a careful bug report)

spChk
should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use get.spChkOption()

x
object to be printed

... arguments to be passed through for printing
Value

A matrix with class jcmulti with row and column names for observed and expected counts, variance, and z-value.

Note

The derivation of the test (Cliff and Ord, 1981, p. 18) assumes that the weights matrix is symmetric. For inherently non-symmetric matrices, such as k-nearest neighbour matrices, listw2U() can be used to make the matrix symmetric. In non-symmetric weights matrix cases, the variance of the test statistic may be negative.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References


See Also

joincount.test

Examples

data(oldcol)
HICRIME <- cut(COL.OLD$CRIME, breaks=c(0,35,80), labels=c("low","high"))
names(HICRIME) <- rownames(COL.OLD)
joincount.multi(HICRIME, nb2listw(COL.nb, style="B"))

# Not run:
data(hopkins, package="spData")
image(1:32, 1:32, hopkins[5:36,36:5], breaks=c(-0.5, 3.5, 20),
col=c("white", "black"))
box()

hopkins.rook.nb <- cell2nb(32, 32, type="rook")
unlist(spweights.constants(nb2listw(hopkins.rook.nb, style="B")))
hopkins.queen.nb <- cell2nb(32, 32, type="queen")
hopkins.bishop.nb <- diffnb(hopkins.rook.nb, hopkins.queen.nb, verbose=FALSE)
hopkins4 <- hopkins[5:36,36:5]
hopkins4[which(hopkins4 > 3, arr.ind=TRUE)] <- 4
hopkins4.f <- factor(hopkins4)
table(hopkins4.f)
joincount.multi(hopkins4.f, nb2listw(hopkins.rook.nb, style="B"))
cat("replicates Upton & Fingleton table 3.4 (p. 166)\n")
joincount.multi(hopkins4.f, nb2listw(hopkins.bishop.nb, style="B"))
cat("replicates Upton & Fingleton table 3.6 (p. 168)\n")
joincount.multi(hopkins4.f, nb2listw(hopkins.queen.nb, style="B"))
cat("replicates Upton & Fingleton table 3.7 (p. 169)\n")

## End(Not run)
joincount.test  

**BB join count statistic for k-coloured factors**

**Description**

The BB join count test for spatial autocorrelation using a spatial weights matrix in weights list form for testing whether same-colour joins occur more frequently than would be expected if the zones were labelled in a spatially random way. The assumptions underlying the test are sensitive to the form of the graph of neighbour relationships and other factors, and results may be checked against those of joincount.mc permutations.

**Usage**

```
joincount.test(fx, listw, zero.policy=NULL, alternative="greater",
              sampling="nonfree", spChk=NULL, adjust.n=TRUE)
## S3 method for class 'jclist'
print(x, ...)```

**Arguments**

- `fx` a factor of the same length as the neighbours and weights objects in listw
- `listw` a listw object created for example by nb2listw
- `zero.policy` default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
- `alternative` a character string specifying the alternative hypothesis, must be one of "greater" (default), "less" or "two.sided".
- `sampling` default "nonfree", may be "free"
- `adjust.n` default TRUE, if FALSE the number of observations is not adjusted for no-neighbour observations, if TRUE, the number of observations is adjusted consistently (up to and including spdep 0.3-28 the adjustment was inconsistent - thanks to Tomoki NAKAYA for a careful bug report)
- `spChk` should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use get.spChkOption()
- `x` object to be printed
- `...` arguments to be passed through for printing

**Value**

A list with class `jclist` of lists with class `htest` for each of the k colours containing the following components:

- `statistic` the value of the standard deviate of the join count statistic.
- `p.value` the p-value of the test.
- `estimate` the value of the observed statistic, its expectation and variance under non-free sampling.
alternative a character string describing the alternative hypothesis.
method a character string giving the method used.
data.name a character string giving the name(s) of the data.

Note

The derivation of the test (Cliff and Ord, 1981, p. 18) assumes that the weights matrix is symmetric. For inherently non-symmetric matrices, such as k-nearest neighbour matrices, listw2U() can be used to make the matrix symmetric. In non-symmetric weights matrix cases, the variance of the test statistic may be negative.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References


See Also

joincount.mc, joincount.multi, listw2U

Examples

data(oldcol)
HICRIME <- cut(COL.OLD$CRIME, breaks=c(0,35,80), labels=c("low","high"))
names(HICRIME) <- rownames(COL.OLD)
joincount.test(HICRIME, nb2listw(COL.nb, style="B"))
joincount.test(HICRIME, nb2listw(COL.nb, style="B"), sampling="free")
joincount.test(HICRIME, nb2listw(COL.nb, style="C"))
joincount.test(HICRIME, nb2listw(COL.nb, style="S"))
joincount.test(HICRIME, nb2listw(COL.nb, style="W"))
by(card(COL.nb), HICRIME, summary)
print(is.symmetric.nb(COL.nb))
coords.OLD <- cbind(COL.OLD$X, COL.OLD$Y)
COL.k4.nb <- knn2nb(knearneigh(coords.OLD, 4))
print(is.symmetric.nb(COL.k4.nb))
joincount.test(HICRIME, nb2listw(COL.k4.nb, style="B"))
cat("Note non-symmetric weights matrix - use listw2U()\n")
joincount.test(HICRIME, listw2U(nb2listw(COL.k4.nb, style="B")))
knearneigh

K nearest neighbours for spatial weights

Description

The function returns a matrix with the indices of points belonging to the set of the \( k \) nearest neighbours of each other. If longlat = TRUE, Great Circle distances are used. A warning will be given if identical points are found.

Usage

knearneigh(x, k=1, longlat = NULL, use_kd_tree=TRUE)

Arguments

- **x**: matrix of point coordinates, an object inheriting from SpatialPoints or an "sf" or "sfc" object; if the "sf" or "sfc" object geometries are in geographical coordinates (sf::st_is_longlat(x) == TRUE and sf::sf_use_s2() == TRUE), s2 will be used to find the neighbours because it uses spatial indexing https://github.com/r-spatial/s2/issues/125 as opposed to the legacy method which uses brute-force
- **k**: number of nearest neighbours to be returned
- **longlat**: TRUE if point coordinates are longitude-latitude decimal degrees, in which case distances are measured in kilometers; if x is a SpatialPoints object, the value is taken from the object itself; longlat will override kd_tree
- **use_kd_tree**: logical value, if the dbscan package is available, use for finding k nearest neighbours when longlat is FALSE, and when there are no identical points; from https://github.com/r-spatial/spdep/issues/38, the input data may have more than two columns if dbscan is used

Details

The underlying legacy C code is based on the knn function in the class package.

Value

A list of class knn

- **nn**: integer matrix of region number ids
- **np**: number of input points
- **k**: input required k
- **dimension**: number of columns of x
- **x**: input coordinates

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>
See Also

knn, dnearneigh, knn2nb, kNN

Examples

columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
coords <- st_centroid(st_geometry(columbus), of_largest_polygon=TRUE)
col.knn <- knearneigh(coords, k=4)
plot(st_geometry(columbus), border="grey")
plot(knn2nb(col.knn), coords, add=TRUE)
title(main="K nearest neighbours, k = 4")
data(state)
us48.fipsno <- read.geoda(system.file("etc/weights/us48.txt", package="spdep")[[1]])
if (as.numeric(paste(version$major, version$minor, sep="")) < 19) {
m50.48 <- match(us48.fipsno$"State.name", state.name)
} else{
m50.48 <- match(us48.fipsno$"State_name", state.name)
}
xy <- as.matrix(as.data.frame(state.center))[m50.48,]
llk4.nb <- knn2nb(knearneigh(xy, k=4, longlat=FALSE))
gck4.nb <- knn2nb(knearneigh(xy, k=4, longlat=TRUE))
plot(llk4.nb, xy)
plot(diffnb(llk4.nb, gck4.nb), xy, add=TRUE, col="red", lty=2)
title(main="Differences between Euclidean and Great Circle k=4 neighbours")
summary(llk4.nb, xy, longlat=TRUE, scale=0.5)
summary(gck4.nb, xy, longlat=TRUE, scale=0.5)

xy1 <- SpatialPoints((as.data.frame(state.center))[m50.48,],
                      proj4string=CRS("+proj=longlat +ellps=GRS80"))
gck4a.nb <- knn2nb(knearneigh(xy1, k=4))
summary(gck4a.nb, xy1, scale=0.5)

xy1 <- st_as_sf((as.data.frame(state.center))[m50.48,], coords=1:2,
                 crs=st_crs("+proj=longlat +ellps=GRS80"))
old_use_s2 <- sf_use_s2()
sf_use_s2(TRUE)
system.time(gck4a.nb <- knn2nb(knearneigh(xy1, k=4)))
summary(gck4a.nb, xy1, scale=0.5)
sf_use_s2(FALSE)
system.time(gck4a.nb <- knn2nb(knearneigh(xy1, k=4)))
summary(gck4a.nb, xy1, scale=0.5)
sf_use_s2(old_use_s2)

# https://github.com/r-spatial/spdep/issues/38
if (require("dbscan", quietly=TRUE)) {
  set.seed(1)
x <- cbind(runif(50), runif(50), runif(50))
out <- knearneigh(x, k=5)
knn2nb(out)
try(out <- knearneigh(rbind(x, x[1:10,]), k=5))
}
**knn2nb**  Neighbours list from knn object

Description

The function converts a knn object returned by knearneigh into a neighbours list of class nb with a list of integer vectors containing neighbour region number ids.

Usage

```r
knn2nb(knn, row.names = NULL, sym = FALSE)
```

Arguments

- **knn**: A knn object returned by knearneigh
- **row.names**: character vector of region ids to be added to the neighbours list as attribute `region.id`, default `seq(1,nrow(x))`
- **sym**: force the output neighbours list to symmetry

Value

The function returns an object of class nb with a list of integer vectors containing neighbour region number ids. See `card` for details of “nb” objects.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

See Also

- `knearneigh, card`

Examples

```r
columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
coords <- coordinates(as(columbus, "Spatial"))
col.knn <- knearneigh(coords, k=4)
plot(st_geometry(columbus), border="grey")
plot(knn2nb(col.knn), coords, add=TRUE)
title(main="K nearest neighbours, k = 4")
```
lag.listw

Spatial lag of a numeric vector

Description

Using a listw sparse representation of a spatial weights matrix, compute the lag vector $V_x$

Usage

```r
## S3 method for class 'listw'
lag(x, var, zero.policy=NULL, NAOK=FALSE, ...)
```

Arguments

- `x`: a listw object created for example by `nb2listw`
- `var`: a numeric vector the same length as the neighbours list in listw
- `zero.policy`: default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
- `NAOK`: If 'FALSE', the presence of 'NA' values is regarded as an error; if 'TRUE' then any 'NA' or 'NaN' or 'Inf' values in var are represented as an NA lagged value.
- `...`: additional arguments

Value

a numeric vector the same length as var

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

See Also

`nb2listw`

Examples

data(oldcol)
Vx <- lag.listw(nb2listw(COL.nb, style="W"), COL.OLD$CRIME)
plot(Vx, COL.OLD$CRIME)
plot(ecdf(COL.OLD$CRIME))
plot(ecdf(Vx), add=TRUE, col.points="red", col.hor="red")
is.na(COL.OLD$CRIME[5]) <- TRUE
VxNA <- lag.listw(nb2listw(COL.nb, style="W"), COL.OLD$CRIME, NAOK=TRUE)
Compute Lee’s statistic

Description

A simple function to compute Lee’s L statistic for bivariate spatial data;

\[
L(x, y) = \frac{n}{\sum_{i=1}^{n} (\sum_{j=1}^{n} w_{ij} (x_i - \bar{x})) (\sum_{j=1}^{n} w_{ij} (y_j - \bar{y}))} \sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^{n} (y_i - \bar{y})^2}
\]

Usage

```
lee(x, y, listw, n, S2, zero.policy=NULL, NAOK=FALSE)
```

Arguments

- `x` a numeric vector the same length as the neighbours list in listw
- `y` a numeric vector the same length as the neighbours list in listw
- `listw` a listw object created for example by nb2listw
- `n` number of zones
- `S2` Sum of squared sum of weights by rows.
- `zero.policy` default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
- `NAOK` if 'TRUE' then any 'NA' or 'NaN' or 'Inf' values in x are passed on to the foreign function. If 'FALSE', the presence of 'NA' or 'NaN' or 'Inf' values is regarded as an error.

Value

a list of

- `L` Lee’s L statistic
- `local L` Lee’s local L statistic

Author(s)

Roger Bivand and Virgilio GÃ¡mez-Rubio <Virgilio.Gomez@uclm.es>

References


See Also

`lee.mc`
Examples

```r
data(boston, package="spData")
lw<-nb2listw(boston.soi)

x<-boston.c$CMEDV
y<-boston.c$CRIM
z<-boston.c$RAD

Lxy<-lee(x, y, lw, length(x), zero.policy=TRUE)
Lxz<-lee(x, z, lw, length(x), zero.policy=TRUE)
```

Description

A permutation test for Lee’s L statistic calculated by using nsim random permutations of x and y for the given spatial weighting scheme, to establish the rank of the observed statistic in relation to the nsim simulated values.

Usage

```r
lee.mc(x, y, listw, nsim, zero.policy=NULL, alternative="greater", na.action=na.fail, spChk=NULL, return_boot=FALSE)
```

Arguments

- `x`: a numeric vector the same length as the neighbours list in `listw`
- `y`: a numeric vector the same length as the neighbours list in `listw`
- `listw`: a `listw` object created for example by `nb2listw`
- `nsim`: number of permutations
- `zero.policy`: default `NULL`, use global option value; if `TRUE` assign zero to the lagged value of zones without neighbours, if `FALSE` assign NA
- `alternative`: a character string specifying the alternative hypothesis, must be one of "greater" (default), or "less".
- `na.action`: a function (default `na.fail`), can also be `na.omit` or `na.exclude` - in these cases the weights list will be subsetted to remove NAs in the data. It may be necessary to set `zero.policy` to `TRUE` because this subsetting may create no-neighbour observations. Note that only weights lists created without using the `glist` argument to `nb2listw` may be subsetted. `na.pass` is not permitted because it is meaningless in a permutation test.
- `spChk`: should the data vector names be checked against the spatial objects for identity integrity, `TRUE`, or `FALSE`, default `NULL` to use `get.spChkOption()`
- `return_boot`: return an object of class `boot` from the equivalent permutation bootstrap rather than an object of class `htest`
Value

A list with class htest and mc.sim containing the following components:

- **statistic**: the value of the observed Lee’s L.
- **parameter**: the rank of the observed Lee’s L.
- **p.value**: the pseudo p-value of the test.
- **alternative**: a character string describing the alternative hypothesis.
- **method**: a character string giving the method used.
- **data.name**: a character string giving the name(s) of the data, and the number of simulations.
- **res**: nsim simulated values of statistic, final value is observed statistic.

Author(s)

Roger Bivand, Virgilio GÃ³mez-Rubio <Virgilio.Gomez@uclm.es>

References


See Also

- `lee`

Examples

```r
data(boston, package="spData")
lw<-nb2listw(boston.soi)
x<-boston.c$CMEDV
y<-boston.c$CRIM
lee.mc(x, y, nsim=99, lw, zero.policy=TRUE, alternative="less")
```

# Test with missing values
```r
x[1:5]<-NA
y[3:7]<-NA
lee.mc(x, y, nsim=99, lw, zero.policy=TRUE, alternative="less",
       na.action=na.omit)
```
Lee's L test for spatial autocorrelation

Description

Lee's L test for spatial autocorrelation using a spatial weights matrix in weights list form. The assumptions underlying the test are sensitive to the form of the graph of neighbour relationships and other factors, and results may be checked against those of lee.mc permutations.

Usage

```r
lee.test(x, y, listw, zero.policy=NULL,
    alternative="greater", na.action=na.fail, spChk=NULL)
```

Arguments

- `x`: a numeric vector the same length as the neighbours list in listw
- `y`: a numeric vector the same length as the neighbours list in listw
- `listw`: a listw object created for example by nb2listw
- `zero.policy`: default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
- `alternative`: a character string specifying the alternative hypothesis, must be one of greater (default), less or two.sided.
- `na.action`: a function (default na.fail), can also be na.omit or na.exclude - in these cases the weights list will be subsetted to remove NAs in the data. It may be necessary to set zero.policy to TRUE because this subsetting may create no-neighbour observations. Note that only weights lists created without using the glist argument to nb2listw may be subsetted. If na.pass is used, zero is substituted for NA values in calculating the spatial lag
- `spChk`: should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use get.spChkOption()

Value

A list with class htest containing the following components:

- `statistic`: the value of the standard deviate of Lee's L.
- `p.value`: the p-value of the test.
- `estimate`: the value of the observed Lee's L, its expectation and variance under the method assumption.
- `alternative`: a character string describing the alternative hypothesis.
- `method`: a character string giving the assumption used for calculating the standard deviate.
- `data.name`: a character string giving the name(s) of the data.
Note
See Lee (2004) for details on how the asymptotic expectation and variance of Lee’s L is computed. In particular, check Lee (2004), table 1, page 1690.
This test may fail for large datasets as the computation of the asymptotic expectation and variance requires the use of dense matrices.

Author(s)
Roger Bivand and Virgilio Gómez-Rubio <Virgilio.Gomez@uclm.es>

References

See Also
lee, lee.mc, listw2U

Examples

data(oldcol)
col.W <- nb2listw(COL.nb, style="W")
crime <- COL.OLD$CRIME

lee.test(crime, crime, col.W, zero.policy=TRUE)
#Test with missing values
x<-crime
y<-crime
x[1:5]<-NA
y[3:7]<-NA

lee.test(x, y, col.W, zero.policy=TRUE, na.action=na.omit)
# lee.test(x, y, col.W, zero.policy=TRUE)#Stops with an error

data(boston, package="spData")
lw<-nb2listw(boston.soi)
x<-boston.c$CMEDV
y<-boston.c$CRIM

lee.test(x, y, lw, zero.policy=TRUE, alternative="less")

#Test with missing values
x[1:5]<-NA
y[3:7]<-NA

lee.test(x, y, lw, zero.policy=TRUE, alternative="less", na.action=na.omit)
listw2sn

Spatial neighbour sparse representation

Description

The function makes a "spatial neighbour" object representation (similar to the S-PLUS spatial statistics module representation of a "listw" spatial weights object. sn2listw() is the inverse function to listw2sn(), creating a "listw" object from a "spatial neighbour" object.

Usage

listw2sn(listw)

sn2listw(sn)

Arguments

listw a listw object from for example nb2listw
sn a spatial.neighbour object

Value

listw2sn() returns a data frame with three columns, and with class spatial.neighbour:

from region number id for the start of the link (S-PLUS row.id)
to region number id for the end of the link (S-PLUS col.id)
weights weight for this link

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

See Also

nb2listw
Examples

columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[[1]])
col.listw <- nb2listw(col.gal.nb)
col.listw$neighbours[[1]]
col.listw$weights[[1]]
col.sn <- listw2sn(col.listw)
str(col.sn)

Description

The function reports the estimates of tests chosen among five statistics for testing for spatial dependence in linear models. The statistics are the simple LM test for error dependence (LMerr), the simple LM test for a missing spatially lagged dependent variable (LMlag), variants of these robust to the presence of the other (RLMerr, RLMlag - RLMerr tests for error dependence in the possible presence of a missing lagged dependent variable, RLMlag the other way round), and a portmanteau test (SARMA, in fact LMerr + RLMlag). Note: from spdep 0.3-32, the value of the weights matrix trace term is returned correctly for both underlying symmetric and asymmetric neighbour lists, before 0.3-32, the value was wrong for listw objects based on asymmetric neighbour lists, such as k-nearest neighbours (thanks to Luc Anselin for finding the bug).

Usage

lm.LMtests(model, listw, zero.policy=NULL, test="LMerr", spChk=NULL, naSubset=TRUE)

Arguments

model an object of class lm returned by lm, or optionally a vector of externally calculated residuals (run though na.omit if any NAs present) for use when only "LMerr" is chosen; weights and offsets should not be used in the lm object
listw a listw object created for example by nb2listw, expected to be row-standardised (W-style)
zero.policy default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
test a character vector of tests requested chosen from LMerr, LMlag, RLMerr, RLMlag, SARMA; test="all" computes all the tests.
should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use get.spChkOption()

default TRUE to subset listw object for omitted observations in model object (this is a change from earlier behaviour, when the model$na.action component was ignored, and the listw object had to be subetted by hand)

object to be printed

a character string specifying the probability value adjustment (see p.adjust) for multiple tests, default "none"

minimum number of significant digits to be used for most numbers

printing arguments to be passed through

The two types of dependence are for spatial lag \( \rho \) and spatial error \( \lambda \):

\[
y = X\beta + \rho W_{(1)} y + u, \\
u = \lambda W_{(2)} u + e
\]

where \( e \) is a well-behaved, uncorrelated error term. Tests for a missing spatially lagged dependent variable test that \( \rho = 0 \), tests for spatial autocorrelation of the error \( u \) test whether \( \lambda = 0 \). \( W \) is a spatial weights matrix; for the tests used here they are identical.

A list of class LMtestlist of htest objects, each with:

- statistic: the value of the Lagrange Multiplier test.
- parameter: number of degrees of freedom
- p.value: the p-value of the test.
- method: a character string giving the method used.
- data.name: a character string giving the name(s) of the data.

Roger Bivand <Roger.Bivand@nhh.no> and Andrew Bernat

Examples

```r
data(oldcol)
oldcrime.lm <- lm(CRIME ~ HOVAL + INC, data = COL.OLD)
summary(oldcrime.lm)
res <- lm.LMtests(oldcrime.lm, nb2listw(COL.nb), test=c("LMerr", "LMlag", "RLMerr", "RLMlag", "SARMA"))
summary(res)
lm.LMtests(oldcrime.lm, nb2listw(COL.nb))
lm.LMtests(residuals(oldcrime.lm), nb2listw(COL.nb))
```

---

**lm.morantest**

*Moran’s I test for residual spatial autocorrelation*

**Description**

Moran’s I test for spatial autocorrelation in residuals from an estimated linear model (`lm()`). The helper function `listw2U()` constructs a weights list object corresponding to the sparse matrix $\frac{1}{2}(W + W')$

**Usage**

```r
lm.morantest(model, listw, zero.policy=NULL, alternative = "greater", spChk=NULL, resfun=weighted.residuals, na Subset=TRUE)
listw2U(listw)
```

**Arguments**

- `model`: an object of class `lm` returned by `lm`; weights may be specified in the `lm` fit, but offsets should not be used
- `listw`: a `listw` object created for example by `nb2listw`
- `zero.policy`: default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
- `alternative`: a character string specifying the alternative hypothesis, must be one of "greater" (default), "less" or "two.sided".
- `spChk`: should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use `get.spChkOption()`
- `resfun`: default: `weighted.residuals`; the function to be used to extract residuals from the `lm` object, may be `residuals`, `weighted.residuals`, `rstandard`, or `rstudent`
- `naSubset`: default TRUE to subset `listw` object for omitted observations in model object (this is a change from earlier behaviour, when the `model$na.action` component was ignored, and the `listw` object had to be subsetted by hand)
Value

A list with class htest containing the following components:

- **statistic**: the value of the standard deviate of Moran's I.
- **p.value**: the p-value of the test.
- **estimate**: the value of the observed Moran's I, its expectation and variance under the method assumption.
- **alternative**: a character string describing the alternative hypothesis.
- **method**: a character string giving the method used.
- **data.name**: a character string giving the name(s) of the data.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References


See Also

- `lm.LMtests,lm`

Examples

data(oldcol)
oldcrime1.lm <- lm(CRIME ~ 1, data = COL.OLD)
oldcrime.lm <- lm(CRIME ~ HOVAL + INC, data = COL.OLD)
lm.morantest(oldcrime.lm, nb2listw(COL.nb, style="W"))
lm.LMtests(oldcrime.lm, nb2listw(COL.nb, style="W"))
lm.morantest(oldcrime.lm, nb2listw(COL.nb, style="S"))
lm.morantest(oldcrime1.lm, nb2listw(COL.nb, style="W"))
moran.test(COL.OLD$CRIME, nb2listw(COL.nb, style="W"),
randomisation=FALSE)
oldcrime.wlm <- lm(CRIME ~ HOVAL + INC, data = COL.OLD,
weights = I(1/AREA_PL))
lm.morantest(oldcrime.wlm, nb2listw(COL.nb, style="W"),
resfun=weighted.residuals)
lm.morantest(oldcrime.wlm, nb2listw(COL.nb, style="W"),
resfun=rstudent)
lm.morantest.exact

Exact global Moran's I test

Description

The function implements Tiefelsdorf's exact global Moran's I test.

Usage

lm.morantest.exact(model, listw, zero.policy = NULL, alternative = "greater",
spChk = NULL, resfun = weighted.residuals, zero.tol = 1e-07, Omega=NULL,
save.M=NULL, save.U=NULL, useTP=FALSE, truncErr=1e-6, zeroTreat=0.1)
## S3 method for class 'moranex'
print(x, ...) 

Arguments

model an object of class lm returned by lm; weights may be specified in the lm fit, but
offsets should not be used
listw a listw object created for example by nb2listw
zero.policy default NULL, use global option value; if TRUE assign zero to the lagged value
of zones without neighbours, if FALSE assign NA
alternative a character string specifying the alternative hypothesis, must be one of greater
(default), less or two.sided.
spChk should the data vector names be checked against the spatial objects for identity
integrity, TRUE, or FALSE, default NULL to use get.spChkOption()
resfun default: weighted.residuals; the function to be used to extract residuals from the
lm object, may be residuals, weighted.residuals, rstandard, or rstudent
zero.tol tolerance used to find eigenvalues close to absolute zero
Omega A SAR process matrix may be passed in to test an alternative hypothesis, for ex-
ample Omega <- invIrW(listw,rho=0.1); Omega <- tcrossprod(Omega), chol() is taken internally
save.M return the full M matrix for use in spdep:::exactMoranAlt
save.U return the full U matrix for use in spdep:::exactMoranAlt
useTP default FALSE, if TRUE, use truncation point in integration rather than up-
per=Inf, see Tiefelsdorf (2000), eq. 6.7, p.69
truncErr when useTP=TRUE, pass truncation error to truncation point function
zeroTreat when useTP=TRUE, pass zero adjustment to truncation point function
x a moranex object
... arguments to be passed through
lm.morantest.exact

Value

A list of class `moranex` with the following components:

- **statistic**: the value of the saddlepoint approximation of the standard deviate of global Moran’s I.
- **p.value**: the p-value of the test.
- **estimate**: the value of the observed global Moran’s I.
- **method**: a character string giving the method used.
- **alternative**: a character string describing the alternative hypothesis.
- **gamma**: eigenvalues (excluding zero values)
- **oType**: usually set to "E"
- **data.name**: a character string giving the name(s) of the data.
- **df**: degrees of freedom

Author(s)

Markus Reder and Roger Bivand

References


See Also

- `lm.morantest.sad`

Examples

eire <- st_read(system.file("shapes/eire.shp", package="spData")[1])
row.names(eire) <- as.character(eire$names)
st_crs(eire) <- "+proj=utm +zone=30 +ellps=airy +units=km"
eire.nb <- poly2nb(eire)
e.lm <- lm(OWNCONS ~ ROADACC, data=eire)
lm.morantest(e.lm, nb2listw(eire.nb))
lm.morantest.sad(e.lm, nb2listw(eire.nb))
lm.morantest.exact(e.lm, nb2listw(eire.nb))
lm.morantest.exact(e.lm, nb2listw(eire.nb), useTP=TRUE)
Saddlepoint approximation of global Moran’s I test

Description

The function implements Tiefelsdorf’s application of the Saddlepoint approximation to global Moran’s I’s reference distribution.

Usage

```r
lm.morantest.sad(model, listw, zero.policy=NULL, alternative="greater",
    spChk=NULL, resfun=weighted.residuals, tol=.Machine$double.eps^0.5,
    maxiter=1000, tol.bounds=0.0001, zero.tol = 1e-07, Omega=NULL,
    save.M=NULL, save.U=NULL)
```

Arguments

- `model`: an object of class `lm` returned by `lm`; weights may be specified in the `lm` fit, but offsets should not be used
- `listw`: a `listw` object created for example by `nb2listw`
- `zero.policy`: default `NULL`, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
- `alternative`: a character string specifying the alternative hypothesis, must be one of greater (default), less or two.sided.
- `spChk`: should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default `NULL` to use `get.spChkOption()`
- `resfun`: default: `weighted.residuals`; the function to be used to extract residuals from the `lm` object, may be `residuals`, `weighted.residuals`, `rstandard`, or `rstudent`
- `tol`: the desired accuracy (convergence tolerance) for `uniroot`
- `maxiter`: the maximum number of iterations for `uniroot`
- `tol.bounds`: offset from bounds for `uniroot`
- `zero.tol`: tolerance used to find eigenvalues close to absolute zero
- `Omega`: A SAR process matrix may be passed in to test an alternative hypothesis, for example `Omega <- invIrW(listw, rho=0.1); Omega <- tcrossprod(Omega), chol()` is taken internally
- `save.M`: return the full M matrix for use in `spdep:::exactMoranAlt`
- `save.U`: return the full U matrix for use in `spdep:::exactMoranAlt`
Details

The function involves finding the eigenvalues of an \( n \) by \( n \) matrix, and numerically finding the root for the Saddlepoint approximation, and should therefore only be used with care when \( n \) is large.

Value

A list of class `moransad` with the following components:

- `statistic`: the value of the saddlepoint approximation of the standard deviate of global Moran's I.
- `p.value`: the p-value of the test.
- `estimate`: the value of the observed global Moran’s I.
- `alternative`: a character string describing the alternative hypothesis.
- `method`: a character string giving the method used.
- `data.name`: a character string giving the name(s) of the data.
- `internal1`: Saddlepoint omega, r and u
- `internal2`: f.root, iter and estim.prec from uniroot
- `df`: degrees of freedom
- `tau`: eigenvalues (excluding zero values)

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References


See Also

`lm.morantest`

Examples

eire <- st_read(system.file("shapes/eire.shp", package="spData")[1])
row.names(eire) <- as.character(eire$names)
st_crs(eire) <- "+proj=utm +zone=30 +ellps=airy +units=km"
eire.nb <- poly2nb(eire)
e.lm <- lm(OWNCONS ~ ROADACC, data=eire)
localC

**Compute Local Geary statistic**

**Description**

The Local Geary is a local adaptation of Geary's C statistic of spatial autocorrelation. The Local Geary uses squared differences to measure dissimilarity unlike the Local Moran. Low values of the Local Geary indicate positive spatial autocorrelation and large refers to negative spatial autocorrelation.

Inference for the Local Geary is based on a permutation approach which compares the observed value to the reference distribution under spatial randomness. `localC_perm()` returns a pseudo p-value. This is not an analytical p-value and is based on the number of permutations and as such should be used with care.

**Usage**

```r
localC(x, ..., zero.policy=NULL)
```

## Default S3 method:
`localC(x, listw, ..., zero.policy=NULL)`

## S3 method for class 'formula'
`localC(formula, data, listw, ..., zero.policy=NULL)`

## S3 method for class 'list'
`localC(x, listw, ..., zero.policy=NULL)`

## S3 method for class 'matrix'
`localC(x, listw, ..., zero.policy=NULL)`

## S3 method for class 'data.frame'
`localC(x, listw, ..., zero.policy=NULL)`

`localC_perm(x, ..., zero.policy=NULL, iseed=NULL)`

## Default S3 method:
`localC_perm(x, listw, nsim = 499, alternative = "two.sided", ..., zero.policy=NULL, iseed=NULL)`

## S3 method for class 'formula'
`localC_perm(x, listw, nsim = 499, alternative = "two.sided", ..., zero.policy=NULL, iseed=NULL)`
```
localC_perm(formula, data, listw, nsim = 499, alternative = "two.sided", ..., zero.policy=NULL, iseed=NULL)

**Arguments**

- **x**: a numeric vector, numeric matrix, or list. See details for more.
- **formula**: A one-sided formula determining which variables to be used.
- **listw**: a listw object created for example by nb2listw.
- **data**: Used when a formula is provided. A matrix or data frame containing the variables in the formula `formula`.
- **nsim**: The number of simulations to be used for permutation test.
- **alternative**: A character defining the alternative hypothesis. Must be one of "two.sided", "less" or "greater".
- **...**: other arguments passed to methods.
- **zero.policy**: default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA.
- **iseed**: default NULL, used to set the seed for possible parallel RNGs

**Details**

The Local Geary can be extended to a multivariate context. When `x` is a numeric vector, the univariate Local Geary will be calculated. To calculate the multivariate Local Moran provide either a list or a matrix. When `x` is a list, each element must be a numeric vector of the same length and of the same length as the neighbours in `listw`. In the case that `x` is a matrix the number of rows must be the same as the length of the neighbours in `listw`.

While not required in the univariate context, the standardized Local Geary is calculated. The multivariate Local Geary is always standardized.

The univariate Local Geary is calculated as \[ c_i = \sum_j w_{ij}(x_i - x_j)^2 \] and the multivariate Local Geary is calculated as \[ c_{k,i} = \sum_{v=1}^{k} c_{v,i} \] as described in Anselin (2019).

**Value**

A numeric vector containing Local Geary statistic with attribute `pseudo-p` when `localC_perm()` is used. `pseudo-p` is an 8 column matrix containing

- **E.Ci**: expectation of the Local Geary statistic based on permutation sample
- **Var.Ci**: variance of Local Geary based on permutation sample
- **Z.Ci**: standard deviate of Local Geary based on permutation sample
- **Pr()**: p-value of Local Geary statistic using `pnorm()` using standard deviates based on permutation sample means and standard deviances
- **Pr() Sim**: rank() and `punif()` of observed statistic rank for [0, 1] p-values using alternative=
- **Pr(folded) Sim**: the simulation folded [0, 0.5] range ranked p-value (based on [link](https://github.com/pysal/esda/blob/4a63e0b5df1e754b17b5f1205b8cadbecc5e061/esda/crand.py#L211-L213))
Skewness  
the output of `e1071::skewness()` for the permutation samples underlying the standard deviates

Kurtosis  
the output of `e1071::kurtosis()` for the permutation samples underlying the standard deviates

Author(s)
Josiah Parry <josiah.parry@gmail.com> and Roger Bivand

References


Examples
orig <- spData::africa.rook.nb
listw <- nb2listw(orig)
x <- spData::afcon$totcon

(A <- localC(x, listw))

listw1 <- nb2listw(droplinks(sym.attr.nb(orig), 3, sym=TRUE), zero.policy=TRUE)
(A1 <- localC(x, listw1, zero.policy=FALSE))
(A2 <- localC(x, listw1, zero.policy=TRUE))

run <- FALSE
if (require(rgeoda, quietly=TRUE)) run <- TRUE
if (run) {
  W <- create_weights(as.numeric(length(x)))
  for (i in 1:length(listw$neighbours)) {
    set_neighbors_with_weights(W, i, listw$neighbours[[i]], listw$weights[[i]])
    update_weights(W)
  }
  set.seed(1)
  B <- local_geary(W, data.frame(x))
  all.equal(A, lisa_values(B))
}

if (run) {
  set.seed(1)
  C <- localC_perm(x, listw, nsim = 499, conditional=TRUE, alternative="two.sided")
  cor(ifelse(lisa_pvalues(B) < 0.5, lisa_pvalues(B), 1-lisa_pvalues(B)),
      attr(C, "pseudo-p"), [6])
}
# pseudo-p values probably wrongly folded https://github.com/GeoDaCenter/rgeoda/issues/28
## Not run:
tmap_ok <- FALSE
if (require(tmap, quietly=TRUE)) tmap_ok <- TRUE
if (run) {
  # doi: 10.1111/gean.12164
guerry_path <- system.file("extdata", "Guerry.shp", package = "rgeoda")
g <- st_read(guerry_path)[, 7:12]
cor(st_drop_geometry(g)) #(Tab. 1)
lw <- nb2listw(poly2nb(g))
moran(g$Crm_prs, lw, n=nrow(g), S0=Szero(lw))$I
moran(g$Crm_prp, lw, n=nrow(g), S0=Szero(lw))$I
moran(g$Litercy, lw, n=nrow(g), S0=Szero(lw))$I
moran(g$Donatns, lw, n=nrow(g), S0=Szero(lw))$I
moran(g$Infants, lw, n=nrow(g), S0=Szero(lw))$I
moran(g$Suicids, lw, n=nrow(g), S0=Szero(lw))$I
}
if (run) {
o <- prcomp(st_drop_geometry(g), scale.=TRUE)
cor(st_drop_geometry(g), o$x[,1:2])^2 #(Tab. 2)
}
if (run) {
gPC1 <- o$X[, "PC1"]
brks <- c(min(gPC1), natural_breaks(k=6, g["PC1"]), max(gPC1))
if (tmap.ok) tm_shape(g) + tm_fill("PC1", breaks=brks, midpoint=0) +
  tmBorders() # Fig. 1
else pplot(g["PC1"], breaks=brks)
}
if (run) {
gPC2 <- -1*o$X[, "PC2"] # eigenvalue sign arbitrary
brks <- c(min(gPC2), natural_breaks(k=6, g["PC2"]), max(gPC2))
if (tmap.ok) tm_shape(g) + tm_fill("PC2", breaks=brks, midpoint=0) +
  tmBorders() # Fig. 2
else plot(g["PC2"], breaks=brks)
}
if (run) {
w <- queen_weights(g)
lm_PC1 <- local_moran(w, g["PC1"], significance_cutoff=0.01,
  permutations=99999)
g$lm_PC1 <- factor(lisa_clusters(lm_PC1), levels=0:4,
  labels=lisa_labels(lm_PC1)[1:5])
if (tmap.ok) tm_shape(g) + tm_fill("lm_PC1", textNA="Insignificant",
  colorNA="gray95") + tmBorders() # Fig. 3
else plot(g["lm_PC1"])
}
if (run) {
set.seed(1)
lm_PC1_spdep <- localmoran_perm(g$PC1, lw, nsim=9999)
q <- attr(lm_PC1_spdep, "quadr")$pysal
if (is.na(g$lm_PC1_spdep)) <- g$lm_PC1_spdep[,6] > 0.02 # note folded p-values
if (tmap.ok) tm_shape(g) + tm_fill("lm_PC1_spdep", textNA="Insignificant",
  colorNA="gray95") + tmBorders() # rep. Fig. 3
else plot(g["lm_PC1_spdep"])
}
if (run) {
localC

```r
lg_PC1 <- local_g(w, g["PC1"], significance_cutoff=0.01, permutations=99999)
g$lg_PC1 <- factor(lisa_clusters(lg_PC1), levels=0:2, labels=lisa_labels(lg_PC1)[0:3])
is.na(g$lg_PC1) <- g$lg_PC1 == "Not significant"
g$lg_PC1 <- droplevels(g$lg_PC1)
if (tmmap_ok) tm_shape(g) + tm_fill("lg_PC1", textNA="Insignificant", colorNA="gray95") + tm_borders() # Fig. 4 (wrong)
else plot(g["lg_PC1"])
g$lg_PC1a <- cut(g$PC1, c(-Inf, mean(g$PC1), Inf), labels=c("Low", "High"))
is.na(g$lg_PC1a) <- lisa_pvalues(lg_PC1) >= 0.01
g$lg_PC1a <- droplevels(g$lg_PC1a)
if (tmmap_ok) tm_shape(g) + tm_fill("lg_PC1", textNA="Insignificant", colorNA="gray95") + tm_borders() # Fig. 4 (guess)
else plot(g["lg_PC1"])
}
if (run) {
  lc_PC1 <- local_geary(w, g["PC1"], significance_cutoff=0.01, permutations=99999)
g$lc_PC1 <- factor(lisa_clusters(lc_PC1), levels=0:4, labels=lisa_labels(lc_PC1)[1:5])
is.na(g$lc_PC1) <- g$lc_PC1 == "Not significant"
g$lc_PC1 <- droplevels(g$lc_PC1)
if (tmmap_ok) tm_shape(g) + tm_fill("lc_PC1", textNA="Insignificant", colorNA="gray95") + tm_borders() # Fig. 5
else plot(g["lc_PC1"])
}
if (run) {
  set.seed(1)
  system.time(lc_PC1_spdep <- localC_perm(g$PC1, lw, nsim=99999, alternative="two.sided"))
}
if (run) {
  if (require(parallel, quietly=TRUE)) {
    ncpus <- detectCores()-1L
    cores <- get.coresOption()
    set.coresOption(ncpus)
    system.time(lmc_PC1_spdep1 <- localC_perm(g$PC1, lw, nsim=99999, alternative="two.sided", iseed=1))
    set.coresOption(cores)
  }
}
if (run) {
  g$lc_PC1_spdep <- attr(lc_PC1_spdep, "cluster")
is.na(g$lc_PC1_spdep) <- attr(lc_PC1_spdep, "pseudo-p")[,6] > 0.01
  g$lc_PC1_spdep <- droplevels(g$lc_PC1_spdep)
  if (tmmap_ok) tm_shape(g) + tm_fill("lc_PC1_spdep", textNA="Insignificant", colorNA="gray95") + tm_borders() # rep. Fig. 5
  else plot(g["lc_PC1_spdep"])
}
if (run) {
  g$both_PC1 <- interaction(g$lc_PC1, g$lm_PC1)
g$both_PC1 <- droplevels(g$both_PC1)
```
if (tmap_ok) tm_shape(g) + tm_fill("both_PC1", textNA="Insignificant", colorNA="gray95") + tm_borders() # Fig. 6} else plot(g["both_PC1"])
}
if (run) {
lc005_PC1 <- local_geary(w, g["PC1"], significance_cutoff=0.005, permutations=99999)
g$lc005_PC1 <- factor(lisa_clusters(lc005_PC1), levels=0:4, labels=lisa_labels(lc005_PC1)[1:5])
is.na(g$lc005_PC1) <- g$lc005_PC1 == "Not significant"
g$lc005_PC1 <- droplevels(g$lc005_PC1)
if (tmap_ok) tm_shape(g) + tm_fill("lc005_PC1", textNA="Insignificant", colorNA="gray95") + tm_borders() # Fig. 7} else plot(g["lc005_PC1"])
}
if (run) {
g$lc005_PC1_spdep <- attr(lc_PC1_spdep, "cluster")
is.na(g$lc005_PC1_spdep) <- attr(lc_PC1_spdep, "pseudo-p")[,6] > 0.005
g$lc005_PC1_spdep <- droplevels(g$lc005_PC1_spdep)
if (tmap_ok) tm_shape(g) + tm_fill("lc005_PC1_spdep", textNA="Insignificant", colorNA="gray95") + tm_borders() # rep. Fig. 7} else plot(g["lc005_PC1_spdep"])
}
if (run) {
lc001_PC1 <- local_geary(w, g["PC1"], significance_cutoff=0.001, permutations=99999)
g$lc001_PC1 <- factor(lisa_clusters(lc001_PC1), levels=0:4, labels=lisa_labels(lc001_PC1)[1:5])
is.na(g$lc001_PC1) <- g$lc001_PC1 == "Not significant"
g$lc001_PC1 <- droplevels(g$lc001_PC1)
if (tmap_ok) tm_shape(g) + tm_fill("lc001_PC1", textNA="Insignificant", colorNA="gray95") + tm_borders() # Fig. 8} else plot(g["lc001_PC1"])
}
if (run) {
g$lc001_PC1_spdep <- attr(lc_PC1_spdep, "cluster")
is.na(g$lc001_PC1_spdep) <- attr(lc_PC1_spdep, "pseudo-p")[,6] > 0.001
g$lc001_PC1_spdep <- droplevels(g$lc001_PC1_spdep)
if (tmap_ok) tm_shape(g) + tm_fill("lc001_PC1_spdep", textNA="Insignificant", colorNA="gray95") + tm_borders() # rep. Fig. 8} else plot(g["lc001_PC1_spdep"])
}
if (run) {
lc_PC2 <- local_geary(w, g["PC2"], significance_cutoff=0.01, permutations=99999)
g$lc_PC2 <- factor(lisa_clusters(lc_PC2), levels=0:4, labels=lisa_labels(lc_PC2)[1:5])
is.na(g$lc_PC2) <- g$lc_PC2 == "Not significant"
g$lc_PC2 <- droplevels(g$lc_PC2)
if (tmap_ok) tm_shape(g) + tm_fill("lc_PC2", textNA="Insignificant", colorNA="gray95") + tm_borders() # Fig. 9} else plot(g["lc_PC2"])
}
if (run) {
    lmc_PC <- local_multigeary(w, g[c("PC1","PC2")], significance_cutoff=0.00247,
        permutations=99999)
    g$lmc_PC <- factor(lisa_clusters(lmc_PC), levels=0:1,
        labels=lisa_labels(lmc_PC)[1:2])
    is.na(g$lmc_PC) <- g$lmc_PC == "Not significant"
    g$lmc_PC <- droplevels(g$lmc_PC)
    table(interaction((p.adjust(lisa_pvalues(lmc_PC), "fdr") < 0.01), g$lmc_PC))
}
if (run) {
    if (tmap_ok) tm_shape(g) + tm_fill("lmc_PC", textNA="Insignificant",
        colorNA="gray95") + tm_borders() # Fig. 10
    else plot(g["lmc_PC"])}
if (run) {
    set.seed(1)
    lmc_PC_spdep <- localC_perm(g[c("PC1","PC2")], lw, nsim=99999, alternative="two.sided")
    all.equal(lisa_values(lmc_PC), c(lmc_PC_spdep))
}
if (run) {
    cor(attr(lmc_PC_spdep, "pseudo-p")[,6], lisa_pvalues(lmc_PC))
}
if (run) {
    g$lmc_PC_spdep <- attr(lmc_PC_spdep, "cluster")
    is.na(g$lmc_PC_spdep) <- p.adjust(attr(lmc_PC_spdep, "pseudo-p")[,6], "fdr") > 0.01
    g$lmc_PC_spdep <- droplevels(g$lmc_PC_spdep)
    if (tmap_ok) tm_shape(g) + tm_fill("lmc_PC_spdep", textNA="Insignificant",
        colorNA="gray95") + tm_borders() # rep. Fig. 10
    else plot(g["lmc_PC_spdep"])}
if (run) {
    lmc_vars <- local_multigeary(w, st_drop_geometry(g)[, 1:6],
        significance_cutoff=0.00247, permutations=99999)
    g$lmc_vars <- factor(lisa_clusters(lmc_vars), levels=0:1,
        labels=lisa_labels(lmc_vars)[1:2])
    is.na(g$lmc_vars) <- g$lmc_vars == "Not significant"
    g$lmc_vars <- droplevels(g$lmc_vars)
    table(interaction((p.adjust(lisa_pvalues(lmc_vars), "fdr") < 0.01),
        g$lmc_vars))
}
if (run) {
    if (tmap_ok) tm_shape(g) + tm_fill("lmc_vars", textNA="Insignificant",
        colorNA="gray95") + tm_borders() # Fig. 11
    else plot(g["lmc_vars"])}
if (run) {
    set.seed(1)
    system.time(lmc_vars_spdep <- localC_perm(st_drop_geometry(g)[, 1:6], lw,
        nsim=99999, alternative="two.sided")
    }
if (run) {
    all.equal(lisa_values(lmc_vars), c(lmc_vars_spdep))
}
if (run) {
  cor(attr(lmc_vars_spdep, "pseudo-p")[,6], lisa_pvalues(lmc_vars))
}

if (run) {
  if (require(parallel, quietly=TRUE)) {
    ncpus <- detectCores()-1L
    cores <- get.coresOption()
    set.coresOption(ncpus)
    system.time(lmc_vars_spdep1 <- localC_perm(st_drop_geometry(g)[, 1:6], lw,
      nsim=9999, alternative="two.sided", iseed=1))
    set.coresOption(cores)
  }
}

if (run) {
  all.equal(lisa_values(lmc_vars), c(lmc_vars_spdep1))
}

if (run) {
  cor(attr(lmc_vars_spdep1, "pseudo-p")[,6], lisa_pvalues(lmc_vars))
}

if (run) {
  g$lmc_vars_spdep <- attr(lmc_vars_spdep1, "cluster")
  is.na(g$lmc_vars_spdep) <- p.adjust(attr(lmc_vars_spdep1, "pseudo-p")[,6], "fdr") > 0.01
  g$lmc_vars_spdep <- droplevels(g$lmc_vars_spdep)
  if (tmap_ok) tm_shape(g, textNA="Insignificant", colorNA="gray95") + tm_borders() # rep. Fig. 11
  else plot(g["lmc_vars_spdep")
}

## End(Not run)
## Not run:
library(reticulate)
use_python("/usr/bin/python", required = TRUE)
gp <- import("geopandas")
ps <- import("libpysal")
W <- listw2mat(listw)
w <- ps$weights$full2W(W, rownames(W))
w$transform <- "R"
esda <- import("esda")
LM <- esda$Moran_Local(x, w)
all.equal(unname(localmoran(x, listw, mlvar=FALSE)[,1]), c(LM$Is))
# confirm x and w the same
LC <- esda$Geary_Local(connectivity=w)$fit(scale(x))
# np$std missing ddof=1
n <- length(x)
D0 <- spdep:::geary.intern((x - mean(x)) / sqrt(var(x)*n/(n-1)), listw, n=n)
# LC components probably wrongly ordered https://github.com/pysal/esda/issues/192
o <- match(round(D0, 6), round(LC$localG, 6))
all.equal(c(LC$localG[o], D0)
# simulation order not retained
LC$p_sim[o]
attr(C, "pseudo-p")[,6]
Description

The local spatial statistic G is calculated for each zone based on the spatial weights object used. The value returned is a Z-value, and may be used as a diagnostic tool. High positive values indicate the possibility of a local cluster of high values of the variable being analysed, very low relative values a similar cluster of low values. For inference, a Bonferroni-type test is suggested in the references, where tables of critical values may be found (see also details below).

Usage

localG(x, listw, zero.policy=NULL, spChk=NULL, return_internals=FALSE, GeoDa=FALSE, alternative = "two.sided")
localG_perm(x, listw, nsim=499, zero.policy=NULL, spChk=NULL, return_internals=TRUE, alternative = "two.sided", iseed=NULL)

Arguments

x a numeric vector the same length as the neighbours list in listw

listw a listw object created for example by nb2listw

zero.policy default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA

spChk should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use get.spChkOption()

return_internals default FALSE but TRUE for permutation version, if TRUE, return internal values of G, EI and VG as as attribute matrix

GeoDa default FALSE, if TRUE, drop x values for no-neighbour and self-neighbour only observations from all summations

nsim default 499, number of conditional permutation simulations

alternative a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less".

iseed default NULL, used to set the seed for possible parallel RNGs

Details

If the neighbours member of listw has a "self.included" attribute set to TRUE, the Gstar variant, including the self-weight \(w_{ii} > 0\), is calculated and returned. The returned vector will have a "gstari" attribute set to TRUE. Self-weights can be included by using the include.self function before converting the neighbour list to a spatial weights list with nb2listw as shown below in the example.

The critical values of the statistic under assumptions given in the references for the 95th percentile are for n=1: 1.645, n=50: 3.083, n=100: 3.289, n=1000: 3.886.
Value
A vector of G or Gstar values, with attributes "gstari" set to TRUE or FALSE, "call" set to the function call, and class "localG".

Note
Conditional permutations added for comparative purposes; permutations are over the whole data vector omitting the observation itself.

Author(s)
Roger Bivand <Roger.Bivand@nhh.no>

References

Examples
data(getisord, package="spData")
# spData 0.3.2 changes x, y, xyz object names to go_x, go_y, go_xyz to
# avoid putting these objects into the global environment via lazy loading
if (exists("go_xyz") && packageVersion("spData") >= "0.3.2") {
  xyz <- go_xyz
  x <- go_x
  y <- go_y
}
xycoords <- cbind(xyz$x, xyz$y)
nb30 <- dnearest(xycoords, 0, 30)
G30 <- localG(xyz$val, nb2listw(nb30, style="B"))
G30[length(xyz$val)-136]
set.seed(1)
G30_sim <- localG_perm(xyz$val, nb2listw(nb30, style="B"))
G30_sim[length(xyz$val)-136]
nb60 <- dnearest(xycoords, 0, 60)
G60 <- localG(xyz$val, nb2listw(nb60, style="B"))
G60[length(xyz$val)-136]
nb90 <- dnearest(xycoords, 0, 90)
G90 <- localG(xyz$val, nb2listw(nb90, style="B"))
G90[length(xyz$val)-136]
nb120 <- dnearest(xycoords, 0, 120)
G120 <- localG(xyz$val, nb2listw(nb120, style="B"))
G120[length(xyz$val)-136]
nb150 <- dnearest(xycoords, 0, 150)
G150 <- localG(xyz$val, nb2listw(nb150, style="B"))
G150[length(xyz$val)-136]
brks <- seq(-5,5,1)
cm.col <- cm.colors(length(brks)-1)
image(x, y, t(matrix(G30, nrow=16, ncol=16, byrow=TRUE)),
    breaks=brks, col=cm.col, asp=1)
text(xyz$x, xyz$y, round(G30, digits=1), cex=0.7)
polygon(c(195,225,225,195), c(195,195,225,225), lwd=2)
title(main=expression(paste("Values of the ", G[i], " statistic")))
G30s <- localG(xyz$val, nb2listw(include.self(nb30),
    style="B"))
cat("value according to Getis and Ord's eq. 14.2, p. 263 (1996)\n")
G30s[length(xyz$val)-136]
cat(paste("value given by Getis and Ord (1996), p. 267,\n    "(division by n-1 rather than n \n in variance)\n")
G30s[length(xyz$val)-136] *
    (sqrt(sum(scale(xyz$val, scale=FALSE)^2)/length(xyz$val)) /
    sqrt(var(xyz$val)))
image(x, y, t(matrix(G30s, nrow=16, ncol=16, byrow=TRUE)),
    breaks=brks, col=cm.col, asp=1)
text(xyz$x, xyz$y, round(G30s, digits=1), cex=0.7)
polygon(c(195,225,225,195), c(195,195,225,225), lwd=2)
title(main=expression(paste("Values of the ", G[i]^"*", " statistic")))

localmoran

Local Moran's I statistic

Description

The local spatial statistic Moran's I is calculated for each zone based on the spatial weights object used. The values returned include a Z-value, and may be used as a diagnostic tool. The statistic is:

\[ I_i = \frac{(x_i - \bar{x})}{\sum_{k=1}^{n}(x_k - \bar{x})^2/(n-1)} \sum_{j=1}^{n} w_{ij}(x_j - \bar{x}) \]

, and its expectation and variance were given in Anselin (1995), but those from Sokal et al. (1998) are implemented here.

Usage

localmoran(x, listw, zero.policy=NULL, na.action=na.fail, conditional=TRUE,
    alternative = "two.sided", mlvar=TRUE,
    spChk=NULL, adjust.x=FALSE)
localmoran_perm(x, listw, nsim=499, zero.policy=NULL, na.action=na.fail,
    alternative = "two.sided", mlvar=TRUE,
    spChk=NULL, adjust.x=FALSE, sample_Ei=TRUE, iseed=NULL)

Arguments

x a numeric vector the same length as the neighbours list in listw
listw a listw object created for example by nb2listw
zero.policy  default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA

na.action  a function (default na.fail), can also be na.omit or na.exclude - in these cases the weights list will be subsetted to remove NAs in the data. It may be necessary to set zero.policy to TRUE because this subsetting may create no-neighbour observations. Note that only weights lists created without using the glist argument to nb2listw may be subsetted. If na.pass is used, zero is substituted for NA values in calculating the spatial lag. (Note that na.exclude will only work properly starting from R 1.9.0, na.omit and na.exclude assign the wrong classes in 1.8.*)

conditional  default TRUE: expectation and variance are calculated using the conditional randomization null (Sokal 1998 Eqs. A7 & A8). Elaboration of these changes available in Sauer et al. (2021). If FALSE: expectation and variance are calculated using the total randomization null (Sokal 1998 Eqs. A3 & A4).

alternative  a character string specifying the alternative hypothesis, must be one of greater, less or two.sided (default).

mlvar  default TRUE: values of local Moran’s I are reported using the variance of the variable of interest (sum of squared deviances over n), but can be reported as the sample variance, dividing by (n-1) instead; both are used in other implementations.

spChk  should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use get.spChkOption()

adjust.x  default FALSE, if TRUE, x values of observations with no neighbours are omitted in the mean of x

nsim  default 499, number of conditional permutation simulations

sample_Ei  default TRUE; if conditional permutation, use the sample $E_i$ values, or the analytical values, leaving only variances calculated by simulation.

iseed  default NULL, used to set the seed for possible parallel RNGs

Details
The values of local Moran’s I are divided by the variance (or sample variance) of the variable of interest to accord with Table 1, p. 103, and formula (12), p. 99, in Anselin (1995), rather than his formula (7), p. 98. The variance of the local Moran statistic is taken from Sokal et al. (1998) p. 334, equations 4 & 5 or equations 7 & 8 located depending on user specification. By default, the implementation divides by n, not (n-1) in calculating the variance and higher moments. Conditional code contributed by Jeff Sauer and Levi Wolf.

Value

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ii</td>
<td>local moran statistic</td>
</tr>
<tr>
<td>E.Ii</td>
<td>expectation of local moran statistic; for localmoran_perm the permutation sample means</td>
</tr>
<tr>
<td>Var.Ii</td>
<td>variance of local moran statistic; for localmoran_perm the permutation sample standard deviations</td>
</tr>
</tbody>
</table>
localmoran

Z_i \text{ standard deviate of local moran statistic; for localmoran_perm based on permutation sample means and standard deviations}

Pr() \text{ p-value of local moran statistic using pnorm(); for localmoran_perm using standard deviate based on permutation sample means and standard deviations}

Pr() Sim \text{ For localmoran_perm, rank() and punif() of observed statistic rank for [0, 1] p-values using alternative=}

Pr(folded) Sim \text{ the simulation folded [0, 0.5] range ranked p-value (based on https://github.com/pysal/esda/blob/4a63e0b5df1e754b17b5f1205b8cadcbec5e061/esda/crand.py#L211-L213)}

Skewness \text{ For localmoran_perm, the output of e1071::skewness() for the permutation samples underlying the standard deviates}

Kurtosis \text{ For localmoran_perm, the output of e1071::kurtosis() for the permutation samples underlying the standard deviates}

In addition, an attribute data frame "quadr" with mean and median quadrant columns, and a column splitting on the demeaned variable and lagged demeaned variable at zero.

Note
Conditional permutations added for comparative purposes; permutations are over the whole data vector omitting the observation itself. For p-value adjustment, use p.adjust() or p.adjustSP() on the output vector.

Author(s)
Roger Bivand <Roger.Bivand@nhh.no>

References

See Also
localG

Examples
data(afcon, package="spData")
oid <- order(afcon$id)
resI <- localmoran(afcon$totcon, nb2listw(paper.nb))
printCoefmat(data.frame(resI[oid,], row.names=afcon$name[oid]),
localmoran.exact

Exact local Moran's I tests

Description

localmoran.exact provides exact local Moran's I tests under the null hypothesis, while localmoran.exact.alt provides exact local Moran's I tests under the alternative hypothesis. In this case, the model may be a fitted model derived from a model fitted by `spatialreg::errorsarlm`, with the covariance matrix can be passed through the Omega= argument.

Usage

localmoran.exact(model, select, nb, glist = NULL, style = "W", zero.policy = NULL, alternative = "two.sided", spChk = NULL, resfun = weighted.residuals, save.Vi = FALSE, useTP=FALSE, truncErr=1e-6, zeroTreat=0.1)
localmoran.exact.alt(model, select, nb, glist = NULL, style = "W", zero.policy = NULL, alternative = "two.sided", spChk = NULL, resfun = weighted.residuals, Omega = NULL, save.Vi = FALSE, save.M = FALSE, useTP=FALSE, truncErr=1e-6, zeroTreat=0.1)

## S3 method for class 'localmoranex'
print(x, ...)
## S3 method for class 'localmoranex'
as.data.frame(x, row.names=NULL, optional=FALSE, ...)
Arguments

model
an object of class \texttt{lm} returned by \texttt{lm} (assuming no global spatial autocorrelation), or an object of class \texttt{sarlm} returned by a spatial simultaneous autoregressive model fit (assuming global spatial autocorrelation represented by the model spatial coefficient); weights may be specified in the \texttt{lm} fit, but offsets should not be used

select
an integer vector of the id. numbers of zones to be tested; if missing, all zones

nb
a list of neighbours of class \texttt{nb}

glist
a list of general weights corresponding to neighbours

style
can take values \texttt{W}, \texttt{B}, \texttt{C}, and \texttt{S}

zero.policy
default \texttt{NULL}, use global option value; if \texttt{TRUE} assign zero to the lagged value of zones without neighbours, if \texttt{FALSE} assign \texttt{NA}

alternative
a character string specifying the alternative hypothesis, must be one of greater (default), less or two.sided.

spChk
should the data vector names be checked against the spatial objects for identity integrity, \texttt{TRUE}, or \texttt{FALSE}, default \texttt{NULL} to use \texttt{get.spChkOption()}

resfun
default: weighted.residuals; the function to be used to extract residuals from the \texttt{lm} object, may be \texttt{residuals}, \texttt{weighted.residuals}, \texttt{rstandard}, or \texttt{rstudent}

Omega
A SAR process matrix may be passed in to test an alternative hypothesis, for example \texttt{Omega <- invIrW(listw, rho=0.1)}; \texttt{Omega <- tcrossprod(Omega), chol()} is taken internally

save.Vi
if \texttt{TRUE}, return the star-shaped weights lists for each zone tested

save.M
if \texttt{TRUE}, save a list of left and right M products

useTP
default \texttt{FALSE}, if \texttt{TRUE}, use truncation point in integration rather than upper=\texttt{Inf}, see Tiefelsdorf (2000), eq. 6.7, p.69

truncErr
when \texttt{useTP=TRUE}, pass truncation error to truncation point function

zeroTreat
when \texttt{useTP=TRUE}, pass zero adjustment to truncation point function

x
object to be printed

row.names
ignored argument to as.data.frame.localmoranex; row names assigned from localmoranex object

optional
ignored argument to as.data.frame.localmoranex; row names assigned from localmoranex object

... arguments to be passed through

Value

A list with class \texttt{localmoran.exact} containing "select" lists, each with class \texttt{moran.exact} with the following components:

\texttt{statistic}
the value of the exact standard deviate of global Moran’s I.

\texttt{p.value}
the p-value of the test.

\texttt{estimate}
the value of the observed local Moran’s I.
method a character string giving the method used.
alternative a character string describing the alternative hypothesis.
gamma eigenvalues (two extreme values for null, vector for alternative)
oType usually set to “E”, but set to “N” if the integration leads to an out of domain value for qnorm, when the Normal assumption is substituted. This only occurs when the output p-value would be very close to zero
data.name a character string giving the name(s) of the data.
df degrees of freedom
i zone tested
Vi zone tested

When the alternative is being tested, a list of left and right M products in attribute M.

Author(s)
Markus Reder and Roger Bivand

References

See Also
lm.morantest.exact, localmoran.sad

Examples

eire <- st_read(system.file("shapes/eire.shp", package="spData")[[1]])
row.names(eire) <- as.character(eire$names)
st_crs(eire) <- "+proj=utm +zone=30 +ellps=airy +units=km"
eire.nb <- poly2nb(eire)
e.lm <- lm(OWNCONS ~ ROADACC, data=eire)
localmoran.sad(e.lm, nb=eire.nb)
localmoran.exact(e.lm, nb=eire.nb)
localmoran.exact(e.lm, nb=eire.nb, useTP=TRUE)
run <- FALSE
if (requireNamespace("spatialreg", quietly=TRUE)) run <- TRUE
if (run) {
e.errorsar <- spatialreg::errorsarlm(OWNCONS ~ ROADACC, data=eire,
listw=nb2listw(eire.nb))
lm.target <- lm(e.errorsar$tary ~ e.errorsar$tarX - 1)
localmoran.exact.alt(lm.target, nb=eire.nb)
}
localmoran.sad

Saddlepoint approximation of local Moran’s Ii tests

Description

The function implements Tiefelsdorf’s application of the Saddlepoint approximation to local Moran’s Ii’s reference distribution. If the model object is of class "lm", global independence is assumed; if of class "sarlm", global dependence is assumed to be represented by the spatial parameter of that model. Tests are reported separately for each zone selected, and may be summarised using summary.localmoransad. Values of local Moran’s Ii agree with those from localmoran(), but in that function, the standard deviate - here the Saddlepoint approximation - is based on the randomisation assumption.

Usage

localmoran.sad(model, select, nb, glist=NULL, style="W", zero.policy=NULL, alternative="two.sided", spChk=NULL, resfun=weighted.residuals, save.Vi=FALSE, tol = .Machine$double.eps^0.5, maxiter = 1000, tol.bounds=0.0001, save.M=FALSE, Omega = NULL)

## S3 method for class 'localmoransad'
print(x, ...)

## S3 method for class 'localmoransad'
summary(object, ...)

## S3 method for class 'summary.localmoransad'
print(x, ...)

listw2star(listw, ireg, style, n, D, a, zero.policy=NULL)

Arguments

model an object of class lm returned by lm (assuming no global spatial autocorrelation), or an object of class sarlm returned by a spatial simultaneous autoregressive model fit (assuming global spatial autocorrelation represented by the model spatial coefficient); weights may be specified in the lm fit, but offsets should not be used

select an integer vector of the id. numbers of zones to be tested; if missing, all zones

nb a list of neighbours of class nb

glist a list of general weights corresponding to neighbours

style can take values W, B, C, and S
zero.policy  default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA

alternative  a character string specifying the alternative hypothesis, must be one of greater (default), less or two.sided.

spChk  should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use get.spChkOption()

resfun  default: weighted.residuals; the function to be used to extract residuals from the lm object, may be residuals, weighted.residuals, rstandard, or rstudent

save.Vi  if TRUE, return the star-shaped weights lists for each zone tested

tol  the desired accuracy (convergence tolerance) for uniroot

maxiter  the maximum number of iterations for uniroot

tol.bounds  offset from bounds for uniroot

save.M  if TRUE, save a list of left and right M products in a list for the conditional tests, or a list of the regression model matrix components

Omega  A SAR process matrix may be passed in to test an alternative hypothesis, for example Omega <- invIrW(listw, rho=0.1); Omega <- tcrossprod(Omega), chol() is taken internally

x  object to be printed

object  object to be summarised

...  arguments to be passed through

listw  a listw object created for example by nb2listw

ireg  a zone number

n  internal value depending on listw and style

D  internal value depending on listw and style

a  internal value depending on listw and style

Details

The function implements the analytical eigenvalue calculation together with trace shortcuts given or suggested in Tiefelsdorf (2002), partly following remarks by J. Keith Ord, and uses the Saddlepoint analytical solution from Tiefelsdorf’s SPSS code.

If a histogram of the probability values of the saddlepoint estimate for the assumption of global independence is not approximately flat, the assumption is probably unjustified, and re-estimation with global dependence is recommended.

No n by n matrices are needed at any point for the test assuming no global dependence, the star-shaped weights matrices being handled as listw lists. When the test is made on residuals from a spatial regression, taking a global process into account. n by n matrices are necessary, and memory constraints may be reached for large lattices.
Value

A list with class localmoransad containing "select" lists, each with class moransad with the following components:

- **statistic**: the value of the saddlepoint approximation of the standard deviate of local Moran's I_i.
- **p.value**: the p-value of the test.
- **estimate**: the value of the observed local Moran's I_i.
- **alternative**: a character string describing the alternative hypothesis.
- **method**: a character string giving the method used.
- **data.name**: a character string giving the name(s) of the data.
- **internal1**: Saddlepoint omega, r and u
- **df**: degrees of freedom
- **tau**: maximum and minimum analytical eigenvalues
- **i**: zone tested

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References


See Also

localmoran, lm.morantest, lm.morantest.sad, errorsarlm

Examples

eire <- st_read(system.file("shapes/eire.shp", package="spData")[[1]])
row.names(eire) <- as.character(eire$names)
st_crs(eire) <- "+proj=utm +zone=30 +ellps=airy +units=km"
eire.nb <- poly2nb(eire)
lw <- nb2listw(eire.nb)
e.lm <- lm(OWNCONS ~ ROADACC, data=eire)
e.locmoran <- summary(localmoran.sad(e.lm, nb=eire.nb))
e.locmoran
mean(e.locmoran[,1])
sum(e.locmoran[,1])/Szero(lw)
lm.morantest(e.lm, lw)
# note equality for mean() only when the sum of weights equals # the number of observations (thanks to Juergen Symanzik)
hist(e.locmoran,"Pr. (Sad)"")
e.wlm <- lm(OWNCONS ~ ROADACC, data=eire, weights=RETSALE)
e.locmoranw1 <- summary(localmoran.sad(e.wlm, nb=eire.nb, resfun=weighted.residuals))
e.locmoranw1
LOSH <- Local spatial heteroscedasticity

Description

Local spatial heteroscedasticity is calculated for each location based on the spatial weights object used. The statistic is:

\[ H_i = \frac{\sum_j w_{ij} \cdot |e_j|^a}{\sum_j w_{ij}} \]

with

\[ e_j = x_j - \bar{x}_j \]

and

\[ \bar{x}_j = \frac{\sum_k w_{jk} \cdot x_k}{\sum_k w_{jk}} \]

Its expectation and variance are given in Ord & Getis (2012). The exponent \( a \) allows for investigating different types of mean dispersal.

Usage

LOSH(x, listw, a=2, var_hi=TRUE, zero.policy=NULL, na.action=na.fail, spChk=NULL)
LOSH

Arguments

- `x`: a numeric vector of the same length as the neighbours list in `listw`
- `listw`: a `listw` object created for example by `nb2listw`
- `a`: the exponent applied to the local residuals; the default value of 2 leads to a measure of heterogeneity in the spatial variance
- `var_hi`: default TRUE, the moments and the test statistics are calculated for each location; if FALSE, only the plain LOSH measures, $\bar{x}_i$ and $e_i$ are calculated
- `zero.policy`: default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
- `na.action`: a function (default `na.fail`), can also be `na.omit` or `na.exclude` - in these cases the weights list will be subsetted to remove NAs in the data. It may be necessary to set `zero.policy` to TRUE because this subsetting may create no-neighbour observations. Note that only weights lists created without using the glist argument to `nb2listw` may be subsetted. If `na.pass` is used, zero is substituted for NA values in calculating the spatial lag. (Note that `na.exclude` will only work properly starting from R 1.9.0, `na.omit` and `na.exclude` assign the wrong classes in 1.8.*)
- `spChk`: should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use `get.spChkOption()`

Details

In addition to the LOSH measure, the values returned include local spatially weighted mean values $\bar{x}_i$ and local residuals $e_i$ estimated about these means. These values facilitate the interpretation of LOSH values. Further, if specified through `var_hi`, the statistical moments and the test statistics as proposed by Ord & Getis (2012) are also calculated and returned.

Value

- `Hi`: LOSH statistic
- `E.Hi`: (optional) expectation of LOSH
- `Var.Hi`: (optional) variance of LOSH
- `Z.Hi`: (optional) the approximately Chi-square distributed test statistics
- `x_bar_i`: local spatially weighted mean values
- `ei`: residuals about local spatially weighted mean values

Author(s)

René Westerholt <renee.westerholt@tu-dortmund.de>

References

LOSH.cs

Chi-square based test for local spatial heteroscedasticity

Description

The function implements the chi-square based test statistic for local spatial heteroscedasticity (LOSH) as proposed by Ord & Getis (2012).

Usage

LOSH.cs(x, listw, zero.policy = NULL, na.action = na.fail, p.adjust.method = "none", spChk = NULL)

Arguments

x
a numeric vector of the same length as the neighbours list in listw

listw
a listw object created for example by nb2listw

zero.policy
default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA

na.action
a function (default na.fail), can also be na.omit or na.exclude - in these cases the weights list will be subsetted to remove NAs in the data. It may be necessary to set zero.policy to TRUE because this subsetting may create no-neighbour observations. Note that only weights lists created without using the glist argument to nb2listw may be subsetted. If na.pass is used, zero is substituted for NA values in calculating the spatial lag. (Note that na.exclude will only work properly starting from R 1.9.0, na.omit and na.exclude assign the wrong classes in 1.8.*)

p.adjust.method
a character string specifying the probability value adjustment for multiple tests, default "none"; see p.adjustSP. Note that the number of multiple tests for each region is only taken as the number of neighbours + 1 for each region, rather than the total number of regions.

spChk
should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use get.spChkOption()
Details

The test uses $a = 2$ (see \texttt{LOSH}) because chi-square based inference is not applicable with other exponents. The function makes use of \texttt{LOSH} in its calculations.

Value

- \texttt{Hi}: LOSH statistic
- \texttt{E.Hi}: expectation of LOSH
- \texttt{Var.Hi}: variance of LOSH
- \texttt{Z.Hi}: the approximately chi-square distributed test statistics
- \texttt{x_bar_i}: local spatially weighted mean values
- \texttt{ei}: residuals about local spatially weighted mean values
- \texttt{Pr()}: p-values for $\text{Hi}$ obtained from a non-central Chi-square distribution with $2/\text{Var.Hi}$ degrees of freedom

Author(s)

René Westerholt <rene.westerholt@tu-dortmund.de>

References


See Also

\texttt{LOSH}, \texttt{LOSH.mc}

Examples

```r
data(boston, package="spData")
resLOSH <- LOSH.cs(boston.c$NOX, nb2listw(boston.soi))
hist(resLOSH[, "Hi"])
mean(resLOSH[, "Hi"])
```

---

\texttt{LOSH.mc}  
\textit{Bootstrapping-based test for local spatial heteroscedasticity}

Description

The function draws inferences about local spatial heteroscedasticity (LOSH) by means of the randomisation-based Monte-Carlo bootstrap proposed by Xu et al. (2014).

Usage

```r
LOSH.mc(x, listw, a = 2, nsim = 99, zero.policy = NULL, na.action = na.fail,
         spChk = NULL, adjust.n = TRUE, p.adjust.method = "none")
```
Arguments

- **x**: a numeric vector of the same length as the neighbours list in `listw`
- **listw**: a `listw` object created for example by `nb2listw`
- **a**: the exponent applied to the local residuals; the default value of 2 leads to a measure of heterogeneity in the spatial variance
- **nsim**: the number of randomisations used in the bootstrap
- **zero.policy**: default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
- **na.action**: a function (default `na.fail`), can also be `na.omit` or `na.exclude` - in these cases the weights list will be subsetted to remove NAs in the data. It may be necessary to set `zero.policy` to TRUE because this subsetting may create no-neighbour observations. Note that only weights lists created without using the `glist` argument to `nb2listw` may be subsetted. If `na.pass` is used, zero is substituted for NA values in calculating the spatial lag. (Note that `na.exclude` will only work properly starting from R 1.9.0, `na.omit` and `na.exclude` assign the wrong classes in 1.8.*)
- **spChk**: should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use `get.spChkOption()`
- **adjust.n**: default TRUE, if FALSE the number of observations is not adjusted for no-neighbour observations, if TRUE, the number of observations is adjusted
- **p.adjust.method**: a character string specifying the probability value adjustment for multiple tests, default "none"; see `p.adjustSP`. Note that the number of multiple tests for each region is only taken as the number of neighbours + 1 for each region, rather than the total number of regions.

Details

The test calculates LOSH (see `LOSH`) and estimates pseudo p-values from a conditional bootstrap. Thereby, the i-th value in each location is held fixed, whereas all other values are permuted `nsim` times over all other spatial units.

Value

- **Hi**: LOSH statistic
- **E.Hi**: expectation of LOSH
- **Var.Hi**: variance of LOSH
- **Z.Hi**: the approximately chi-square distributed test statistics
- **x_bar_i**: local spatially weighted mean values
- **ei**: residuals about local spatially weighted mean values
- **Pr()**: p-values for Hi obtained from a conditional bootstrap distribution

Author(s)

René Westerholt <rene.westerholt@tu-dortmund.de>
References


See Also

LOSH, LOSH.mc

Examples

data(columbus, package="spData")
resLOSH_mc <- LOSH.mc(columbus$CRIME, nb2listw(col.gal.nb), 2, 100)
resLOSH_cs <- LOSH.cs(columbus$CRIME, nb2listw(col.gal.nb))
plot(resLOSH_mc[,"Pr()"], resLOSH_cs[,"Pr()"])

mat2listw

Convert a square spatial weights matrix to a weights list object

Description

The function converts a square spatial weights matrix, optionally a sparse matrix to a weights list object, optionally adding region IDs from the row names of the matrix, as a sequence of numbers 1:nrow(x), or as given as an argument. The style can be imposed by rebuilding the weights list object internally.

Usage

mat2listw(x, row.names = NULL, style="M")

Arguments

x A square non-negative matrix with no NAs representing spatial weights; may be a matrix of class "sparseMatrix"
row.names row names to use for region IDs
style default "M", unknown style; if not "M", passed to nb2listw to re-build the object

Value

A listw object with the following members:

style "M", meaning matrix style, underlying style unknown, or assigned style argument in rebuilt object
neighbours the derived neighbours list
weights the weights for the neighbours derived from the matrix
moran
Compute Moran's I

Description
A simple function to compute Moran's I, called by Moran.test and Moran.mc;

\[ I = \frac{n}{\sum w_{ij}} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} (x_i - \bar{x})(x_j - \bar{x}) \]

\[ \sum_{i=1}^{n} (x_i - \bar{x})^2 \]

Usage
moran(x, listw, n, S0, zero.policy=NULL, NAOK=FALSE)
**Arguments**

- **x**: a numeric vector the same length as the neighbours list in listw
- **listw**: a listw object created for example by nb2listw
- **n**: number of zones
- **S0**: global sum of weights
- **zero.policy**: default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
- **NAOK**: if 'TRUE' then any 'NA' or 'NaN' or 'Inf' values in x are passed on to the foreign function. If 'FALSE', the presence of 'NA' or 'NaN' or 'Inf' values is regarded as an error.

**Value**

a list of

- **I**: Moran’s I
- **K**: sample kurtosis of x

**Author(s)**

Roger Bivand <Roger.Bivand@nhh.no>

**References**


**See Also**

- `moran.test`
- `moran.mc`

**Examples**

```r
data(oldcol)
col.W <- nb2listw(COL.nb, style = "W")
crime <- COL.OLD$CRIME
str(moran(crime, col.W, length(COL.nb), Szero(col.W)))
is.na(crime) <- sample(1:length(crime), 10)
str(moran(crime, col.W, length(COL.nb), Szero(col.W), NAOK = TRUE))
```
Permutation test for Moran’s I statistic

Description

A permutation test for Moran’s I statistic calculated by using nsim random permutations of x for the given spatial weighting scheme, to establish the rank of the observed statistic in relation to the nsim simulated values.

Usage

```r
moran.mc(x, listw, nsim, zero.policy=NULL, alternative="greater", na.action=na.fail, spChk=NULL, return_boot=FALSE, adjust.n=TRUE)
```

Arguments

- `x`: a numeric vector the same length as the neighbours list in `listw`
- `listw`: a `listw` object created for example by `nb2listw`
- `nsim`: number of permutations
- `zero.policy`: default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
- `alternative`: a character string specifying the alternative hypothesis, must be one of "greater" (default), or "less".
- `na.action`: a function (default `na.fail`), can also be `na.omit` or `na.exclude` - in these cases the weights list will be subsetted to remove NAs in the data. It may be necessary to set zero.policy to TRUE because this subsetting may create no-neighbour observations. Note that only weights lists created without using the glist argument to `nb2listw` may be subsetted. `na.pass` is not permitted because it is meaningless in a permutation test.
- `spChk`: should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use `get.spChkOption()`
- `return_boot`: return an object of class `boot` from the equivalent permutation bootstrap rather than an object of class `htest`
- `adjust.n`: default TRUE, if FALSE the number of observations is not adjusted for no-neighbour observations, if TRUE, the number of observations is adjusted

Value

A list with class `htest` and `mc.sim` containing the following components:

- `statistic`: the value of the observed Moran’s I.
- `parameter`: the rank of the observed Moran’s I.
- `p.value`: the pseudo p-value of the test.
- `alternative`: a character string describing the alternative hypothesis.
moran.plot

method a character string giving the method used.
data.name a character string giving the name(s) of the data, and the number of simulations.
res nsim simulated values of statistic, final value is observed statistic

Author(s)
Roger Bivand <Roger.Bivand@nhh.no>

References

See Also
moran, moran.test

Examples
data(oldcol)
colw <- nb2listw(COL.nb, style="W")
nsim <- 99
set.seed(1234)
sim1 <- moran.mc(COL.OLD$CRIME, listw=colw, nsim=nsim)
sim1
mean(sim1$res[1:nsim])
var(sim1$res[1:nsim])
summary(sim1$res[1:nsim])
colold.lags <- nblag(COL.nb, 3)
set.seed(1234)
sim2 <- moran.mc(COL.OLD$CRIME, nb2listw(colold.lags[[2]],
    style="W"), nsim=nsim)
sim2
summary(sim2$res[1:nsim])
sim3 <- moran.mc(COL.OLD$CRIME, nb2listw(colold.lags[[3]],
    style="W"), nsim=nsim)
sim3
summary(sim3$res[1:nsim])

moran.plot(x, listw, zero.policy=NULL, spChk=NULL, labels=NULL,
xlab=NULL, ylab=NULL, quiet=NULL, plot=TRUE, return_df=TRUE, ...)
Arguments

- **x**: a numeric vector the same length as the neighbours list in `listw`
- **listw**: a `listw` object created for example by `nb2listw`
- **zero.policy**: default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
- **spChk**: should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use `get.spChkOption()`
- **labels**: character labels for points with high influence measures, if set to FALSE, no labels are plotted for points with large influence
- **xlab**: label for x axis
- **ylab**: label for y axis
- **quiet**: default NULL, use `!verbose` global option value; if TRUE, output of summary of influence object suppressed
- **plot**: default TRUE, if false, plotting is suppressed
- **return_df**: default TRUE, invisibly return a data.frame object; if FALSE invisibly return an influence measures object
- **...**: further graphical parameters as in `par(..)`

Value

The function returns a data.frame object with coordinates and influence measures if `return_df` is TRUE, or an influence object from `influence.measures`.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References


See Also

`localmoran`, `influence.measures`

Examples

data(afcon, package="spData")
mp <- moran.plot(afcon$totcon, nb2listw(paper.nb),
  labels=as.character(afcon$name), pch=19)
moran.plot(as.vector(scale(afcon$totcon)), nb2listw(paper.nb),
  labels=as.character(afcon$name), xlab=-2, ylab=-2, pch=19)
if (require(ggplot2, quietly=TRUE)) {
moran.test <- attr(mp, "xname")

ggplot(mp, aes(x=x, y=wx)) + geom_point(shape=1) +
  geom_smooth(formula=y ~ x, method="lm") +
  geom_hline(yintercept=mean(mp$wx), lty=2) +
  geom_vline(xintercept=mean(mp$x), lty=2) + theme_minimal() +
  geom_point(data=mp[mp$is_inf,], aes(x=x, y=wx), shape=9) +
  geom_text(data=mp[mp$is_inf,], aes(x=x, y=wx, label=labels, vjust=1.5)) +
  xlab(xname) + ylab(paste0("Spatially lagged ", xname))

moran.test  

Moran's I test for spatial autocorrelation

Description

Moran's test for spatial autocorrelation using a spatial weights matrix in weights list form. The assumptions underlying the test are sensitive to the form of the graph of neighbour relationships and other factors, and results may be checked against those of `moran.mc` permutations.

Usage

```r
moran.test(x, listw, randomisation=TRUE, zero.policy=NULL,
           alternative="greater", rank = FALSE, na.action=na.fail,
           spChk=NULL, adjust.n=TRUE, drop.EI2=FALSE)
```

Arguments

- **x**: a numeric vector the same length as the neighbours list in `listw`
- **listw**: a `listw` object created for example by `nb2listw`
- **randomisation**: variance of I calculated under the assumption of randomisation, if FALSE normality
- **zero.policy**: default NULL, use global option value; if TRUE assign zero to the lagged value of zones without neighbours, if FALSE assign NA
- **alternative**: a character string specifying the alternative hypothesis, must be one of greater (default), less or two.sided.
- **rank**: logical value - default FALSE for continuous variables, if TRUE, uses the adaptation of Moran's I for ranks suggested by Cliff and Ord (1981, p. 46)
- **na.action**: a function (default `na.fail`), can also be `na.omit` or `na.exclude` - in these cases the weights list will be subsetted to remove NAs in the data. It may be necessary to set `zero.policy` to TRUE because this subsetting may create no-neighbour observations. Note that only weights lists created without using the `glist` argument to `nb2listw` may be subsetted. If `na.pass` is used, zero is substituted for NA values in calculating the spatial lag
- **spChk**: should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use `get.spChkOption()`
- **adjust.n**: default TRUE, if FALSE the number of observations is not adjusted for no-neighbour observations, if TRUE, the number of observations is adjusted
- **drop.EI2**: default FALSE, if TRUE, emulate CrimeStat <= 4.02
Value

A list with class htest containing the following components:

- **statistic**: the value of the standard deviate of Moran’s I.
- **p.value**: the p-value of the test.
- **estimate**: the value of the observed Moran’s I, its expectation and variance under the method assumption.
- **alternative**: a character string describing the alternative hypothesis.
- **method**: a character string giving the assumption used for calculating the standard deviate.
- **data.name**: a character string giving the name(s) of the data.

Note

Var(I) is taken from Cliff and Ord (1969, p. 28), and Goodchild’s CATMOG 47 (1986), see also Upton & Fingleton (1985) p. 171; it agrees with SpaceStat, see Tutorial workbook Chapter 22; VI is the second crude moment minus the square of the first crude moment. The derivation of the test (Cliff and Ord, 1981, p. 18) assumes that the weights matrix is symmetric. For inherently non-symmetric matrices, such as k-nearest neighbour matrices, listw2U() can be used to make the matrix symmetric.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References


See Also

moran, moran.mc, listw2U

Examples

```r
data(oldcol)
coords.OLD <- cbind(COL.OLD$X, COL.OLD$Y)
moran.test(COL.OLD$CRIME, nb2listw(COL.nb, style="W"))
moran.test(COL.OLD$CRIME, nb2listw(COL.nb, style="B"))
moran.test(COL.OLD$CRIME, nb2listw(COL.nb, style="C"))
moran.test(COL.OLD$CRIME, nb2listw(COL.nb, style="S"),
randomisation=FALSE)
colold.lags <- nblag(COL.nb, 3)
moran.test(COL.OLD$CRIME, nb2listw(colold.lags[[2]],
style="W"))
moran.test(COL.OLD$CRIME, nb2listw(colold.lags[[3]],
```
The minimal spanning tree is a connected graph with \( n \) nodes and \( n-1 \) edges. This is a smaller class of possible partitions of a graph by pruning edges with high dissimilarity. If one edge is removed, the graph is partitioned into two unconnected subgraphs. This function implements the algorithm due to Prim (1987).

### Usage

\[
\text{mstree}(\text{nbw}, \text{ini} = \text{NULL})
\]

### Arguments

- **nbw**: An object of `listw` class returned by `nb2listw` function. See this help for details.
- **ini**: The initial node in the minimal spanning tree.
Details

The minimum spanning tree algorithm.
Input a connected graph.
Begin a empty set of nodes.
Add an arbitrary node in this set.
While are nodes not in the set, find a minimum cost edge connecting a node in the set and a node out of the set and add this node in the set.
The set of edges is a minimum spanning tree.

Value

A matrix with n-1 rows and tree columns. Each row is two nodes and the cost, i.e. the edge and it cost.

Author(s)

Renato M. Assuncao and Elias T. Krainski

References


Examples

```r
### loading data
bh <- st_read(system.file("etc/shapes/bhicv.shp", package="spdep")[[1]], quiet=TRUE)
st_crs(bh) <- "+proj=longlat +ellps=WGS84"
### data padronized
dpad <- data.frame(scale(as.data.frame(bh)[,5:8]))

### neighborhood list
bh.nb <- poly2nb(bh)

### calculating costs
lcosts <- nbcosts(bh.nb, dpad)

### making listw
nb.w <- nb2listw(bh.nb, lcosts, style="B")

### find a minimum spanning tree
system.time(mst.bh <- mstree(nb.w,5))
dim(mst.bh)
head(mst.bh)
tail(mst.bh)

### the mstree plot
par(mar=c(0,0,0,0))
plot(st_geometry(bh), border=gray(.5))
```
plot(mst.bh, coordinates(as(bh, "Spatial")), col=2, cex.lab=.6, cex.circles=0.035, fg="blue", add=TRUE)

**nb.set.operations**

Set operations on neighborhood objects

**Description**

Set operations on neighbors list objects

**Usage**

```r
intersect.nb(nb.obj1,nb.obj2)
union.nb(nb.obj1,nb.obj2)
setdiff.nb(nb.obj1,nb.obj2)
complement.nb(nb.obj)
```

**Arguments**

- `nb.obj` a neighbor list created from any of the neighborhood list functions
- `nb.obj1` a neighbor list created from any of the neighborhood list functions
- `nb.obj2` a neighbor list created from any of the neighborhood list functions

**Details**

These functions perform set operations on each element of a neighborlist. The arguments must be neighbor lists created from the same coordinates, and the region.id attributes must be identical.

**Value**

- `nb.obj` A new neighborlist created from the set operations on the input neighbor list(s)

**Author(s)**

Nicholas Lewin-Koh <nikko@hailmail.net>

**See Also**

- `intersect.nb`, `union.nb`, `setdiff.nb`
Examples

columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[[1]])
coords <- coordinates(as(columbus, "Spatial"))
col.tri.nb <- tri2nb(coords)
oldpar <- par(mfrow=c(1,2))
if (require("dbscan", quietly=TRUE)) {
  col.soi.nb <- graph2nb(soi.graph(col.tri.nb, coords))
  plot(st_geometry(columbus), border="grey")
  plot(col.soi.nb, coords, add=TRUE)
  title(main="Sphere of Influence Graph", cex.main=0.7)
  plot(st_geometry(columbus), border="grey")
  plot(complement.nb(col.soi.nb), coords, add=TRUE)
  title(main="Complement of Sphere of Influence Graph", cex.main=0.7)
}
par(mfrow=c(2,2))
col2 <- droplinks(col.gal.nb, 21)
plot(intersect.nb(col.gal.nb, col2), coords)
plot(union.nb(col.gal.nb, col2), coords)
plot(setdiff.nb(col.gal.nb, col2), coords)
par(oldpar)

---

nb2blocknb  
Block up neighbour list for location-less observations

Description

The function blocks up a neighbour list for known spatial locations to create a new neighbour list for multiple location-less observations known to belong to the spatial locations, using the identification tags of the locations as the key.

Usage

nb2blocknb(nb=NULL, ID, row.names = NULL)

Arguments

nb  
an object of class nb with a list of integer vectors containing neighbour region number ids; if null, an nb object with no neighbours is created of length unique(as.character(ID))

ID  
identification tags of the locations for the location-less observations; sort(unique(as.character(ID))) must be identical to sort(as.character(attr(nb, "region.id"))); same length as row.names if provided.

row.names  
character vector of observation ids to be added to the neighbours list as attribute region.id, default seq(1,nrow(x)); same length as ID if provided.
Details

Assume that there is a list of unique locations, then a neighbour list can be built for that, to create an input neighbour list. This needs to be "unfolded", so that observations belonging to each unique location are observation neighbours, and observations belonging to the location neighbours of the unique location in question are also observation neighbours, finally removing the observation itself (because it should not be its own neighbour). This scenario also arises when say only post codes are available, and some post codes contain multiple observations, where all that is known is that they belong to a specific post code, not where they are located within it (given that the post code locations are known).

Value

The function returns an object of class `nb` with a list of integer vectors containing neighbour observation number ids.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

See Also

`knn2nb, dnearneigh, cell2nb, tri2nb, poly2nb`

Examples

```r
## Not run:
data(boston, package="spData")
summary(as.vector(table(boston.c$TOWN)))
townaggr <- aggregate(boston.utm, list(town=boston.c$TOWN), mean)
block.rel <- graph2nb(relativeneigh(as.matrix(townaggr[,2:3])),
                      as.character(townaggr[,1]), sym=TRUE)
block.rel
print(is.symmetric.nb(block.rel))
plot(block.rel, as.matrix(townaggr[,2:3]))
points(boston.utm, pch=18, col="lightgreen")
block.nb <- nb2blocknb(block.rel, as.character(boston.c$TOWN))
block.nb
print(is.symmetric.nb(block.nb))
plot(block.nb, boston.utm)
points(boston.utm, pch=18, col="lightgreen")
n.comp.nb(block.nb)$nc
moran.test(boston.c$CMEDV, nb2listw(boston.soi))
moran.test(boston.c$CMEDV, nb2listw(block.nb))
block.nb <- nb2blocknb(NULL, as.character(boston.c$TOWN))
block.nb
print(is.symmetric.nb(block.nb))
plot(block.nb, boston.utm)
n.comp.nb(block.nb)$nc
moran.test(boston.c$CMEDV, nb2listw(block.nb, zero.policy=TRUE), zero.policy=TRUE)

## End(Not run)
```
nb2INLA                

Output spatial neighbours for INLA

Description

Output spatial neighbours for INLA

Usage

nb2INLA(file, nb)

Arguments

file  file where adjacency matrix will be stored
nb    an object of class nb

Value

Nothing is returned but a file will be created with the representation of the adjacency matrix as required by INLA for its spatial models.

Author(s)

Virgilio Gomez-Rubio

References

http://www.r-inla.org

Examples

col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[[1]])
td <- tempdir()
x <- nb2INLA(paste(td, "columbus-INLA.adj", sep="/""), col.gal.nb)
readLines(paste(td, "columbus-INLA.adj", sep="/""), n=10)
Use vector files for import and export of weights

Description

Use vector files for import and export of weights, storing spatial entity coordinates in the arcs, and the entity indices in the data frame.

Usage

nb2lines(nb, wts, coords, proj4string=NULL, as_sf=FALSE)
listw2lines(listw, coords, proj4string=NULL, as_sf=FALSE)
df2sn(df, i="i", i_ID="i_ID", j="j", wt="wt")

Arguments

nb a neighbour object of class nb
wts list of general weights corresponding to neighbours
coords matrix of region point coordinates, a Spatial object (points or polygons), or an sf object (points or polygons)
proj4string default NULL; if coords is a Spatial or sf object, this value will be used, otherwise the value will be converted appropriately
as_sf output object in Spatial or sf format, default FALSE, set to TRUE if coords is an sf object and FALSE if a Spatial object
listw a listw object of spatial weights
df a data frame read from a shapefile, derived from the output of nb2lines
i character name of column in df with from entity index
i_ID character name of column in df with from entity region ID
j character name of column in df with to entity index
wt character name of column in df with weights

Details

The neighbour and weights objects may be retrieved by converting the specified columns of the data slot of the SpatialLinesDataFrame object into a spatial.neighbour object, which is then converted into a weights list object.

Value

nb2lines and listw2lines return a SpatialLinesDataFrame object or an sf object; the data frame contains with the from and to indices of the neighbour links and their weights. df2sn converts the data retrieved from reading the data from df back into a spatial.neighbour object.
**nb2listw**

**Spatial weights for neighbours lists**

**Description**

The `nb2listw` function supplements a neighbours list with spatial weights for the chosen coding scheme. The `can.be.simmed` helper function checks whether a spatial weights object is similar to symmetric and can be so transformed to yield real eigenvalues or for Cholesky decomposition.

**Usage**

```r
nb2listw(neighbours, glist=NULL, style="W", zero.policy=NULL)
```

**Arguments**

- `neighbours` an object of class `nb`
- `glist` list of general weights corresponding to neighbours
- `style` style can take values “W”, “B”, “C”, “U”, “minmax” and “S”
- `zero.policy` default NULL, use global option value; if FALSE stop with error for any empty neighbour sets, if TRUE permit the weights list to be formed with zero-length weights vectors
Details

Starting from a binary neighbours list, in which regions are either listed as neighbours or are absent (thus not in the set of neighbours for some definition), the function adds a weights list with values given by the coding scheme style chosen. B is the basic binary coding, W is row standardised (sums over all links to n), C is globally standardised (sums over all links to n), U is equal to C divided by the number of neighbours (sums over all links to unity), while S is the variance-stabilizing coding scheme proposed by Tiefelsdorf et al. 1999, p. 167-168 (sums over all links to n).

If zero policy is set to TRUE, weights vectors of zero length are inserted for regions without neighbour in the neighbours list. These will in turn generate lag values of zero, equivalent to the sum of products of the zero row t(rep(0, length=length(neighbours))) %*% x, for arbitrary numerical vector x of length length(neighbours). The spatially lagged value of x for the zero-neighbour region will then be zero, which may (or may not) be a sensible choice.

If the sum of the glist vector for one or more observations is zero, a warning message is issued. The consequence for later operations will be the same as if no-neighbour observations were present and the zero.policy argument set to true.

The “minmax” style is based on Kelejian and Prucha (2010), and divides the weights by the minimum of the maximum row sums and maximum column sums of the input weights. It is similar to the C and U styles; it is also available in Stata.

Value

A listw object with the following members:

- style one of W, B, C, U, S, minmax as above
- neighbours the input neighbours list
- weights the weights for the neighbours and chosen style, with attributes set to report the type of relationships (binary or general, if general the form of the glist argument), and style as above

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References


See Also

summary.nb, read.gal
Examples

columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[[1]])
coords <- coordinates(as(columbus, "Spatial"))
cards <- card(col.gal.nb)
col.w <- nb2listw(col.gal.nb)
plot(cards, unlist(lapply(col.w$weights, sum)),xlim=c(0,10),
ylim=c(0,10), xlab="number of links", ylab="row sums of weights")
 col.b <- nb2listw(col.gal.nb, style="B")
points(cards, unlist(lapply(col.b$weights, sum)), col="red")
 col.c <- nb2listw(col.gal.nb, style="C")
points(cards, unlist(lapply(col.c$weights, sum)), col="green")
 col.u <- nb2listw(col.gal.nb, style="U")
points(cards, unlist(lapply(col.u$weights, sum)), col="orange")
 col.s <- nb2listw(col.gal.nb, style="S")
points(cards, unlist(lapply(col.s$weights, sum)), col="blue")
legend(x=c(0, 1), y=c(7, 9), legend=c("W", "B", "C", "U", "S"), bty="n",
col=c("black", "red", "green", "orange", "blue"), pch=rep(1,5), cex=0.8,
y.intersp=2.5)
summary(nb2listw(col.gal.nb, style="minmax"))
dlist <- nb2dist(col.gal.nb, coords)
dlist <- lapply(dlist, function(x) 1/x)
col.w.d <- nb2listw(col.gal.nb, glist=dlist)
summary(unlist(col.w$weights))
summary(unlist(col.w.d$weights))
# introducing other conditions into weights - only earlier sales count
# see http://sal.uiuc.edu/pipermail/openspace/2005-October/000610.html
data(baltimore, package="spData")
set.seed(211)
dates <- sample(1:500, nrow(baltimore), replace=TRUE)
nb_15nn <- knn2nb(knearneigh(cbind(baltimore$X, baltimore$Y), k=15))
glist <- vector(mode="list", length=length(nb_15nn))
for (i in seq(along=nb_15nn))
  glist[i] <- ifelse(dates[i] > dates[nb_15nn[[i]]], 1, 0)
listw_15nn_dates <- nb2listw(nb_15nn, glist=glist, style="B")
which(lag(listw_15nn_dates, baltimore$PRICE) == 0)
which(sapply(glist, sum) == 0)
ex <- which(sapply(glist, sum) == 0)[1]
dates[ex]
dates[nb_15nn[[ex]]]

---

### nb2listwdist

**Distance-based spatial weights for neighbours lists**

### Description

The `nb2listwdist` function supplements a neighbours list with spatial weights for the chosen types of distance modelling and coding scheme. While the offered coding schemes parallel those of the `nb2listw` function, three distance-based types of weights are available: inverse distance weighting.
nb2listwdist

(IDW), double-power distance weights, and exponential distance decay. The `can.be.simmed` helper function checks whether a spatial weights object is similar to symmetric and can be so transformed to yield real eigenvalues or for Cholesky decomposition.

Usage

```r	nb2listwdist(neighbours, x, type="idw", style="raw",
              alpha = 1, dmax = NULL, longlat = NULL, zero.policy=NULL)
```

Arguments

- `neighbours`: an object of class `nb`
- `x`: an `sp sf`, or `sfc` object
- `type`: default "idw"; the intended type of distance modelling, can take values "idw", "exp", and "dpd"
- `style`: default "raw"; style can take values "raw", "W", "B", "C", "U", "minmax", and "S"
- `alpha`: default 0; a parameter for controlling the distance modelling, see “Details”
- `dmax`: default NULL, maximum distance threshold that is required for type “dpd” but optional for all other types
- `longlat`: default NULL; TRUE if point coordinates are longitude-latitude decimal degrees, in which case distances are measured in metres; if `x` is a `SpatialPoints` object, the value is taken from the object itself, and overrides this argument if not NULL; distances are measured in map units if FALSE or NULL
- `zero.policy`: default NULL; use global option value; if FALSE stop with error for any empty neighbour sets, if TRUE permit the weights list to be formed with zero-length weights vectors

Details

Starting from a binary neighbours list, in which regions are either listed as neighbours or are absent (thus not in the set of neighbours for some definition), the function adds a distance-based weights list. Three types of distance weight calculations based on pairwise distances $d_{ij}$ are possible, all of which are controlled by parameter “alpha” ($\alpha$ below):

- **idw**: $w_{ij} = d_{ij}^{-\alpha}$,
- **exp**: $w_{ij} = \exp(-\alpha \cdot d_{ij})$,
- **dpd**: $w_{ij} = \left[1 - \left(\frac{d_{ij}}{d_{\text{max}}}ight)^\alpha\right]^\alpha$,

the latter of which leads to $w_{ij} = 0$ for all $d_{ij} > d_{\text{max}}$. Note that IDW weights show extreme behaviour close to 0 and can take on the value infinity. In such cases, the infinite values are replaced by the largest finite weight present in the weights list.

The default coding scheme is “raw”, which outputs the raw distance-based weights without applying any kind of normalisation. In addition, the same coding scheme styles that are also available in the `nb2listw` function can be chosen. B is the basic binary coding, W is row standardised (sums over all links to n), C is globally standardised (sums over all links to n), U is equal to C divided by
the number of neighbours (sums over all links to unity), while S is the variance-stabilising coding
scheme proposed by Tiefelsdorf et al. 1999, p. 167-168 (sums over all links to n). The “minmax
style is based on Kelejian and Prucha (2010), and divides the weights by the minimum of the
maximum row sums and maximum column sums of the input weights. It is similar to the C and U
styles; it is also available in Stata.

If zero.policy is set to TRUE, weights vectors of zero length are inserted for regions without neigh-
bour in the neighbours list. These will in turn generate lag values of zero, equivalent to the sum of
products of the zero row \( t(\text{rep}(0, \text{length}=\text{length}(\text{neighbours}))) \times x \), for arbitraty numerical
vector \( x \) of length \( \text{length}(\text{neighbours}) \). The spatially lagged value of \( x \) for the zero-neighbour
region will then be zero, which may (or may not) be a sensible choice.

Value

A listw object with the following members:

- **style**
  - one of W, B, C, U, S, minmax as above
- **type**
  - one of idw, exp, dpd as above
- **neighbours**
  - the input neighbours list
- **weights**
  - the weights for the neighbours and chosen style, with attributes set to report the
type of relationships (binary or general, if general the form of the glist argu-
ment), and style as above

Author(s)

Rene Westerholt <rene.westerholt@tu-dortmund.de>

References

Tiefelsdorf, M., Griffith, D. A., Boots, B. 1999 A variance-stabilizing coding scheme for spatial
2010. Specification and estimation of spatial autoregressive models with autoregressive and het-

See Also

nb2listw, summary.nb

Examples

```r
# World examples
data(world, package="spData")
# neighbours on distance interval [0, 1000] kilometres
suppressWarnings(st_crs(world) <- "+proj=longlat") # for older PROJ
pts <- st_centroid(st_transform(world, 3857))
nb_world <- dnearneigh(pts, 0, 1000000)
# Moran's I (life expectancy) with IDW with alpha = 2, no coding scheme
world_weights <- nb2listwdist(nb_world, as(pts, "Spatial"), type = "idw",
                              alpha = 2, zero.policy = TRUE)
moran.test(world$d1ifeExp, world_weights, zero.policy = TRUE, na.action = na.pass)
## Not run:
```
# Moran's I (life expectancy) with IDW with alpha = 2, no coding scheme
world_weights <- nb2listwdist(nb_world, pts, type = "idw",
  alpha = 2, zero.policy = TRUE)
moran.test(world$LifeExp, world_weights, zero.policy = TRUE, na.action = na.pass)

# Moran's I (life expectancy), DPD, alpha = 2, dmax = 1000 km, no coding scheme
world_weights <- nb2listwdist(nb_world, pts, type = "dpd",
  dmax = 1000000, alpha = 2, zero.policy = TRUE)
moran.test(world$LifeExp, world_weights, zero.policy = TRUE, na.action = na.pass)

# Boston examples
data(boston, package="spData")
boston_coords <- data.frame(x = boston.utm[,1], y = boston.utm[,2])
boston.geoms <- st_as_sf(boston_coords, coords = c("x", "y"), remove = FALSE)
b_boston <- dnearneigh(boston.geoms, 0, 3)

# Moran's I (crime) with exp weights with alpha = 2, no coding scheme
boston_weights <- nb2listwdist(nb_boston, boston.geoms, type = "exp", alpha = 2,
  style="raw", zero.policy = TRUE)
moran.test(boston.c$CRIM, boston_weights, zero.policy = TRUE, na.action = na.pass)

# Moran's I (crime) with idw weights with alpha = 2, coding scheme = W
boston_weights <- nb2listwdist(nb_boston, boston.geoms, type = "idw", alpha = 2,
  style="W", zero.policy = TRUE)
moran.test(boston.c$CRIM, boston_weights, zero.policy = TRUE, na.action = na.pass)

## End(Not run)

---

### nb2mat

**Spatial weights matrices for neighbours lists**

**Description**

The function generates a weights matrix for a neighbours list with spatial weights for the chosen coding scheme.

**Usage**

```r
nb2mat(neighbours, glist=NULL, style="W", zero.policy=NULL)
listw2mat(listw)
```

**Arguments**

- `neighbours` an object of class `nb`
- `glist` list of general weights corresponding to neighbours
- `style` style can take values `W`, `B`, `C`, and `S`
- `zero.policy` default `NULL`, use global option value; if `FALSE` stop with error for any empty neighbour sets, if `TRUE` permit the weights list to be formed with zero-length weights vectors
- `listw` a `listw` object from for example `nb2listw`
Details

Starting from a binary neighbours list, in which regions are either listed as neighbours or are absent (thus not in the set of neighbours for some definition), the function creates an n by n weights matrix with values given by the coding scheme style chosen. B is the basic binary coding, W is row standardised, C is globally standardised, while S is the variance-stabilizing coding scheme proposed by Tiefelsdorf et al. 1999, p. 167-168.

The function leaves matrix rows as zero for any regions with zero neighbours fore zero.policy TRUE. These will in turn generate lag values of zero, equivalent to the sum of products of the zero row t(rep(0, length=length(neighbours))) %*% x, for arbitraty numerical vector x of length length(neighbours). The spatially lagged value of x for the zero-neighbour region will then be zero, which may (or may not) be a sensible choice.

Value

An n by n matrix, where n=length(neighbours)

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References


See Also

nb2listw

Examples

columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
col005 <- dnearneigh(st_coordinates(st_centroid(st_geometry(columbus),
  of_largest_polygon=TRUE)), 0, 0.5, as.character(columbus$NEIGNO))
summary(col005)
col005.w.mat <- nb2mat(col005, style="B", zero.policy=TRUE)
table(round(rowSums(col005.w.mat)))

---

nb2WB

Output spatial weights for WinBUGS

Description

Output spatial weights for WinBUGS

Usage

nb2WB(nb)
listw2WB(listw)
Arguments

nb  an object of class nb
listw  a listw object from for example nb2listw

Value

A list suitable for conveting using dput for WinBUGS

Author(s)

Virgilio Gomez-Rubio

References


See Also

dput

Examples

col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[[1]])
x <- nb2WB(col.gal.nb)
dput(x, control=NULL)
x <- listw2WB(nb2listw(col.gal.nb))
dput(x, control=NULL)

nbcosts

Compute cost of edges

Description

The cost of each edge is the distance between its nodes. This function computes this distance using a data.frame with observations vector in each node.

Usage

nbcost(data, id, id.neigh, method = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski", "mahalanobis"), p = 2, cov, inverted = FALSE)
nbcosts(nb, data, method = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski", "mahalanobis"), p = 2, cov, inverted = FALSE)
Arguments

nb  An object of nb class. See poly2nb for details.
data A matrix with observations in the nodes.
id  Node index to compute the cost
id.neigh Idex of neighbours nodes of node id
method Character or function to declare distance method. If method is character, method must be "mahalanobis" or "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowisk". If method is one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowisk", see dist for details, because this function as used to compute the distance. If method= "mahalanobis", the mahalanobis distance is computed between neighbour areas. If method is a function, this function is used to compute the distance.
p  The power of the Minkowski distance.
cov The covariance matrix used to compute the mahalanobis distance.
inverted logical. If 'TRUE', 'cov' is supposed to contain the inverse of the covariance matrix.

Value

A object of nbdist class. See nbdists for details.

Note

The neighbours must be a connected graph.

Author(s)

Elias T. Krainski and Renato M. Assuncao

See Also

See Also as nbdists, nb2listw

nbdists

Spatial link distance measures

Description

Given a list of spatial neighbour links (a neighbours list of object type nb), the function returns the Euclidean distances along the links in a list of the same form as the neighbours list. If longlat = TRUE, Great Circle distances are used.

Usage

nbdists(nb, coords, longlat = NULL)
Arguments

nb: an object of class nb
coords: matrix of point coordinates, an object inheriting from SpatialPoints or an "sf" or "sfc" object; if the "sf" or "sfc" object geometries are in geographical coordinates (sf::st_is_longlat(x) == TRUE and sf::sf_use_s2() == TRUE), s2 will be used to find distances https://github.com/r-spatial/s2/issues/125
longlat: TRUE if point coordinates are longitude-latitude decimal degrees, in which case distances are measured in kilometers; if coords is a SpatialPoints object, the value is taken from the object itself

Value

A list with class nbdist

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

See Also

summary.nb, nb2listw

Examples

columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[[1]])
coords <- coordinates(as(columbus, "Spatial"))
dlist <- nbdists(col.gal.nb, coords)
dlist <- lapply(dlist, function(x) 1/x)
stem(unlist(dlist))

nblag

Higher order neighbours lists

Description

The function creates higher order neighbour lists, where higher order neighbours are only lags links from each other on the graph described by the input neighbours list. It will refuse to lag neighbours lists with the attribute self.included set to TRUE. nblag_cumul cumulates neighbour lists to a single neighbour list ("nb" object).

Usage

nblag(neighbours, maxlag)
nblag_cumul(nblags)
Arguments

neighbours  input neighbours list of class nb
maxlag      the maximum lag to be constructed
nblags      a list of neighbour lists as output by nblag

Value

returns a list of lagged neighbours lists each with class nb

Author(s)

Roger Bivand <Roger.Bivand@nhh.no> and Giovanni Millo

See Also

summary.nb

Examples

columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[[1]])
coords <- coordinates(as(columbus, "Spatial"))
summary(col.gal.nb, coords)
col.lags <- nblag(col.gal.nb, 2)
print(col.lags)
summary(col.lags[[2]], coords)
plot(st_geometry(columbus), border="grey")
plot(col.gal.nb, coords, add=TRUE)
title(main="GAL order 1 (black) and 2 (red) links")
plot(col.lags[[2]], coords, add=TRUE, col="red", lty=2)
cuml <- nblag_cumul(col.lags)
cuml
run <- FALSE
if (require(igraph, quietly=TRUE) && require(spatialreg, quietly=TRUE)) run <- TRUE
if (run) {
  W <- as(nb2listw(col.gal.nb), "CsparseMatrix")
  G <- graph.adjacency(W, mode="directed", weight="W")
  D <- diameter(G)
  nbs <- nblag(col.gal.nb, maxlag=D)
  n <- length(col.gal.nb)
  lmat <- lapply(nbs, nb2mat, style="B", zero.policy=TRUE)
  mat <- matrix(0, n, n)
  for (i in seq(along=lmat)) mat = mat + i*lmat[[i]]
  G2 <- shortest.paths(G)
  print(all.equal(G2, mat, check.attributes=FALSE))
}
Description

The COL.OLD data frame has 49 rows and 22 columns. The observations are ordered and numbered as in the original analyses of the data set in the SpaceStat documentation and in Anselin, L. 1988 Spatial econometrics: methods and models, Dordrecht: Kluwer. Unit of analysis: 49 neighbourhoods in Columbus, OH, 1980 data. In addition the data set includes COL.nb, the neighbours list as used in Anselin (1988).

Usage

data(oldcol)

Format

This data frame contains the following columns:

- **AREA_PL** computed by ArcView (agrees with areas of polygons in the “columbus” data set
- **PERIMETER** computed by ArcView
- **COLUMBUS** internal polygon ID (ignore)
- **COLUMBUS.I** another internal polygon ID (ignore)
- **POLYID** yet another polygon ID
- **NEIG** neighborhood id value (1-49); conforms to id value used in Spatial Econometrics book.
- **HOVAL** housing value (in \$1,000)
- **INC** household income (in \$1,000)
- **CRIME** residential burglaries and vehicle thefts per thousand households in the neighborhood
- **OPEN** open space in neighborhood
- **PLUMB** percentage housing units without plumbin
- **DISCBD** distance to CBD
- **X** x coordinate (in arbitrary digitizing units, not polygon coordinates)
- **Y** y coordinate (in arbitrary digitizing units, not polygon coordinates)
- **AREA_SS** neighborhood area (computed by SpaceStat)
- **NSA** north-south dummy (North=1)
- **NSB** north-south dummy (North=1)
- **EW** east-west dummy (East=1)
- **CP** core-periphery dummy (Core=1)
- **THOUS** constant=1,000
- **NEIGNO** NEIG+1,000, alternative neighborhood id value
- **PERIM** polygon perimeter (computed by SpaceStat)
The row names of COL.OLD and the region.id attribute of COL.nb are set to columbus$NEIGNO.

Note

All source data files prepared by Luc Anselin, Spatial Analysis Laboratory, Department of Agricultural and Consumer Economics, University of Illinois, Urbana-Champaign, https://spatial.uchicago.edu/sample-data.

Source


Description

Make an adjustment to local association measures' p-values based on the number of neighbours (+1) of each region, rather than the total number of regions.

Usage

p.adjustSP(p, nb, method = "none")

Arguments

- `p` vector of p-values
- `nb` a list of neighbours of class nb
- `method` correction method as defined in `p.adjust`: "The adjustment methods include the Bonferroni correction ("bonferroni") in which the p-values are multiplied by the number of comparisons. Four less conservative corrections are also included by Holm (1979) ("holm"), Hochberg (1988) ("hochberg"), Hommel (1988) ("hommel") and Benjamini & Hochberg (1995) ("fdr"), respectively. A pass-through option ("none") is also included."

Value

A vector of corrected p-values using only the number of neighbours + 1.

Author(s)

Danlin Yu and Roger Bivand <Roger.Bivand@nhh.no>

See Also

`p.adjust`, `localG`, `localmoran`
Examples

```r
data(afcon, package = "spData")
oid <- order(afcon$id)
resG <- as.vector(localG(afcon$totcon, nb2listw(include.self(paper.nb))))
non <- format.pval(pnorm(2*(abs(resG)), lower.tail = FALSE), 2)
bon <- format.pval(p.adjustSP(pnorm(2*(abs(resG)), lower.tail = FALSE),
paper.nb, "bonferroni"), 2)
tot <- format.pval(p.adjust(pnorm(2*(abs(resG)), lower.tail = FALSE),
"bonferroni", n = length(resG)), 2)
data.frame(resG, non, bon, tot, row.names = afcon$name)[oid,]
```

**plot.mst**

*Plot the Minimum Spanning Tree*

Description

This function plots a MST, the nodes are circles and the edges are segments.

Usage

```r
## S3 method for class 'mst'
plot(x, coords, label.areas = NULL,
cex.circles = 1, cex.labels = 1, add = FALSE, ...)
```

Arguments

- **x**: Object of `mst` class.
- **coords**: A two column matrix with the coordinates of nodes.
- **label.areas**: A vector with the labels of nodes.
- **cex.circles**: The length of circles to plot.
- **cex.labels**: The length of nodes labels ploted.
- **add**: default FALSE, create new plot
- **...**: Further arguments passed to plotting functions.

Author(s)

Elias T. Krainski and Renato M. Assuncao

See Also

See Also as `skater` and `mstree`

Examples

```r
### see example in mstree function documentation
```
plot.nb  

Plot a neighbours list

Description

A function to plot a neighbours list given point coordinates to represent the region in two dimensions; plot.listw is a wrapper that passes its neighbours component to plot.nb.

Usage

```r
## S3 method for class 'nb'
plot(x, coords, col="black", points=TRUE, add=FALSE, arrows=FALSE, length=0.1, xlim=NULL, ylim=NULL, ...)
## S3 method for class 'listw'
plot(x, coords, col="black", points=TRUE, add=FALSE, arrows=FALSE, length=0.1, xlim=NULL, ylim=NULL, ...)
```

Arguments

- `x`: an object of class `nb` or (for `plot.listw`) class `listw`
- `coords`: matrix of region point coordinates, a `Spatial` object (points or polygons), or an `sfc` object (points or polygons)
- `col`: plotting colour
- `points`: (logical) add points to plot
- `add`: (logical) add to existing plot
- `arrows`: (logical) draw arrowheads for asymmetric neighbours
- `length`: length in plot inches of arrow heads drawn for asymmetric neighbours lists
- `xlim, ylim`: plot window bounds
- `...`: further graphical parameters as in `par(..)`

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

See Also

`summary.nb`

Examples

```r
columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[[1]])
plot(col.gal.nb, st_geometry(columbus))
title(main="GAL order 1 links with first nearest neighbours in red", cex.main=0.6)
plot(col.gal.nb, as(columbus, "Spatial"))
```
coords <- st_centroid(st_geometry(columbus), of_largest_polygon=TRUE)
col.knn <- knearneigh(coords, k=1)
plot(knn2nb(col.knn), coords, add=TRUE, col="red", length=0.08)

plot.skater

Plot the object of skater class

Description

This function displays the results of the skater function. The subgraphs are plotted with different colours.

Usage

## S3 method for class 'skater'
plot(x, coords, label.areas = NULL,
    groups.colors, cex.circles = 1, cex.labels = 1, ...)

Arguments

x An object of skater class.
coords A matrix of two columns with coordinates of nodes.
label.areas A vector of labels of nodes.
groups.colors A vector with colors of groups or sub-graphs.
cex.circles The length of circles with represent the nodes.
cex.labels The length of labels of nodes.
... Further arguments passed to plotting functions.

Author(s)

Elias T. Krainski and Renato M. Assuncao

See Also

See Also as skater and mstree

Examples

### see example in the skater function documentation
Description

The function builds a neighbours list based on regions with contiguous boundaries, that is sharing one or more boundary point. The current function is in part interpreted and may run slowly for many regions or detailed boundaries, but from 0.2-16 should not fail because of lack of memory when single polygons are built of very many border coordinates.

Usage

poly2nb(pl, row.names = NULL, snap=sqrt(.Machine$double.eps), queen=TRUE, useC=TRUE, foundInBox=NULL)

Arguments

pl list of polygons of class extending SpatialPolygons, or an sf or sfc object containing non-empty (multi-)polygon objects
row.names character vector of region ids to be added to the neighbours list as attribute region.id, default seq(1,nrow(x)); if pl has row.names, they are used instead of the default sequence.
snap boundary points less than snap distance apart are considered to indicate contiguity; used both to find candidate and actual neighbours for planar geometries, but only actual neighbours for spherical geometries, as spherical spatial indexing itself injects some fuzzyness.
queen if TRUE, a single shared boundary point meets the contiguity condition, if FALSE, more than one shared point is required; note that more than one shared boundary point does not necessarily mean a shared boundary line
useC default TRUE, doing the work loop in C, may be set to false to revert to R code calling two C functions in an n*k work loop, where k is the average number of candidate neighbours
foundInBox default NULL using R code or st_intersects() to generate candidate neighbours (using snap= if the geometries are not spherical); if not NULL (for legacy purposes) a list of length (n-1) with integer vectors of candidate neighbours (j > i) (as created by the poly_findInBoxGEOS function in rgeos for clean polygons)

Value

A neighbours list with class nb. See card for details of “nb” objects.
Note

From 0.5-8, the function includes faster bounding box indexing and other improvements contributed by Micah Altman. If a cluster is provided using set.clusterOption, it will be used for finding candidate bounding box overlaps for exact testing for contiguity.

Until 1.1-7, sf polygons included both start and end points, so could erroneously report queen neighbourhood where only rook was present, see https://github.com/r-spatial/spdep/issues/50.

From 1.1-9 with sf 1.0-0, s2 is used in bounding box indexing internally when pl is in geographical coordinates. Because the topology engine of s2 differs from the use of GEOS for planar coordinates by sf, some output differences may be expected. Since treating spherical geometries as planar is also questionable, it is not clear whether spherical contiguous polygon neighbours should simply follow neighbours found by treating the geometries as planar https://github.com/r-spatial/s2/issues/125#issuecomment-864403372. However, current advice is not necessarily to use s2 for finding contiguity neighbours, or at least to check output.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no> with contributions from Micah Altman

See Also

summary.nb, card

Examples

columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[[1]])
coords <- st_coordinates(st_centroid(st_geometry(columbus)))
xx <- poly2nb(as(columbus, "Spatial"))
dxx <- diffnb(xx, col.gal.nb)
plot(st_geometry(columbus), border="grey")
plot(col.gal.nb, coords, add=TRUE)
plot(dxx, coords, add=TRUE, col="red")
title(main=paste("Differences (red) in Columbus GAL weights (black)",
"and polygon generated queen weights", sep="\n"), cex.main=0.6)
# poly2nb with sf sfc_MULTIPOLYGON objects
sf_xx <- poly2nb(columbus)
diffnb(sf_xx, xx)
sfc_xx <- poly2nb(st_geometry(columbus))
diffnb(sfc_xx, xx)
xxx <- poly2nb(as(columbus, "Spatial"), queen=FALSE)
dxxx <- diffnb(xxx, col.gal.nb)
plot(st_geometry(columbus), border = "grey")
plot(col.gal.nb, coords, add = TRUE)
plot(dxxx, coords, add = TRUE, col = "red")
title(main=paste("Differences (red) in Columbus GAL weights (black)",
"and polygon generated rook weights", sep="\n"), cex.main=0.6)
cards <- card(xx)
maxconts <- which(cards == max(cards))
if(length(maxconts) > 1) maxconts <- maxconts[[1]]
fg <- rep("grey", length(cards))
fg[maxconts] <- "red"
fg[xx[[maxconts]]] <- "green"
plot(st_geometry(columbus), col=fg)
title(main="Region with largest number of contiguities", cex.main=0.6)
nc.sids <- st_read(system.file("shapes/sids.shp", package="spData")[[1]], quiet=TRUE)
system.time(xxnb <- poly2nb(nc.sids))
system.time(xxnb <- poly2nb(as(nc.sids, "Spatial")))
plot(st_geometry(nc.sids))
plot(xxnb, coordinates(as(nc.sids, "Spatial")), add=TRUE, col="blue")
sq <- st_polygon(list(rbind(c(0,0), c(1,0), c(1,1), c(0,1), c(0,0))))
sq2 <- sq + c(0,1)
sq3 <- sq + c(1,0)
sq4 <- sq + c(1,1)
gm <- st_sfc(list(sq, sq2, sq3, sq4))
df <- st_as_sf(gm, id=1:4)
plot(st_geometry(df))
text(st_coordinates(st_centroid(gm)), as.character(df$id))
unclass(poly2nb(df, queen = FALSE))

---

**probmap**

*Probability mapping for rates*

**Description**

The function returns a data frame of rates for counts in populations at risk with crude rates, expected counts of cases, relative risks, and Poisson probabilities.

**Usage**

```r
probmap(n, x, row.names=NULL, alternative="less")
```

**Arguments**

- `n` a numeric vector of counts of cases
- `x` a numeric vector of populations at risk
- `row.names` row names passed through to output data frame
- `alternative` default “less”, may be set to “greater”

**Details**

The function returns a data frame, from which rates may be mapped after class intervals have been chosen. The class intervals used in the examples are mostly taken from the referenced source.
**Value**

- **raw**: raw (crude) rates
- **expCount**: expected counts of cases assuming global rate
- **relRisk**: relative risks: ratio of observed and expected counts of cases multiplied by 100
- **pmap**: Poisson probability map values: probability of getting a more “extreme” count than actually observed - one-tailed, default alternative observed “less” than expected

**Author(s)**

Roger Bivand <Roger.Bivand@nhh.no>

**References**


**See Also**

EBest, EBlocal, ppois

**Examples**

```r
auckland <- st_read(system.file("shapes/auckland.shp", package="spData")[[1]], quiet=TRUE)
res <- probmap(auckland$M77_85, 9*auckland$Und5_81)
rt <- sum(auckland$M77_85)/sum(9*auckland$Und5_81)
ppois_pmap <- numeric(length(auckland$Und5_81))
  for (i in seq(along=ppois_pmap)) {
    ppois_pmap[i] <- poisson.test(auckland$M77_85[i], r=rt, T=(9*auckland$Und5_81[i]), alternative="less")$p.value
  }
all.equal(ppois_pmap, res$pmap)
res$id <- 1:nrow(res)
auckland$id <- res$id <- 1:nrow(res)
auckland_res <- merge(auckland, res, by="id")
plot(auckland_res[, "raw"], main="Crude (raw) estimates")
plot(auckland_res[, "relRisk"], main="Standardised mortality ratios")
plot(auckland_res[, "pmap"], main="Poisson probabilities", breaks=c(0, 0.05, 0.1, 0.5, 0.9, 0.95, 1))
```

---

**prunecost**

*Compute cost of prune each edge*

**Description**

If any edge are dropped, the MST are pruned. This generate a two subgraphs. So, it makes a tree graphs and tree dissimilarity values are computed, one for each graph. The dissimilarity is the sum over squared differences between the observations in the nodes and mean vector of observations in the graph. The dissimilarity of original graph and the sum of dissimilarity of subgraphs are returned.
Usage

prunecost(edges, data, method = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski", "mahalanobis"), p = 2, cov, inverted = FALSE)

Arguments

edges A matrix with 2 columns with each row is one edge

data A data.frame with observations in the nodes.

method Character or function to declare distance method. If method is character, method must be "mahalanobis" or "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowisk". If method is one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowisk", see dist for details, because this function as used to compute the distance. If method="mahalanobis", the mahalanobis distance is computed between neighbour areas. If method is a function, this function is used to compute the distance.

p The power of the Minkowski distance.

cov The covariance matrix used to compute the mahalanobis distance.

inverted logical. If 'TRUE', 'cov' is supposed to contain the inverse of the covariance matrix.

Value

A vector with the differences between the dissimilarity of all nodes and the dissimilarity sum of all subgraphs obtained by pruning one edge each time.

Author(s)

Elias T. Krainski and Renato M. Assuncao

See Also

See Also as prunemst

Examples

d <- data.frame(a=-2:2, b=runif(5))
e <- matrix(c(1,2, 2,3, 3,4, 4,5), ncol=2, byrow=TRUE)

sum(sweep(d, 2, colMeans(d))^2)

prunecost(e, d)
prunemst

Prune a Minimum Spanning Tree

Description

This function deletes a first edge and makes two subsets of edges. Each subset is a Minimum Spanning Tree.

Usage

prunemst(edges, only.nodes = TRUE)

Arguments

edges A matrix with two columns with each row is one edge
only.nodes If only.nodes=FALSE, return a edges and nodes of each MST resulted. If only.nodes=TRUE, return a two sets of nodes. Default is TRUE

Value

A list of length two. If only.nodes=TRUE each element is a vector of nodes. If only.nodes=FALSE each element is a list with nodes and edges.

Author(s)

Elias T. Krainski and Renato M. Assuncao

See Also

See Also as mstree

Examples

e <- matrix(c(2,3, 1,2, 3,4, 4,5), ncol=2, byrow=TRUE)
e
prunemst(e)
prunemst(e, only.nodes=FALSE)
**read.gal**

Read a GAL lattice file into a neighbours list

**Description**

The function `read.gal()` reads a GAL lattice file into a neighbours list for spatial analysis. It will read old and new style (GeoDa) GAL files. The function `read.geoda` is a helper file for reading comma separated value data files, calling `read.csv()`.

**Usage**

```r
read.gal(file, region.id=NULL, override.id=FALSE)
read.geoda(file, row.names=NULL, skip=0)
```

**Arguments**

- `file`: name of file with GAL lattice data
- `region.id`: region IDs in specified order to coerce neighbours list order and numbering to that of the `region.id`
- `override.id`: override any given (or NULL) region.id, collecting region.id numbering and order from the GAL file.
- `row.names`: as in `row.names` in `read.csv()`, typically a character string naming the column of the file to be used
- `skip`: skip number of lines, as in `read.csv()`

**Details**

Luc Anselin (2003): Spatial Analysis Laboratory, Department of Agricultural and Consumer Economics, University of Illinois, Urbana-Champaign, now dead link: http://www.csiss.org/gispops/worksshops/2011/PSU/readings/W15_Anselin2007.pdf;

**Value**

The function `read.gal()` returns an object of class `nb` with a list of integer vectors containing neighbour region number ids. The function `read.geoda` returns a data frame, and issues a warning if the returned object has only one column.

**Note**

Example data originally downloaded from now dead link: http://sal.agecon.uic.edu/weights/zips/us48.zip

**Author(s)**

Roger Bivand <Roger.Bivand@nhh.no>
read.gwt2nb

Read and write spatial neighbour files

Description

The "gwt" functions read and write GeoDa GWT files (the example file baltk4.GWT was downloaded from the site given in the reference), and the "dat" functions read and write Matlab sparse matrix files as used by James LeSage’s Spatial Econometrics Toolbox (the example file wmat.dat was downloaded from the site given in the reference). The body of the files after any headers should have three columns separated by white space, and the third column must be numeric in the locale of the reading platform (correct decimal separator).

Usage

read.gwt2nb(file, region.id=NULL)
write.sn2gwt(sn, file, shpfile=NULL, ind=NULL, useInd=FALSE, legacy=FALSE)
read.dat2listw(file)
write.sn2dat(sn, file)

Arguments

file name of file with weights data
region.id region IDs
sn a spatial.neighbour object
shpfile character string: if not given Shapefile name taken from GWT file for this dataset

See Also

summary.nb

Examples

us48.fipsno <- read.geoda(system.file("etc/weights/us48.txt", package="spdep")[1])
us48.q <- read.gal(system.file("etc/weights/us48_q.GAL", package="spdep")[1], us48.fipsno$Fipsno)
us48.r <- read.gal(system.file("etc/weights/us48_rk.GAL", package="spdep")[1], us48.fipsno$Fipsno)
data(state)
if (as.numeric(paste(version$major, version$minor, sep="")) < 19) {
m50.48 <- match(us48.fipsno$"State.name", state.name)
} else {
m50.48 <- match(us48.fipsno$"State_name", state.name)
}
plot(us48.q, as.matrix(as.data.frame(state.center))[m50.48,])
plot(diffnb(us48.r, us48.q), as.matrix(as.data.frame(state.center))[m50.48,], add=TRUE, col="red")
title(main="Differences between rook and queen criteria imported neighbours lists")
ind character string: region id indicator field name

useInd default FALSE, if TRUE, write region.id attribute ID key tags to output file (use in OpenGeoDa will depend on the shapefile having the field named in the ind argument matching the exported tags)

legacy default FALSE; if TRUE, header has single field with number of observations only

Details

Attempts to honour the region.id argument given when reading GWT files. If the region IDs given in region.id= do not match the origins or destinations in the GWT file, an error will be thrown reporting Error: !anyNA(reg*dij) is not TRUE where '*' may be 'o' for origins or 'd' for destinations.

Value

read.gwt2nb returns a neighbour "nb" object with the generalised weights stored as a list element called "dlist" of the "GeoDa" attribute.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References


See Also

read.gal

Examples

data(baltimore, package="spData")
STATION <- baltimore$STATION
gwt1 <- read.gwt2nb(system.file("weights/baltk4.GWT", package="spData")[1],
                        STATION)
cat(paste("Neighbours list symmetry;", is.symmetric.nb(gwt1, FALSE, TRUE),
                        "\n"))
listw1 <- nb2listw(gwt1, style="B", glist=attr(gwt1, "GeoDa")$dist)
tmpGWT <- tempfile()
write.sn2gwt(listw2sn(listw1), tmpGWT)
gwt2 <- read.gwt2nb(tmpGWT, STATION)
cat(paste("Neighbours list symmetry;", is.symmetric.nb(gwt2, FALSE, TRUE),
                        "\n"))
diffnb(gwt1, gwt2)
data(oldcol)
tmpMAT <- tempfile()
COL.W <- nb2listw(COL.nb)
write.sn2dat(listw2sn(COL.W), tmpMAT)
listwmat1 <- read.dat2listw(tmpMAT)
diffnb(listwmat1$neighbours, COL.nb, verbose=TRUE)
listwmat2 <- read.dat2listw(system.file("etc/weights/wmat.dat", package="spdep")[[1]])
diffnb(listwmat1$neighbours, listwmat2$neighbours, verbose=TRUE)

---

**Rotation**

*Rotate a set of point by a certain angle*

**Description**

Rotate a set of XY coordinates by an angle (in radians)

**Usage**

```r
Rotation(xy, angle)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xy</td>
<td>A 2-columns matrix or data frame containing a set of X and Y coordinates.</td>
</tr>
<tr>
<td>angle</td>
<td>Numeric. A scalar giving the angle at which the points should be rotated. The angle is in radians.</td>
</tr>
</tbody>
</table>

**Value**

A 2-columns matrix of the same size as *xy* giving the rotated coordinates.

**Author(s)**

F. Guillaume Blanchet

**Examples**

```r
set.seed(1)
### Create a set of coordinates
coords <- cbind(runif(20), runif(20))

### Create a series of angles
rad <- seq(0, pi, l=20)

opar <- par(mfrow=c(5,4), mar=c(3,3,1,1))
for(i in rad){
  coords.rot <- Rotation(coords, i)
  plot(coords.rot, xlab="", ylab="")
}
par(opar)
```
### Rotate the coordinates by an angle of 90 degrees
```r
coords.90 <- Rotation(coords, 90*pi/180)
plot(coords, xlim=range(rbind(coords.90,coords)
[,1]),
     ylim=range(rbind(coords.90,coords)
[,2]), asp=1)
points(coords.90, pch=19)
```

---

**set.mcOption**

**Options for parallel support**

**Description**

Provides support for the use of parallel computation in the parallel package.

**Usage**

```r
set.mcOption(value)
get.mcOption()
set.coresOption(value)
get.coresOption()
set.ClusterOption(cl)
get.ClusterOption()
```

**Arguments**

- **value** valid replacement value
- **cl** a cluster object created by `makeCluster` in `parallel`

**Details**

Options in the spdep package are held in an environment local to the package namespace and not exported. Option values are set and retrieved with pairs of access functions, `get` and `set`. The `mc` option is set by default to `FALSE` on Windows systems, as they cannot fork the R session; by default it is `TRUE` on other systems, but may be set `FALSE`. If `mc` is `FALSE`, the `Cluster` option is used: if `mc` is `FALSE` and the `Cluster` option is `NULL` no parallel computing is done, or the `Cluster` option is passed a “cluster” object created by the parallel or snow package for access without being passed as an argument. The `cores` option is set to `NULL` by default, and can be used to store the number of cores to use as an integer. If `cores` is `NULL`, facilities from the parallel package will not be used.

**Value**

The option access functions return their current settings, the assignment functions usually return the previous value of the option.
Note

An extended example is shown in the documentation of `aple.mc`, including treatment of seeding of RNG for multicore/cluster.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

Examples

```r
ls(envir=spdep:::.spdepOptions)
library(parallel)
nc <- detectCores(logical=FALSE)
nc
# set nc to 1L here
if (nc > 1L) nc <- 1L
nc <- ifelse(nc > 2L, 2L, nc)
coresOpt <- get.coresOption()
coresOpt
if (!is.na(nc)) {
  invisible(set.coresOption(nc))
  print(exists("aple.mc"))
  if(.Platform$OS.type == "windows") {
    # forking not permitted on Windows - start cluster
    print(get.mcOption())
    cl <- makeCluster(get.coresOption())
    print(clusterEvalQ(cl, exists("aple.mc")))
    set.ClusterOption(cl)
    clusterEvalQ(get.ClusterOption(), library(spdep))
    print(clusterEvalQ(cl, exists("aple.mc")))
    clusterEvalQ(get.ClusterOption(), detach(package:spdep))
    set.ClusterOption(NULL)
    print(clusterEvalQ(cl, exists("aple.mc")))
    stopCluster(cl)
  } else {
    mcOpt <- get.mcOption()
    print(mcOpt)
    print(mclapply(1:get.coresOption(), function(i) exists("aple.mc"),
      mc.cores=get.coresOption()))
    invisible(set.mcOption(FALSE))
    cl <- makeCluster(nc)
    print(clusterEvalQ(cl, exists("aple.mc")))
    set.ClusterOption(cl)
    clusterEvalQ(get.ClusterOption(), library(spdep))
    print(clusterEvalQ(cl, exists("aple.mc")))
    clusterEvalQ(get.ClusterOption(), detach(package:spdep))
    set.ClusterOption(NULL)
    print(clusterEvalQ(cl, exists("aple.mc")))
    stopCluster(cl)
    invisible(set.mcOption(mcOpt))
  }
  invisible(set.coresOption(coresOpt))
```
set.spChkOption

Control checking of spatial object IDs

Description

Provides support for checking the mutual integrity of spatial neighbour weights and spatial data; similar mechanisms are used for passing global verbose and zero.policy options, and for providing access to a running cluster for embarrassingly parallel tasks.

Usage

set.spChkOption(check)
get.spChkOption()
chkIDs(x, listw)
spNamedVec(var, data)
set.VerboseOption(check)
get.VerboseOption()
set.ZeroPolicyOption(check)
get.ZeroPolicyOption()
set.listw_is_CsparseMatrix_Option(check)
get.listw_is_CsparseMatrix_Option()

Arguments

check  a logical value, TRUE or FALSE
x      a vector the same length, or a two-dimensional array, or data frame with the same number of rows as the neighbours list in listw
listw  a listw object or nb object inheriting from "nb"
var    a character string or integer value for the column to be selected
data   a two-dimensional array or data frame containing var

Details

Analysis functions will have an spChk argument by default set to NULL, and will call get.spChkOption() to get the global spatial option for whether to check or not — this is initialised to FALSE, and consequently should not break anything. It can be changed to TRUE using set.spChkOption(TRUE), or the spChk argument can be assigned in analysis functions. spNamedVec() is provided to ensure that rownames are passed on to single columns taken from two-dimensional arrays and data frames.

Value

set.spChkOption() returns the old logical value, get.spChkOption() returns the current logical value, and chkIDs() returns a logical value for the test lack of difference. spNamedVec() returns the selected column with the names set to the row names of the object from which it has been extracted.
Note

The motivation for this mechanism is provided by the observation that spatial objects on a map and their attribute data values need to be linked uniquely, to avoid spurious results. The reordering between the legacy Columbus data set used the earlier publications and that available for download from the Spacestat website is just one example of a common problem.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

Examples

data(oldcol)
rownames(COL.OLD)
data(columbus, package="spData")
rownames(columbus)
get.spChkOption()
oldChk <- set.spChkOption(TRUE)
get.spChkOption()
chkIDs(COL.OLD, nb2listw(COL.nb))
chkIDs(columbus, nb2listw(col.gal.nb))
chkIDs(columbus, nb2listw(COL.nb))
tmp <- try(moran.test(spNamedVec("CRIME", COL.OLD), nb2listw(COL.nb)))
print(tmp)
tmp <- try(moran.test(spNamedVec("CRIME", columbus), nb2listw(col.gal.nb)))
print(tmp)
tmp <- try(moran.test(spNamedVec("CRIME", columbus), nb2listw(COL.nb)))
print(tmp)
set.spChkOption(FALSE)
get.spChkOption()
moran.test(spNamedVec("CRIME", columbus), nb2listw(COL.nb))
tmp <- try(moran.test(spNamedVec("CRIME", columbus), nb2listw(COL.nb), spChk=TRUE), silent=TRUE)
print(tmp)
set.spChkOption(oldChk)
get.spChkOption()

skater

Spatial 'K'uster Analysis by Tree Edge Removal

Description

This function implements a SKATER procedure for spatial clustering analysis. This procedure essentially begins with an edges set, a data set and a number of cuts. The output is an object of 'skater' class and is valid for input again.
Usage

skater(edges, data, ncuts, crit, vec.crit, method = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski", "mahalanobis"), p = 2, cov, inverted = FALSE)

Arguments

edges A matrix with 2 columns with each row is an edge
data A data.frame with data observed over nodes.
cuts The number of cuts
crit A scalar or two dimensional vector with criteria for groups. Examples: limits of group size or limits of population size. If scalar, is the minimum criteria for groups.
vec.crit A vector for evaluating criteria.
method Character or function to declare distance method. If method is character, method must be "mahalanobis" or "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowsk". If method is one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski", see dist for details, because this function is used to compute the distance. If method="mahalanobis", the mahalanobis distance is computed between neighbour areas. If method is a function, this function is used to compute the distance.
p The power of the Minkowski distance.
cov The covariance matrix used to compute the mahalanobis distance.
inverted logical. If 'TRUE', 'cov' is supposed to contain the inverse of the covariance matrix.

Value

A object of skater class with:

groups A vector with length equal the number of nodes. Each position identifies the group of node
edges.groups A list of length equal the number of groups with each element is a set of edges
not.prune A vector identifying the groups with are not candidates to partition.
candidates A vector identifying the groups with are candidates to partition.
ssto The total dissimilarity in each step of edge removal.

Author(s)

Renato M. Assuncao and Elias T. Krainski
References


See Also

See Also as mstree

Examples

```r
### loading data
bh <- st_read(system.file("etc/shapes/bhicv.shp", package="spdep")[[1]], quiet=TRUE)
st_crs(bh) <- "+proj=longlat +ellps=WGS84"
### data standardized
dpad <- data.frame(scale(as.data.frame(bh)[,5:8]))
### neighboorhod list
bh.nb <- poly2nb(bh)

### calculating costs
lcosts <- nbcosts(bh.nb, dpad)

### making listw
nb.w <- nb2listw(bh.nb, lcosts, style="B")

### find a minimum spanning tree
mst.bh <- mstree(nb.w,5)

### the mstree plot
par(mar=c(0,0,0,0))
plot(st_geometry(bh), border=gray(.5))
plot(mst.bh, coordinates(as(bh, "Spatial")), col=2,
     cex.lab=.6, cex.circles=0.035, fg="blue", add=TRUE)

### three groups with no restriction
res1 <- skater(mst.bh[,1:2], dpad)

### groups size
table(res1$groups)

### the skater plot
opar <- par(mar=c(0,0,0,0))
plot(res1, coordinates(as(bh, "Spatial")), cex.circles=0.035, cex.lab=.7)

### the skater plot, using other colors
plot(res1, coordinates(as(bh, "Spatial")), cex.circles=0.035, cex.lab=.7,
groups.colors=heat.colors(length(res1$ed)))
```
### the Spatial Polygons plot

```r
plot(st_geometry(bh), col=heat.colors(length(res1$edg))[res1$groups])
```

```r
par(opar)
```

### EXPERT OPTIONS

### more one partition

```r
res1b <- skater(res1, dpad, 1)
```

### length groups frequency

```r
table(res1$groups)
```

```r
table(res1b$groups)
```

### thee groups with minimum population

```r
res2 <- skater(mst.bh[,1:2], dpad, 2, 200000, bh$Pop)
```

```r
table(res2$groups)
```

### thee groups with minimum number of areas

```r
res3 <- skater(mst.bh[,1:2], dpad, 2, 3, rep(1,nrow(bh)))
```

```r
table(res3$groups)
```

### thee groups with minimum and maximum number of areas

```r
res4 <- skater(mst.bh[,1:2], dpad, 2, c(20,50), rep(1,nrow(bh)))
```

```r
table(res4$groups)
```

### if I want to get groups with 20 to 40 elements

```r
res5 <- skater(mst.bh[,1:2], dpad, 2, c(20,40), rep(1,nrow(bh))) ## DON'T MAKE DIVISIONS
```

```r
table(res5$groups)
```

### In this MST don't have groups with this restrictions

```r
## In this case, first I do one division
```

```r
### with the minimum criteria
```

```r
res5a <- skater(mst.bh[,1:2], dpad, 1, 20, rep(1,nrow(bh)))
```

```r
table(res5a$groups)
```

### and do more one division with the full criteria

```r
res5b <- skater(res5a, dpad, 1, c(20, 40), rep(1,nrow(bh)))
```

```r
table(res5b$groups)
```

### and do more one division with the full criteria

```r
res5c <- skater(res5b, dpad, 1, c(20, 40), rep(1,nrow(bh)))
```

```r
table(res5c$groups)
```

### It don't have another division with this criteria

```r
res5d <- skater(res5c, dpad, 1, c(20, 40), rep(1,nrow(bh)))
```

```r
table(res5d$groups)
```

### Not run:

```r
data(boston, package="spData")
bh.nb <- boston.soi
dpad <- data.frame(scale(boston.c[,c(7:10)]))
```
### calculating costs
```
system.time(lcosts <- nbcosts(bh.nb, dpad))
```

### making listw
```
wb <- nb2listw(bh.nb, lcosts, style="B")
```

### find a minimum spanning tree
```
mst.bh <- mstree(wb, 5)
```

### three groups with no restriction
```
system.time(res1 <- skater(mst.bh[,1:2], dpad, 2))
```

### using parallel
```
library(parallel)
c <- detectCores(logical=FALSE)
# set nc to 1L here
if (c > 1L) c <- 1L
coreOpt <- get.coresOption()
invisible(set.coresOption(c))
if(!get.mcOption()) {
  # no-op, "snow" parallel calculation not available
  cl <- makeCluster(get.coresOption())
  set.ClusterOption(cl)
}

### calculating costs
```
system.time(plcosts <- nbcosts(bh.nb, dpad))
```

```
all.equal(lcosts, plcosts, check.attributes=FALSE)

### making listw
```
pwb <- nb2listw(bh.nb, plcosts, style="B")
```

### find a minimum spanning tree
```
pmst.bh <- mstree(pwb, 5)
```

### three groups with no restriction
```
system.time(pres1 <- skater(pwb[,1:2], dpad, 2))
```

if(!get.mcOption()) {
  # no-op, "snow" parallel calculation not available
  cl <- makeCluster(NULL)
  stopCluster(cl)
}
```

```
all.equal(res1, pres1, check.attributes=FALSE)
```

invisible(set.coresOption(coresOpt))

## End(Not run)

---

**sp.correlogram**

**Spatial correlogram**

**Description**

Spatial correlograms for Moran’s I and the autocorrelation coefficient, with print and plot helper functions.

**Usage**

```
sp.correlogram(neighbours, var, order = 1, method = "corr",
               style = "W", randomisation = TRUE, zero.policy = NULL, spChk=NULL)
```

## S3 method for class 'spcor'
plot(x, main, ylab, ylim, ...)  
## S3 method for class 'spcor'
print(x, p.adj.method="none", ...)

Arguments

- **neighbours**: an object of class nb
- **var**: a numeric vector
- **order**: maximum lag order
- **method**: "corr" for correlation, "I" for Moran’s I, "C" for Geary’s C
- **style**: style can take values W, B, C, and S
- **randomisation**: variance of I or C calculated under the assumption of randomisation, if FALSE normality
- **zero.policy**: default NULL, use global option value; if FALSE stop with error for any empty neighbour sets, if TRUE permit the weights list to be formed with zero-length weights vectors
- **spChk**: should the data vector names be checked against the spatial objects for identity integrity, TRUE, or FALSE, default NULL to use get.spChkOption()
- **x**: an object from sp.correlogram() of class spcor
- **p.adj.method**: correction method as in p.adjust
- **main**: an overall title for the plot
- **ylab**: a title for the y axis
- **ylim**: the y limits of the plot
- **...**: further arguments passed through

Details

The print function also calculates the standard deviates of Moran’s I or Geary’s C and a two-sided probability value, optionally using p.adjust to correct by the number of lags. The plot function plots a bar from the estimated Moran’s I, or Geary’s C value to +/- twice the square root of its variance (in previous releases only once, not twice). The table includes the count of included observations in brackets after the lag order. Care needs to be shown when interpreting results for few remaining included observations as lag order increases.

Value

returns a list of class spcor:

- **res**: for "corr" a vector of values; for "I", a matrix of estimates of "I", expectations, and variances
- **method**: "I" or "corr"
- **cardnos**: list of tables of neighbour cardinalities for the lag orders used
- **var**: variable name
Author(s)
Roger Bivand, <Roger.Bivand@nhh.no>

References

See Also
nblag, moran, p.adjust

Examples
nc.sids <- st_read(system.file("shapes/sids.shp", package="spData")[[1]], quiet=TRUE)
rn <- as.character(nc.sids$FIPS)
ncCC89_nb <- read.gal(system.file("weights/ncCC89.gal", package="spData")[[1]],
      region.id=rn)
ft.SID74 <- sqrt(1000)*(sqrt(nc.sids$SID74/nc.sids$BIR74) + 
      sqrt((nc.sids$SID74+1)/nc.sids$BIR74))
tr.SIDS74 <- ft.SID74*sqrt(nc.sids$BIR74)
cspc <- sp.correlogram(ncCC89_nb, tr.SIDS74, order=8, method="corr",
      zero.policy=TRUE)
print(cspc)
plot(cspc)
Ispc <- sp.correlogram(ncCC89_nb, tr.SIDS74, order=8, method="I",
      zero.policy=TRUE)
print(Ispc)
plot(Ispc, "bonferroni")
plot(Ispc)
Cspc <- sp.correlogram(ncCC89_nb, tr.SIDS74, order=8, method="C",
      zero.policy=TRUE)
print(Cspc)
plot(Cspc, "bonferroni")
plot(Cspc)
drop.no.neighs <- !(1:length(ncCC89_nb) %in% which(card(ncCC89_nb) == 0))
sub.ncCC89.nb <- subset(ncCC89_nb, drop.no.neighs)
plot(sp.correlogram(sub.ncCC89.nb, subset(tr.SIDS74, drop.no.neighs),
      order=8, method="corr"))

Description
A permutation test for the spatial general cross product statistic with Moran ($C_{ij} = z_i z_j$), Geary ($C_{ij} = (z_i - z_j)^2$), and Sokal ($C_{ij} = |z_i - z_j|$) criteria, for $z_i = (x_i - \bar{x})/\sigma_x$. plot.mc.sim is a helper function to plot the outcomes of the permutation test.
Usage

sp.mantel.mc(var, listw, nsim, type = "moran", zero.policy = NULL,
alternative = "greater", spChk=NULL, return_boot=FALSE)
## S3 method for class 'mc.sim'
plot(x, xlim, xlab, main, sub, ..., ptype="density")

Arguments

var a numeric vector the same length as the neighbours list in listw
listw a listw object created for example by nb2listw
nsim number of permutations
type "moran", "geary" or "sokal" criteria for similarity
zero.policy default NULL, use global option value; if TRUE assign zero to the lagged value
of zones without neighbours, if FALSE assign NA
alternative a character string specifying the alternative hypothesis, must be one of "greater"
(default), or "less".
spChk should the data vector names be checked against the spatial objects for identity
integrity, TRUE, or FALSE, default NULL to use get.spChkOption()
return_boot return an object of class boot from the equivalent permutation bootstrap rather
than an object of class htest
x the object to be plotted
xlim the range of the x axis
xlab a title for the x axis
main an overall title for the plot
sub a sub title for the plot
ptype either "density" or "hist"
... further arguments passed through

Value

A list with class htest and mc.sim containing the following components:

statistic the value of the observed Geary’s C.
parameter the rank of the observed Geary’s C.
alternative a character string describing the alternative hypothesis.
method a character string giving the method used.
data.name a character string giving the name(s) of the data, and the number of simulations.
p.value the pseudo p-value of the test.
res nsim simulated values of statistic, final value is observed statistic
estimate the mean and variance of the simulated distribution.
Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References

Cliff, A. D., Ord, J. K. 1981 Spatial processes, Pion, p. 22-24, Haining, R. 1990 *Spatial data analysis in the social and environmental sciences*, Cambridge: Cambridge University Press, p. 230–1. The function has been checked against general matrix code posted to the r-help list by Ben Bolker on 1 May 2001; another mantel() function is in the vegan package.

See Also

moran.mc, joincount.mc, geary.mc

Examples

data(oldcol)
sim1 <- sp.mantel.mc(COL.OLD$CRIME, nb2listw(COL.nb), 
nsim=99, type="geary", alternative="less")
sim2
plot(sim)
sp.mantel.mc(COL.OLD$CRIME, nb2listw(COL.nb), nsim=99, 
type="sokal", alternative="less")
sp.mantel.mc(COL.OLD$CRIME, nb2listw(COL.nb), nsim=99, 
type="moran")

spdep

Return package version number

Description

The function retrieves package version and build information

Usage

spdep(build = FALSE)

Arguments

build if TRUE, also returns build information

Value

a character vector with one or two elements

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>
Description

These functions are defunct from release 1.2-1. The functions have been moved to the `spatialreg` package.

Usage

```r
ape.plot()
localAple()
ape.mc()
ape()
lextrB()
lextrW()
lextrS()
griffith_sone()
subgraph_eigenw()
mom_calc()
mom_calc_int2()
stsls()
## S3 method for class 'stsls'
impacts()
GMerorsar()
## S3 method for class 'gmsar'
summary()
gstsls()
## S3 method for class 'gmsar'
impacts()
## S3 method for class 'gmsar'
Hausman.test()
lagmess()
ME()
SpatialFiltering()
LR.sarlm()
## S3 method for class 'sarlm'
logLik()
LR1.sarlm()
Wald1.sarlm()
## S3 method for class 'sarlm'
Hausman.test()
as.spam.listw()
as_dgRMatrix_listw()
as_dsTMatrix_listw()
as_dsCMatrix_I()
as_dsCMatrix_IrW()
```
Jacobian_W()
powerWeights()
## S3 method for class 'lagImpact'
plot()
## S3 method for class 'lagImpact'
print()
## S3 method for class 'lagImpact'
summary()
## S3 method for class 'lagImpact'
HPDinterval()
intImpacts()
can.be.simmed()
eigenw()
similar.listw()
do_ldet()
jacobianSetup()
cheb_setup()
mcdet_setup()
eigen_setup()
eigen_pre_setup()
spam_setup()
spam_update_setup()
Matrix_setup()
Matrix_J_setup()
LU_setup()
LU_prepermutate_setup()
moments_setup()
SE_classic_setup()
SE_whichMin_setup()
SE_interp_setup()
MCMCsamp()
## S3 method for class 'spautolm'
MCMCsamp()
## S3 method for class 'sarlm'
MCMCsamp()
spautolm()
## S3 method for class 'spautolm'
spautolm()
## S3 method for class 'MCMC_sar_g'
impacts()
## S3 method for class 'MCMC_sem_g'
impacts()
## S3 method for class 'MCMC_sac_g'
impacts()
spBreg_err()
spBreg_lag()
## S3 method for class 'SLX'
## S3 method for class 'SLX'
impacts()
create_WX()
## S3 method for class 'sarlm'
anova() bptest.sarlm() errorsarlm()
## S3 method for class 'sarlm'
impacts() lagsarlm()
## S3 method for class 'sarlm'
predict()
## S3 method for class 'sarlm.pred'
predict()
## S3 method for class 'sarlm.pred'
print() as.data.frame()
## S3 method for class 'sarlm'
residuals() deviance() coef() vcov()
## S3 method for class 'sarlm'
fitted()
sacsarlm()
## S3 method for class 'sarlm'
summary() pvalue()
## S3 method for class 'summary.sarlm'
print()

**Details**

Model-fitting functions and functions supporting model fitting have been moved to the `spatialreg` package.

**See Also**

`Defunct`
spweights.constants

Provides constants for spatial weights matrices

Description

The function calculates the constants needed for tests of spatial autocorrelation for general weights matrices represented as listw objects. Note: from spdep 0.3-32, the values of $S_1$ and $S_2$ are returned correctly for both underlying symmetric and asymmetric neighbour lists, before 0.3-32, $S_1$ and $S_2$ were wrong for listw objects based on asymmetric neighbour lists, such as k-nearest neighbours (thanks to Luc Anselin for finding the bug).

Usage

spweights.constants(listw, zero.policy=NULL, adjust.n=TRUE)
Szero(listw)

Arguments

listw a listw object from for example nb2listw
zero.policy default NULL, use global option value; if TRUE ignore zones without neighbours, if FALSE fail when encountered
adjust.n default TRUE, if FALSE the number of observations is not adjusted for no-neighbour observations, if TRUE, the number of observations is adjusted

Value

n number of zones
n1 n - 1
n2 n - 2
n3 n - 3
nn n * n
S0 global sum of weights
S1 $S_1$ sum of weights
S2 $S_2$ sum of weights

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

References

ssw

Compute the sum of dissimilarity

Description

This function computes the sum of dissimilarity between each observation and the mean (scalar of vector) of the observations.

Usage

```
ssw(data, id, method = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski", "mahalanobis"), p = 2, cov, inverted = FALSE)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>A matrix with observations in the nodes.</td>
</tr>
<tr>
<td>id</td>
<td>Node index to compute the cost</td>
</tr>
<tr>
<td>method</td>
<td>Character or function to declare distance method. If method is character, method must be &quot;mahalanobis&quot; or &quot;euclidean&quot;, &quot;maximum&quot;, &quot;manhattan&quot;, &quot;canberra&quot;, &quot;binary&quot; or &quot;minkowski&quot;. If method is one of &quot;euclidean&quot;, &quot;maximum&quot;, &quot;manhattan&quot;, &quot;canberra&quot;, &quot;binary&quot; or &quot;minkowski&quot;, see <code>dist</code> for details, because this function as used to compute the distance. If method=&quot;mahalanobis&quot;, the mahalanobis distance is computed between neighbour areas. If method is a function, this function is used to compute the distance.</td>
</tr>
<tr>
<td>p</td>
<td>The power of the Minkowski distance.</td>
</tr>
<tr>
<td>cov</td>
<td>The covariance matrix used to compute the mahalanobis distance.</td>
</tr>
<tr>
<td>inverted</td>
<td>logical. If 'TRUE', 'cov' is supposed to contain the inverse of the covariance matrix.</td>
</tr>
</tbody>
</table>
Value

A numeric, the sum of dissimilarity between the observations id of data and the mean (scalar of vector) of this observations.

Author(s)

Elias T. Krainski and Renato M. Assuncao

See Also

See Also as nbcost

Examples

data(USArrests)
n <- nrow(USArrests)
ssw(USArrests, 1:n)
ssw(USArrests, 1:(n/2))
ssw(USArrests, (n/2+1):n)
ssw(USArrests, 1:(n/2)) + ssw(USArrests, (n/2+1):n)

subset.listw  Subset a spatial weights list

Description

The function subsets a spatial weights list, retaining objects for which the subset argument vector is TRUE. At present it will only subset non-general weights lists (that is those created by nb2listw with glist=NULL).

Usage

## S3 method for class 'listw'
subset(x, subset, zero.policy = NULL, ...)

Arguments

x  an object of class listw
subset  logical expression
zero.policy  default NULL, use global option value; if FALSE stop with error for any empty neighbour sets, if TRUE permit the weights list to be formed with zero-length weights vectors - passed through to nb2listw
...  generic function pass-through
Value

The function returns an object of class `listw` with component `style` the same as the input object, component `neighbours` a list of integer vectors containing neighbour region number ids (compacted to run from 1:number of regions in subset), and component `weights` as the weights computed for neighbours using `style`.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

See Also

`nb2listw`, `subset.nb`

Examples

```r
col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[[1]])
to.be.dropped <- c(31, 34, 36, 39, 42, 46)
pre <- nb2listw(col.gal.nb)
print(pre)
post <- subset(pre, !(1:length(col.gal.nb) %in% to.be.dropped))
print(post)
```

**Description**

The function subsets a neighbors list, retaining objects for which the subset argument vector is TRUE.

Usage

```r
## S3 method for class 'nb'
subset(x, subset, ...)
```

Arguments

- `x` an object of class `nb`
- `subset` logical expression
- `...` generic function pass-through

Value

The function returns an object of class `nb` with a list of integer vectors containing neighbour region number ids (compacted to run from 1:number of regions in subset).
summary.nb

Author(s)
Roger Bivand <Roger.Bivand@nhh.no>

See Also

nb2listw

Examples

columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[1], quiet=TRUE)
col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[1])
coords <- coordinates(as(columbus, "Spatial"))
plot(col.gal.nb, coords)
to.be.dropped <- c(31, 34, 36, 39, 42, 46)
text(coords[to.be.dropped,1], coords[to.be.dropped,2], labels=to.be.dropped,
pos=2, offset=0.3)
sub.col.gal.nb <- subset(col.gal.nb,
!((1:length(col.gal.nb) %in% to.be.dropped))
plot(sub.col.gal.nb, coords[-to.be.dropped,], col="red", add=TRUE)
which(!(attr(col.gal.nb, "region.id") %in%
attr(sub.col.gal.nb, "region.id")))

summary.nb

Print and summary function for neighbours and weights lists

Description

The function prints summary measures for links in a neighbours list. If a matrix of coordinates is given as well, summary descriptive measures for the link lengths are also printed. Print and summary functions are also available for "listw" weights list objects, also reporting constants (S0, S1, S2) used in inference for global spatial autocorrelation statistics such as Moran's I, Geary's C, join-count tests and Getis-Ord G.

Usage

## S3 method for class 'nb'
summary(object, coords=NULL, longlat = NULL, scale = 1, ...)
## S3 method for class 'nb'
print(x, ...)
## S3 method for class 'listw'
summary(object, coords, longlat, zero.policy = NULL,
        scale = 1, ...)
## S3 method for class 'listw'
print(x, zero.policy = NULL, ...)
tolerance.nb

Function to construct edges based on a tolerance angle and a maximum distance

Description

This function creates an object of class nb (defined in the library spdep) containing a connexion diagram. The edges between sites are based on a tolerance angle and a maximum distance. The angle is directional; its direction is always from the bottom to the top of the screen.

Usage

```r
tolerance.nb(coords, unit.angle = "degrees", max.dist, tolerance, rot.angle, plot.sites=FALSE)
```
tolerance.nb

Arguments

- **coords**: A matrix or a data frame containing the X and Y coordinates of the study sites.
- **unit.angle**: Character. The measurement units in which angles are defined: either "degrees" (default) or "radians".
- **max.dist**: Numeric. The maximum distance of an edge linking two sites together.
- **tolerance**: Numeric. The tolerance angle in which a site can influence another site. The angle is measured vertically and from bottom to top of the pictures after rotation of the points.
- **rot.angle**: Numeric, optional. An angle at which a set of coordinates should be rotated before creating the connexion diagram. The set of coordinates is rotated counterclockwise. Negative values will produce a clockwise rotation.
- **plot.sites**: Logical (TRUE, FALSE) determining if the site should be plotted in a graphic window. This graph allows one to make sure the points are rotated in a correct direction.

Details

Even though this function creates a connexion diagram based on a tolerance angle going from the bottom to the top of the screen, the resulting object is symmetric, meaning that a site influences another and vice versa. The final object does not represent a directional connexion network.

Value

The function returns an object of class nb with a list of integer vectors corresponding to neighbour region numbers.

Warning

This function was not design to handle a large number of rows in coords. To use this function for a set of coordinates with more than 1500 entries is memory intensive.

Author(s)

F. Guillaume Blanchet

See Also

dnearneigh, cell2nb, graphneigh, tri2nb, knn2nb

Examples

```r
set.seed(1)
ex.data<-cbind(runif(50),rexp(50))

### Construct object of class nb with a tolerance angle of 30 degrees
### and a maximum distance of 2 m.
 nb.ex<-tolerance.nb(ex.data, unit.angle = "degrees", max.dist=1,
tolerance = 30)
```
### Construct object of class nb with a tolerance angle of 30 degrees and a maximum distance of 2 m. The coordinates are rotated at an angle of 45 degrees counterclockwise.

```r
nb.ex2 <- tolerance.nb(ex.data, unit.angle = "degrees", max.dist=1, tolerance = 30, rot.angle = 45)
```

### Construct object of class nb with a tolerance angle of \(\pi/8\) radians and a maximum distance of 1.5 m. The coordinates are rotated at an angle of \(\pi/4\) radians clockwise.

```r
nb.ex3 <- tolerance.nb(ex.data, unit.angle = "radians", max.dist=1.5, tolerance = \(\pi/8\), rot.angle = \(-\pi*2/3\))
```

```r
par(mfrow=c(1,3))
plot(nb.ex, ex.data, asp=1)
plot(nb.ex2, ex.data, asp=1)
plot(nb.ex3, ex.data, asp=1)
```

---

**tri2nb**

*Neighbours list from tri object*

### Description

The function uses the `deldir` package to convert a matrix of two-dimensional coordinates into a neighbours list of class `nb` with a list of integer vectors containing neighbour region number ids.

### Usage

```r
tri2nb(coords, row.names = NULL)
```

### Arguments

- **coords**: matrix of point coordinates with two columns, a `SpatialPoints` object or an `sfc` points object
- **row.names**: character vector of region ids to be added to the neighbours list as attribute `region.id`; default `seq(1, nrow(x))`

### Details

If coordinates are duplicated, this function cannot be used. If the coordinates are from a grid, then they need to be ordered such that the first three are not collinear, so that the first triangle can be constructed. This can be achieved by randomising the order of the coordinates (possibly several times), and then re-ordering the order of the data to match the new order of the neighbour list - if this fix is used, remember to re-order the row.names argument as well as the coordinates! Please also note that triangulation of grid points will give arbitrary diagonal neighbours, which may not be a sensible outcome, and `dnearneigh()` may serve better where `tri2nb()` cannot be used.
Value
The function returns an object of class `nb` with a list of integer vectors containing neighbour region number ids.

Author(s)
Roger Bivand <Roger.Bivand@nhh.no>

See Also
- `knn2nb`, `dnearneigh`, `cell2nb`

Examples
```r
# Read Columbus dataset
columbus <- st_read(system.file("shapes/columbus.shp", package="spData")[[1]], quiet=TRUE)
# Calculate centroid
coords <- st_centroid(st_geometry(columbus), of_largest_polygon=TRUE)
# Select row names
ind <- row.names(columbus)
# Require deldir package
suppressPackageStartupMessages(require(deldir))
# Convert to nb object
col.tri.nb <- tri2nb(coords, row.names=ind)
# Plot
plot(st_geometry(columbus), border="grey")
plot(col.tri.nb, coords, add=TRUE)
# Title
title(main="Raw triangulation links", cex.main=0.6)
# Create grid
x <- seq(0,1,0.1)
y <- seq(0,2,0.2)
xy <- expand.grid(x, y)
# Try
try(xy.nb <- tri2nb(xy))
# Seed
seed <- 1234
# Sample
xid <- sample(1:nrow(xy))
# Select
xy.nb <- tri2nb(xy[xid,])
# Plot
plot(xy.nb, xy[xid,])
```

---

**Write a neighbours list as a GAL lattice file**

**Description**
Write a neighbours list as a GAL lattice file, may also use newer GeoDa header format

**Usage**
```r
write.nb.gal(nb, file, oldstyle=TRUE, shpfile=NULL, ind=NULL)
```

**Arguments**
- `nb` an object of class `nb` with a list of integer vectors containing neighbour region number ids.
- `file` name of file with GAL lattice data
oldstyle if TRUE, first line of file contains only number of spatial units, if FALSE, uses newer GeoDa style
shpfile Shapefile name taken from GAL file for this dataset
ind region id indicator variable name

Author(s)
Roger Bivand <Roger.Bivand@nhh.no>

See Also
read.gal

Examples

col.gal.nb <- read.gal(system.file("weights/columbus.gal", package="spData")[1])
GALfile <- tempfile("GAL")
write.nb.gal(col.gal.nb, GALfile)
col.queen <- read.gal(GALfile)
summary(diffnb(col.queen, col.gal.nb))
Index

* cluster
  nbcosts, 109
  plot.skater, 117
  prunecost, 121
  prunemst, 123
  skater, 131
  ssw, 144
* datasets
  columbus, 13
  eire, 25
  oldcol, 113
* data
  bhicv, 8
* graphs
  mstree, 95
  prunecost, 121
* hplot
  plot.mst, 115
  plot.skater, 117
* manip
  Rotation, 127
* multivariate
  ssw, 144
* spatial
  aggregate.nb, 5
  airdist, 6
  autocov_dist, 6
  card, 9
  cell2nb, 10
  choynowski, 12
  diffnb, 14
  dnearyneigh, 15
  droplinks, 17
  EBlest, 19
  EB1moran.mc, 20
  EBlocal, 22
  edit.nb, 24
  geary, 25
  geary.mc, 26
  geary.test, 28
  globalG.test, 30
  Graph Components, 32
  graphneigh, 33
  grid2nb, 36
  include.self, 37
  is.symmetric.nb, 38
  joincount.mc, 40
  joincount.multi, 41
  joincount.test, 43
  knearyneigh, 45
  knn2nb, 47
  lag.listw, 48
  lee, 49
  lee.mc, 50
  lee.test, 52
  listw2sn, 54
  lm.LMtests, 55
  lm.morantest, 57
  lm.morantest.exact, 59
  lm.morantest.sad, 61
  localG, 71
  localmoran, 73
  localmoran.exact, 76
  localmoran.sad, 79
  LOSH, 82
  LOSH.cs, 84
  LOSH.mc, 85
  mat2listw, 87
  moran, 88
  moran.mc, 90
  moran.plot, 91
  moran.test, 93
  mstree, 95
  nb.set.operations, 97
  nb2blocknb, 98
  nb2INLA, 100
  nb2lines, 101
  nb2listw, 102
nb2listwdist, 104
nb2mat, 107
nb2WB, 108
nbcosts, 109
nbdist, 110
nblag, 111
p.adjustSP, 114
plot.nb, 116
poly2nb, 118
probmap, 120
read.gal, 124
read.gwt2nb, 125
set.mcOption, 128
set.spChkOption, 130
sp.correlogram, 135
sp.mantel.mc, 137
spdep, 139
spdep-defunct, 140
spweights.constants, 143
subset.listw, 145
subset.nb, 146
summary.nb, 147
tolerance.nb, 148
tri2nb, 150
write.nb.gal, 151

* tree

plot.mst, 115
prunemst, 123
skater, 131

aggregate.nb, 5
airdist, 6
anova.sarlm (spdep-defunct), 140
aple (spdep-defunct), 140
aple.mc, 129
as.data.frame.localmoranex
  (localmoran.exact), 76
as.data.frame.localmoransad
  (localmoran.sad), 79
as.data.frame.sarlm.pred
  (spdep-defunct), 140
as.spam.listw (spdep-defunct), 140
as_dgRMatrix_listw (spdep-defunct), 140
as_dsCMatrix_I (spdep-defunct), 140
as_dsCMatrix_IrW (spdep-defunct), 140
as_dsTMatrix_listw (spdep-defunct), 140
autocov_dist, 6

bbs (columbus), 13

bhicv, 8
bptest.sarlm (spdep-defunct), 140
can.be.simmed (spdep-defunct), 140
card, 9, 11, 16, 34, 35, 37, 47, 118, 119
cell2nb, 10, 99, 149, 151
cheb_setup (spdep-defunct), 140
chkIDs (set.spChkOption), 130
choynowski, 12
coeff.gmsar (spdep-defunct), 140
coeff.lagmess (spdep-defunct), 140
coeff.sarlm (spdep-defunct), 140
coeff.spautolm (spdep-defunct), 140
coeff.stslls (spdep-defunct), 140
corner, listw, CsparseMatrix-method
  (spdep-defunct), 140
corner, listw, RsparseMatrix-method
  (spdep-defunct), 140
corner, listw, symmetricMatrix-method
  (spdep-defunct), 140
col.gal.nb (columbus), 13
COL.nb (oldcol), 113
COL.OLD (oldcol), 113
columbus, 13
complement.nb (nb.set.operations), 97
goords (columbus), 13
create_WX (spdep-defunct), 140

Defunct, 142
deviance.gmsar (spdep-defunct), 140
deviance.lagmess (spdep-defunct), 140
deviance.sarlm (spdep-defunct), 140
deviance.spautolm (spdep-defunct), 140
deviance.stslls (spdep-defunct), 140
df2sn (nb2lines), 101
diffnb, 14
dist, 110, 122, 132, 144
dinearneigh, 15, 35, 46, 99, 149, 151
do_ldet (spdep-defunct), 140
dput, 109
droplinks, 17

EBest, 19, 22, 23, 121
EBImoran (EBImoran.mc), 20
EBImoran.mc, 20, 20
EBlocal, 20, 22, 121
edit.nb, 24
eigen_pre_setup (spdep-defunct), 140
eigen_setup (spdep-defunct), 140
LOSH.mc, 84, 85, 85, 87
LR.sarlm (spdep-defunct), 140
LR1.sarlm (spdep-defunct), 140
LR1.spautolm (spdep-defunct), 140
LU_prepermutate_setup (spdep-defunct), 140
LU_setup (spdep-defunct), 140
make.sym.nb (is.symmetric.nb), 38
mat2listw, 87
Matrix_J_setup (spdep-defunct), 140
Matrix_setup (spdep-defunct), 140
mcdet_setup (spdep-defunct), 140
MCMCsamp (spdep-defunct), 140
mom_calc (spdep-defunct), 140
mom_calc_int2 (spdep-defunct), 140
moments_setup (spdep-defunct), 140
moran, 22, 88, 91, 94, 137
moran.gc, 22, 89, 90, 94, 139
moran.plot, 91
moran.test, 89, 91, 93
mstree, 95, 115, 117, 123, 133
n.comp.nb (Graph Components), 32
nb.set.operations, 97
nb2blocknb, 98
nb2INLA, 100
nb2lines, 101
nb2listw, 7, 48, 54, 87, 88, 95, 102, 106, 108, 110, 111, 114, 146, 147
nb2listwdist, 104
nb2mat, 88, 107
nb2WB, 108
nbcost, 145
nbcost (nbcosts), 109
nbcosts, 109
nbdist, 110, 110
nblag, 111, 137
nbflag_cumu (nbflag), 111
old.make.sym.nb (is.symmetric.nb), 38
oldcol, 113
p.adjust, 56, 114, 137
p.adjustSP, 84, 86, 114
plot.Gabriel (graphneigh), 33
plot.lagImpact (spdep-defunct), 140
plot.listw (plot.nb), 116
plot.mc.sim (sp.mantel.mc), 137
plot.mst, 115
plot.nb, 24, 32, 116, 148
plot.relative (graphneigh), 33
plot.skater, 117
plot.spcor (sp.correlogram), 135
poly2nb, 37, 99, 110, 118
polys (columbus), 13
powerWeights (spdep-defunct), 140
ppois, 121
predict.sarlm (spdep-defunct), 140
predict.SLX (spdep-defunct), 140
print.gmsar (spdep-defunct), 140
print.jclist (joincount.test), 43
print.jcmulti (joincount.multi), 41
print.lagImpact (spdep-defunct), 140
print.lagmess (spdep-defunct), 140
print.listw (summary.nb), 147
print.LMtestlist (lm.LMtests), 55
print.localmoransad (localmoransad), 76
print.localmoransad (localmoransad), 79
print.ME_res (spdep-defunct), 140
print.moransad (lm.morantest.exact), 59
print.moransad (lm.morantest.exact), 61
print.nb (summary.nb), 147
print.sarlm (spdep-defunct), 140
print.SFResult (spdep-defunct), 140
print.spautolm (spdep-defunct), 140
print.spcor (sp.correlogram), 135
print.stsls (spdep-defunct), 140
print.summary.gmsar (spdep-defunct), 140
print.summary.lagImpact (spdep-defunct), 140
print.summary.lagmess (spdep-defunct), 140
print.summary.localmoransad (localmoransad), 79
print.summary/localmoransad (localmoransad), 79
print.summary.moransad (lm.morantest.exact), 59
print.summary.moransad (lm.morantest.exact), 61
print.summary.nb (summary.nb), 147
print.summary.sarlm (spdep-defunct), 140
print.summary.spautolm (spdep-defunct), 140
print.summary.stsls (spdep-defunct), 140
prunemst, 122, 123
read.dat2listw (read.gwt2nb), 125
INDEX

read.gal, 39, 103, 124, 126, 152
read.geoda (read.gal), 124
read.gwt2nb, 125
relativeneigh (graphneigh), 33
residuals.gmsar (spdep-defunct), 140
residuals.lagmess (spdep-defunct), 140
residuals.sarlm (spdep-defunct), 140
residuals.spautolm (spdep-defunct), 140
residuals.stsels (spdep-defunct), 140
Rotation, 127
sacsarlm (spdep-defunct), 140
SE_classic_setup (spdep-defunct), 140
SE_interp_setup (spdep-defunct), 140
SE_whichMin_setup (spdep-defunct), 140
set.ClusterOption (set.mcOption), 128
set.coresOption (set.mcOption), 128
set.listw_is_CsparseMatrix_Option
(set.spChkOption), 130
set.mcOption, 128
set.spChkOption, 130
set.VerboseOption (set.spChkOption), 130
set.ZeroPolicyOption (set.spChkOption), 130
setdiff.nb, 97
setdiff.nb (nb.set.operations), 97
similar.listw (spdep-defunct), 140
skater, 115, 117, 131
sn2listw, 102
sn2listw (listw2sn), 54
soi.graph (graphneigh), 33
sp.correlogram, 135
sp.mantel.mc, 26, 137
spam_setup (spdep-defunct), 140
spam_update_setup (spdep-defunct), 140
SpatialFiltering (spdep-defunct), 140
spautolm (spdep-defunct), 140
spBreg_err (spdep-defunct), 140
spBreg_lag (spdep-defunct), 140
spBreg_sac (spdep-defunct), 140
spdep, 139
spdep-defunct, 140
spNamedVec (set.spChkOption), 130
spweights.constants, 143
ssw, 144
stsels (spdep-defunct), 140
subgraph_eigenw (spdep-defunct), 140
subset.listw, 145
subset.nb, 146, 146
summary.gmsar (spdep-defunct), 140
summary.lagImpact (spdep-defunct), 140
summary.lagmess (spdep-defunct), 140
summary.listw (summary.nb), 147
summary.LMtestlist (lm.LMtests), 55
summary.localmoransad (localmoran.sad), 79
summary.moransad (lm.morantest.sad), 61
summary.nb, 10, 11, 24, 37, 38, 103, 106, 111
112, 116, 119, 125, 147
summary.sarlm (spdep-defunct), 140
summary.spatolm (spdep-defunct), 140
summary.stsels (spdep-defunct), 140
sym.attr.nb (is.symmetric.nb), 38
Szero (spweights.constants), 143
tolerance.nb, 148
tri2nb, 99, 149, 150
trW (spdep-defunct), 140
union.nb, 97
union.nb (nb.set.operations), 97
vcov.sarlm (spdep-defunct), 140
vi2mrc (cell2nb), 10
Wald1.sarlm (spdep-defunct), 140
write.nb (read.gwt2nb), 125
write.sn2dat (read.gwt2nb), 125
write.sn2gwt (read.gwt2nb), 125