Package ‘spatialEco’

May 14, 2021

Type Package

Title Spatial Analysis and Modelling Utilities

Version 1.3-7

Date 2021-05-12

Description Utilities to support spatial data manipulation, query, sampling and modelling. Functions include models for species population density, download utilities for climate and global deforestation spatial products, spatial smoothing, multivariate separability, point process model for creating pseudo-absences and sub-sampling, polygon and point-distance landscape metrics, auto-logistic model, sampling models, cluster optimization, statistical exploratory tools and raster-based metrics.

Depends R (>= 4.0)

Imports sp, sf, raster, spatstat.geom, spatstat.core, spdep, rgeos, MASS, methods

Suggests exactextractr, cluster, readr, RCurl, RANN, yaImpute, SpatialPack (>= 0.3), mgcv, EnvStats, maptools, GeNetIt, gstat, RStoolbox, terra, tabularaster, stringr

Maintainer Jeffrey S. Evans <jeffrey_evans@tnc.org>

License GPL-3

URL https://github.com/jeffreyevans/spatialEco

BugReports https://github.com/jeffreyevans/spatialEco/issues

NeedsCompilation no

Repository CRAN

LazyData true

Encoding UTF-8

RoxygenNote 7.1.1

Author Jeffrey S. Evans [aut, cre], Melanie A. Murphy [ctb], Karthik Ram [ctb]

Date/Publication 2021-05-14 18:30:02 UTC
**R topics documented:**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>annulus.matrix</td>
<td>5</td>
</tr>
<tr>
<td>ants</td>
<td>6</td>
</tr>
<tr>
<td>background</td>
<td>6</td>
</tr>
<tr>
<td>bearing.distance</td>
<td>8</td>
</tr>
<tr>
<td>breeding.density</td>
<td>9</td>
</tr>
<tr>
<td>built.index</td>
<td>10</td>
</tr>
<tr>
<td>cglis_urls</td>
<td>12</td>
</tr>
<tr>
<td>chae</td>
<td>14</td>
</tr>
<tr>
<td>chen</td>
<td>15</td>
</tr>
<tr>
<td>class.comparison</td>
<td>16</td>
</tr>
<tr>
<td>classBreaks</td>
<td>18</td>
</tr>
<tr>
<td>collinear</td>
<td>19</td>
</tr>
<tr>
<td>combine</td>
<td>20</td>
</tr>
<tr>
<td>concordance</td>
<td>22</td>
</tr>
<tr>
<td>conf.interval</td>
<td>23</td>
</tr>
<tr>
<td>cor.data</td>
<td>24</td>
</tr>
<tr>
<td>correlogram</td>
<td>24</td>
</tr>
<tr>
<td>cross.tab</td>
<td>25</td>
</tr>
<tr>
<td>crossCorrelation</td>
<td>27</td>
</tr>
<tr>
<td>csi</td>
<td>30</td>
</tr>
<tr>
<td>curvature</td>
<td>31</td>
</tr>
<tr>
<td>dahi</td>
<td>32</td>
</tr>
<tr>
<td>date_seq</td>
<td>33</td>
</tr>
<tr>
<td>daymet.point</td>
<td>34</td>
</tr>
<tr>
<td>daymet.tiles</td>
<td>35</td>
</tr>
<tr>
<td>DAYMET_tiles</td>
<td>36</td>
</tr>
<tr>
<td>dispersion</td>
<td>37</td>
</tr>
<tr>
<td>dissection</td>
<td>38</td>
</tr>
<tr>
<td>divergence</td>
<td>39</td>
</tr>
<tr>
<td>download.daymet</td>
<td>39</td>
</tr>
<tr>
<td>download.hansen</td>
<td>40</td>
</tr>
<tr>
<td>download.prism</td>
<td>42</td>
</tr>
<tr>
<td>effect.size</td>
<td>42</td>
</tr>
<tr>
<td>elev</td>
<td>43</td>
</tr>
<tr>
<td>erase.point</td>
<td>44</td>
</tr>
<tr>
<td>explode</td>
<td>45</td>
</tr>
<tr>
<td>extract.vertices</td>
<td>46</td>
</tr>
<tr>
<td>focal.lmetrics</td>
<td>47</td>
</tr>
<tr>
<td>fuzzySum</td>
<td>48</td>
</tr>
<tr>
<td>gaussian.kernel</td>
<td>49</td>
</tr>
<tr>
<td>geo.buffer</td>
<td>50</td>
</tr>
<tr>
<td>group.pdf</td>
<td>51</td>
</tr>
<tr>
<td>hexagons</td>
<td>52</td>
</tr>
<tr>
<td>hli</td>
<td>53</td>
</tr>
<tr>
<td>hsp</td>
<td>54</td>
</tr>
<tr>
<td>hybrid.kmeans</td>
<td>55</td>
</tr>
<tr>
<td>R topics documented:</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>---</td>
</tr>
<tr>
<td>idw.smoothing</td>
<td>56</td>
</tr>
<tr>
<td>impute.loess</td>
<td>57</td>
</tr>
<tr>
<td>insert</td>
<td>58</td>
</tr>
<tr>
<td>insert.values</td>
<td>59</td>
</tr>
<tr>
<td>is.empty</td>
<td>60</td>
</tr>
<tr>
<td>is.whole</td>
<td>61</td>
</tr>
<tr>
<td>kde.2D</td>
<td>62</td>
</tr>
<tr>
<td>kendall</td>
<td>62</td>
</tr>
<tr>
<td>kl.divergence</td>
<td>64</td>
</tr>
<tr>
<td>knn</td>
<td>65</td>
</tr>
<tr>
<td>land.metrics</td>
<td>66</td>
</tr>
<tr>
<td>local.min.max</td>
<td>67</td>
</tr>
<tr>
<td>loess.boot</td>
<td>68</td>
</tr>
<tr>
<td>loess.ci</td>
<td>70</td>
</tr>
<tr>
<td>logistic.regression</td>
<td>71</td>
</tr>
<tr>
<td>max_extent</td>
<td>74</td>
</tr>
<tr>
<td>moments</td>
<td>75</td>
</tr>
<tr>
<td>morans.plot</td>
<td>76</td>
</tr>
<tr>
<td>mwCorr</td>
<td>78</td>
</tr>
<tr>
<td>nni</td>
<td>78</td>
</tr>
<tr>
<td>nth.values</td>
<td>79</td>
</tr>
<tr>
<td>o.ring</td>
<td>80</td>
</tr>
<tr>
<td>oli.asw</td>
<td>81</td>
</tr>
<tr>
<td>optimal.k</td>
<td>83</td>
</tr>
<tr>
<td>optimized.sample.variance</td>
<td>84</td>
</tr>
<tr>
<td>outliers</td>
<td>85</td>
</tr>
<tr>
<td>overlap</td>
<td>86</td>
</tr>
<tr>
<td>parea.sample</td>
<td>87</td>
</tr>
<tr>
<td>parse.bits</td>
<td>89</td>
</tr>
<tr>
<td>partial.cor</td>
<td>90</td>
</tr>
<tr>
<td>plot.effect.size</td>
<td>92</td>
</tr>
<tr>
<td>plot.loess.boot</td>
<td>92</td>
</tr>
<tr>
<td>point.in.poly</td>
<td>93</td>
</tr>
<tr>
<td>poly.regression</td>
<td>95</td>
</tr>
<tr>
<td>polygon_extract</td>
<td>97</td>
</tr>
<tr>
<td>polyPerimeter</td>
<td>99</td>
</tr>
<tr>
<td>pp.subsample</td>
<td>100</td>
</tr>
<tr>
<td>print.cross.cor</td>
<td>102</td>
</tr>
<tr>
<td>print.effect.size</td>
<td>102</td>
</tr>
<tr>
<td>print.loess.boot</td>
<td>103</td>
</tr>
<tr>
<td>proximity.index</td>
<td>103</td>
</tr>
<tr>
<td>pseudo.absence</td>
<td>104</td>
</tr>
<tr>
<td>pu</td>
<td>107</td>
</tr>
<tr>
<td>quadrats</td>
<td>109</td>
</tr>
<tr>
<td>random.raster</td>
<td>110</td>
</tr>
<tr>
<td>raster.change</td>
<td>112</td>
</tr>
<tr>
<td>raster.deviation</td>
<td>114</td>
</tr>
<tr>
<td>raster.downscale</td>
<td>115</td>
</tr>
</tbody>
</table>
R topics documented:

- raster.entropy .............................................................. 117
- raster.gaussian.smooth .................................................. 118
- raster.invert ............................................................... 119
- raster.kendall .............................................................. 120
- raster.mds ................................................................. 122
- raster.modified.ttest .................................................... 123
- raster.moments ............................................................ 125
- raster.transformation ................................................... 126
- raster.vol ................................................................. 127
- raster.Zscore ............................................................. 128
- rasterCorrelation ......................................................... 129
- rasterDistance ............................................................ 130
- remove.holes ............................................................... 132
- rm.ext ........................................................................ 133
- rotate.polygon ............................................................ 133
- sa.trans ................................................................. 134
- sample.annulus ............................................................ 135
- sample.line ............................................................... 136
- sample.poly ............................................................... 138
- sampleTransect .......................................................... 139
- sar .......................................................................... 140
- se.news .................................................................... 141
- separability ............................................................... 141
- sg.smooth ................................................................. 143
- shannons ................................................................. 144
- shift ....................................................................... 146
- similarity ................................................................. 146
- smooth.time.series ...................................................... 148
- sobal ...................................................................... 149
- sp.kde .................................................................... 151
- sp.na.omit ............................................................... 153
- spatial.select ............................................................ 154
- spectral.separability .................................................... 156
- spherical.sd ............................................................. 157
- srr ................................................................. 159
- stratified.random .......................................................... 160
- subsample.distance ..................................................... 161
- summary.cross.cor ...................................................... 163
- summary.effect.size ..................................................... 163
- summary.loess.boot ...................................................... 164
- swvi ................................................................. 164
- time_to_event ............................................................ 166
- topo.distance ............................................................ 168
- tpi ................................................................. 169
- trasp ................................................................. 170
- trend.line ............................................................... 171
- tri ................................................................. 172
- vrm ................................................................. 173
annulus.matrix

Description

Creates a square matrix representing annulus position values of 1 and defined null

Usage

annulus.matrix(scale = 3, inner.scale = 0, outer.scale = 0, null.value = 0)

Arguments

scale Number of rings (defines dimensions of matrix)
inner.scale Number of inner rings to set to null.value
outer.scale Number of outer rings to set to null.value
null.value Value to set inner and outer scale(s) to

Value

A matrix object with defined null.value and 1, representing retained rings

Note

This function will return a matrix of 1 and defined null.value based on a specification of the scale, inner scale and outer scale. The scale defines how many rings will be represented in the matrix based on \((2 \times \text{scale} - 1)\). So, a scale of 3 will result in a 5x5 matrix. The inner.scale and outer.scale arguments represent the > and < rings that will be set to the defined null.value (see examples). The resulting matrix can be used as the specified window in a focal function.

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

annulus.matrix(5) # 5 concentric rings
annulus.matrix(5, 3) # 5 concentric rings with the 3 inner set to 0
annulus.matrix(5, 3, null.value=NA) # 5 concentric rings with the 3 inner set to NA
annulus.matrix(5, 3, 5) # 5 rings with 3 inner and 5 outer set to 0
annulus.matrix(9, 3, 7) # 9 rings with 3 inner and 7 outer set to 0
Description

Roth et al., (1994) Costa Rican ant diversity data

Format

A data.frame with 82 rows (species) and 5 columns (covetypres):

- **species** Ant species (family)
- **Primary.Forest** Primary forest type
- **Abandoned.cacao.plantations** Abandoned cacao plantations type
- **Productive.cacao.plantations** Active cacao plantations type
- **Banana.plantations** Active banana plantations type

Source

http://www.tiem.utk.edu/~gross/bioed/bealsmodules/shannonDI.html

References


Description

Creates a point sample that can be used as a NULL for SDM’s and other modeling approaches.

Usage

```r
background(
  x,
  ext = NULL,
  p = 1000,
  known = NULL,
  d = NULL,
  type = c("regular", "random", "hexagon", "nonaligned")
)
```
**background**

**Arguments**

- **x**: A polygon defining sample region
- **ext**: Vector of extent coordinates (xmin, xmax, ymin, ymax)
- **p**: Size of sample
- **known**: SpatialPoints of known locations (same CSR as x)
- **d**: Threshold distance for known proximity
- **type**: Type of sample c("systematic", "random", "hexagon", "nonaligned")

**Value**

A SpatialPointsDataFrame or data.frame with x,y coordinates

**Note**

This function creates a background point sample based on an extent or polygon sampling region. The known argument can be used with d to remove sample points based on distance-based proximity to existing locations (e.g., known species locations). The size (p) of the resulting sample will be dependent on the known locations and the influence of the distance threshold (d). As such, if the know and d arguments are provided the exact value provided in p will not be returned.

**Author(s)**

Jeffrey S. Evans <jeffrey_evans@tnc.org>

**Examples**

```r
library(sp)
library(raster)
library(rgeos)
data(meuse)
coordinates(meuse) <- ~x+y

# create "known" locations
locs <- meuse[sample(1:nrow(meuse), 5),]

# systematic sample using extent polygon
e <- as(extent(meuse), "SpatialPolygons")
s <- background(e, p=1000, known=locs, d=300)
plot(s,pch=20)
points(locs, pch=20, col="red")

# systematic sample using irregular polygon
data(meuse.grid)
coordinates(meuse.grid) = c("x", "y")
gridded(meuse.grid) = TRUE
meuse.poly = gUnaryUnion(as(meuse.grid, "SpatialPolygons"))
s <- background(meuse.poly, p=1000, known=locs, d=200)
plot(s,pch=20)
```
```r
# random sample using irregular polygon
s <- background(meuse.poly, p=500, known=locs, d=200, type="random")
plot(s, pch=20)
plot(meuse.poly, add=TRUE)
points(locs, pch=20, col="red")

# systematic sample using defined extent
extent(meuse)
s <- background(ext=c(178605, 181390, 329714, 333611),
          p=1000, known=locs, d=300)
plot(s, pch=20)
points(locs, pch=20, col="red")
```

---

**bearing.distance**

*Bearings and Distance*

**Description**

Calculates a new point \([X,Y]\) based on defined bearing and distance

**Usage**

```r
bearing.distance(x, y, distance, azimuth, EastOfNorth = TRUE)
```

**Arguments**

- `x`: x coordinate
- `y`: y coordinate
- `distance`: Distance to new point (in same units as x,y)
- `azimuth`: Azimuth to new point
- `EastOfNorth`: Specified surveying convention

**Value**

A new point representing location of bearing and distance

**Note**

East of north is a surveying convention and defaults to true.

**Author(s)**

Jeffrey S. Evans <jeffrey_evans@tnc.org>
breeding.density

Examples

pt <- cbind(x=480933, y=4479433)
bearing.distance(pt[1], pt[2], 1000, 40)

breeding.density Breeding density areas (aka, core habitat areas)

Description

Calculates breeding density areas base on population counts and spatial point density.

Usage

breeding.density(x, pop, p = 0.75, bw = 6400, b = 8500, self = TRUE)

Arguments

x
sp SpatialPointsDataFrame object

pop
Population count/density column in x@data

p
Target percent of population

bw
Bandwidth distance for the kernel estimate (default 8500)

b
Buffer distance (default 8500)

self
(TRUE/FALSE) Should source observations be included in density (default TRUE)

Value

A list object with:

• pop.pts sp point object with points identified within the specified p
• pop.area sp polygon object of buffered points specified by parameter b
• bandwidth Specified distance bandwidth used in identifying neighbor counts
• buffer Specified buffer distance used in buffering points for pop.area
• p Specified population percent

Note

The breeding density areas model identifies the Nth-percent population exhibiting the highest spatial density and counts/frequency. It then buffers these points by a specified distance to produce breeding area polygons. If you would like to recreate the results in Doherty et al., (2010), then define bw = 6400m and b[if p < 0.75 b = 6400m, | p >= 0.75 b = 8500m]

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>
References


Examples

```r
require(sp)
n=1500
bb <- rbind(c(-1281299,-761876.5),c(1915337,2566433.5))
bb.mat <- cbind(c(bb[1,1], bb[1,2], bb[1,2], bb[1,1]),
c(bb[2,1], bb[2,1], bb[2,2], bb[2,2]))
bbp <- Polygon(bb.mat)
s <- spsample(bbp, n, type="random")
pop <- SpatialPointsDataFrame(s, data.frame(ID=1:length(s),
counts=runif(length(s), 1, 250)))
bd75 <- breeding.density(pop, pop="counts", p=0.75, b=8500, bw=6400)
plot(bd75$pop.area, main="75% breeding density areas")
plot(pop, pch=20, col="black", add=TRUE)
plot(bd75$pop.pts, pch=20, col="red", add=TRUE)
```

```r

description
```
- band 7 2.09-2.35 (SWIR 2)

**OLI (Landsat 8)**
- band-3 0.53-0.59 (green)
- band-4 0.64-0.67 (red)
- band-5 0.85-0.88 (NIR)
- band-6 1.57-1.65 (SWIR 1)
- band-7 2.11-2.29 (SWIR 2)

**Usage**
```r
built.index(
  green,
  red,
  nir,
  swir1,
  swir2,
  L = 0.5,
  method = c("Bouhennache", "Zha", "Xu")
)
```

**Arguments**
- `green` Green band (0.53 - 0.59mm), landsat 5&7 band 3, OLI (landsat 8) band 3
- `red` Red band (0.636 - 0.673mm), landsat 5&7 band 3, OLI (landsat 8) band 4
- `nir` Near infrared band (0.851 - 0.879mm) landsat 5&7 band 4, OLI (landsat 8) band 5
- `swir1` short-wave infrared band 1 (1.566 - 1.651mm), landsat 5&7 band 5, OLI (landsat 8) band 6
- `swir2` short-wave infrared band 2 (2.11 - 2.29mm), landsat 5&7 band 7, OLI (landsat 8) band 7
- `L` The L factor for the savi index
- `method` Method to use for index options are "Bouhennache", "Zha", "Xu"

**Author(s)**
Jeffrey S. Evans [jeffrey_evans@tnc.org](mailto:jeffrey_evans@tnc.org)

**References**
Examples

```r
## Not run:
library(raster)
library(RStoolbox)

data(lsat)
lsat <- radCor(lsat, metaData = readMeta(system.file("external/landsat/LT52240631988227CUB02_MTL.txt", package="RStoolbox")), method = "apref")
plotRGB(lsat, r=3, g=2, b=1, scale=1.0, stretch="lin")

# Using Bouhennache et al., (2018) method (needs green, red, swir1 and swir2)
(bouh <- built.index(red = lsat[[3]], green = lsat[[4]], swir1 = lsat[[5]], swir2 = lsat[[7]]))
plotRGB(lsat, r=6,g=5,b=2, scale=1, stretch="lin")
plot(bouh, legend=FALSE, col=rev(terrain.colors(100, alpha=0.35)), add=TRUE )

# Using simple Zha et al., (2003) method (needs nir and swir1)
(zha <- built.index(nir = lsat[[4]], swir1 = lsat[[5]], method = "Zha") )
plotRGB(lsat, r=6,g=5,b=2, scale=1, stretch="lin")
plot(zha, legend=FALSE, col=rev(terrain.colors(100, alpha=0.35)), add=TRUE )

# Using Xu (2008) normalized modification of Zha (needs green, red, nir and swir1)
(xu <- built.index(green= lsat[[3]], red = lsat[[3]], nir = lsat[[4]], swir1 = lsat[[5]], , method = "Xu") )
plotRGB(lsat, r=6,g=5,b=2, scale=1, stretch="lin")
plot(xu, legend=FALSE, col=rev(terrain.colors(100, alpha=0.35)), add=TRUE )
```

## End(Not run)

---

cgls_urls

Provide URL's for Copernicus Global Land Service datasets

Description

Returns URL's of a product/version/resolution

Usage

```r
cgls_urls(
  dates = NULL,
  resolution = c(1000, 300),
  product = c("fapar", "fcover", "lai", "ndvi"),
  ver = c("newest", "v1", "v2", "v3")
)
```
**cgls_urls**

**Arguments**

- **dates**: Dates to subset default is NULL, returns all products
- **resolution**: The product resolution c("1km", "300m")
- **product**: Which product to query options are "fapar"
- **ver**: Product version options are "newest", "v1", "v2", "v3"

**Details**

Provides a query of the ESA's Copernicus Global Land Service global The query is performed on the manifest files and return URL's however, to download data you will need login credentials which, can be acquired from: http://land.copernicus.eu

If provided, dates need to be in a "YYYY-MM-DD" format. The dates are an explicit search string and can contain dates that are not in the imagery. As such, the user should generate a daily date string representing the range of the desired download as not to have to guess the available dates. Also note that multiple processing versions of a given image are retained in the manifest. This means that if you download a previous processing version, it could be an invalid image. It is highly recommended that you do not change the default ver="newest" argument unless there is a specific reason to.

**Available products**

- **fapar** Fraction of photosynthetically active radiation absorbed by the vegetation
- **fcover** Fraction of green vegetation cover
- **lai** Leaf Area index
- **ndvi** Normalized Difference Vegetation Index

Not yet implemented; Soil Water Index, Surface Soil Moisture, Copernicus product details: http://land.copernicus.eu/global/products/

**Value**

A vector of download URL's for the products

**Author(s)**

Jeffrey S. Evans <jeffrey_evans@tnc.org>

**Examples**

# Create date string for query
d <- seq(as.Date("2020/05/01"), as.Date("2020-09-01"), by="day")

# Search for 300m (333m) LAI within specified date range
all.urls <- cgls_urls(dates = d, resolution = 300,
product = "lai")

# Search for 1000m LAI within specified date range
all.urls <- cgls_urls(dates = d, resolution = 1000,
product = "lai")

# Return all 300m LAI
```r
lai <- cglsls(resolution = 300, product = "lai")
  head(basename(lai))

## Example of downloading URL's
## You need to define your login credentials to download data
# username = "xxxx"
# password = "xxxx"
#
# for(i in 1:length(all.urls)){
#   if(i > 1){ Sys.sleep(3) }
#   file.url <- paste0("https://", paste(username, password, sep=":") , "@",
#     sub(".*://", "", all.urls[i])
#   download.file(file.url, file.path(getwd(),
#     basename(all.urls[i])), mode = 'wb')
# }
```

---

**chae**  
*Canine-Human Age Equivalent*

**Description**
Calculates canines equivalent human age (for fun)

**Usage**

```r
chae(x)
```

**Arguments**

- `x` numeric vector, dog age

**Value**
numeric vector, equivalent human age

**Author(s)**
Jeffrey S. Evans <jeffrey_evans@tnc.org>

**References**
Wang, T., J. M, A.N. Hogan, S. Fong, K. Licon et al. (2020) quantitative translation of dog-to-human aging by conserved remodeling of epigenetic networks
Examples

dat <- data.frame(DogAge = seq(0,18,0.25),
                  HumanAge=chae(seq(0,18,0.25)))[-1,]

plot(dat$DogAge, dat$HumanAge, "l",
     main="Canine-Human Age Equivalence",
     ylab="Human Age", xlab="Dog Age")
points(12, chae(12), col="red", pch=19, cex=1.5)
points(7, chae(7), col="blue", pch=19, cex=1.5)
points(0.5, chae(0.5), col="black", pch=19, cex=1.5)
legend("bottomright", legend=c("Camas (12-YO)", "Kele (7-YO)", "Aster (0.5-YO)",
                            pch=c(19,19,19), cex=c(1.5,1.5,1.5),
                            col=c("red","blue","black"))

Cross-correlation data from Chen (2015)

Description


Format

A list object with 3 elements:

X per capita GRP(yuan)
Y Level of urbanization percent
M Railway Distance (km) matrix of 29 Chinese regions

Source

https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0126158

References

class.comparison

Class comparison between two nominal rasters

Description

Compares two categorical rasters using Cohen’s Kappa (d) or paired t-test statistic(s)

Usage

class.comparison(
    x,
    y,
    x.idx = 1,
    y.idx = 1,
    d = "AUTO",
    stat = "kappa",
    sub.sample = FALSE,
    type = "hexagon",
    p = 0.1,
    size = NULL
)

Arguments

x                      First raster for comparison, SpatialPixelsDataFrame or SpatialGridDataFrame object
y                      Second raster for comparison, SpatialPixelsDataFrame or SpatialGridDataFrame object
x.idx                   Index for the column in the x raster object
y.idx                   Index for the column in the y raster object
d                      Distance for finding neighbors, the default "AUTO" will derive a distance
stat                    Statistic to use in comparison ("kappa", "t.test", "both")
sub.sample              Should a subsampling approach be employed (FALSE/TRUE)
type                    If sub.sample = TRUE, what type of sample ("random" or "hexagon")
p                       If sub.sample = TRUE, what proportion of population should be sampled
size                    If sub.sample = TRUE, alternate to proportion of population (p), using fixed sample size

Value

A SpatialPixelsDataFrame or SpatialPointsDataFrame with the following attributes:

- x x variable used to derive Kappa (d)
- y y variable used to derive Kappa (d)
class.comparison

- kappa Kappa (d) statistic
- t.test Paired t.test statistic (if stat = "t.test" or "both")
- p.value p-value of the paired t.test statistic (if stat = "t.test" or "both")

Note
This function provides a Cohen’s Kappa or paired t-test to compare two classified maps. Point based subsampling is provided for computation tractability. The hexagon sampling is recommended as it is good at capturing spatial process that includes nonstationarity and anisotropy.

Author(s)
Jeffrey S. Evans <jeffrey.evans@tnc.org>

References

Examples

```r
library(sp)
library(raster)

data(meuse.grid)

r1 <- sp::SpatialPixelsDataFrame(points = meuse.grid[c("x", "y")],
data = meuse.grid)

r1@data$class1 <- round(runif(nrow(r1), 1,5),0)

r2 <- sp::SpatialPixelsDataFrame(points = meuse.grid[c("x", "y")],
data = meuse.grid)

r2@data$class2 <- round(runif(nrow(r2), 1,5),0)

d <- class.comparison(r1, r2, x.idx = 8, y.idx = 8, stat="both")

opar <- par(no.readonly=TRUE)

par(mfrow=2)

plot(raster(d, layer=3), main="Kappa")

plot(raster(d, layer=4), main="t.test")

plot(raster(d, layer=5), main="t.test p-value")

par(opar)

# Hexagonal sampling

d.hex <- class.comparison(r1, r2, x.idx = 8, y.idx = 8, stat = "both",
sub.sample = TRUE, d = 500, size = 1000)

sp::bubble(d.hex, "kappa")
d.hex <- sp::na.omit(d.hex, col.name = "t.test")
sp::bubble(d.hex, "t.test")
```
classBreaks

Description

Finds class breaks in a distribution

Usage

classBreaks(x, n, type = c("equal", "quantile", "std", "geometric"))

Arguments

x
A vector to find breaks for

n
Number of breaks

type
Statistic used to find breaks c("equal", "quantile", "std", "geometric")

Value

A vector containing class break values the length is n+1 to allow for specification of ranges

Note

The robust std method uses sqrt(sum(x^2)/(n-1)) to center the data before deriving "pretty" breaks.

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

y <- rnbinom(100, 10, 0.5)
classBreaks(y, 10)
classBreaks(y, 10, type="quantile")

par <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
d <- density(y)
plot(d, type="n", main="Equal Area breaks")
polygon(d, col="cyan")
abline(v=classBreaks(y, 10))
plot(d, type="n", main="Quantile breaks")
polygon(d, col="cyan")
abline(v=classBreaks(y, 10, type="quantile"))
plot(d, type="n", main="Robust Standard Deviation breaks")
polygon(d, col="cyan")
abline(v=classBreaks(y, 10, type="std"))
plot(d, type="n", main="Geometric interval breaks")
polygon(d, col="cyan")
collinear

```r
abline(v=classBreaks(y, 10, type="geometric"))
par(opar)
(y.breaks <- classBreaks(y, 10))
cut(y, y.breaks, include.lowest = TRUE, labels = 1:10)
```

collinear  

**Collinearity test**

**Description**

Test for linear or nonlinear collinearity/correlation in data

**Usage**

```r
collinear(x, p = 0.85, nonlinear = FALSE, p.value = 0.001)
collinear(x, p = 0.85, nonlinear = FALSE, p.value = 0.001)
```

**Arguments**

- **x**: A data.frame or matrix containing continuous data
- **p**: The correlation cutoff (default is 0.85)
- **nonlinear**: A boolean flag for calculating nonlinear correlations (FALSE/TRUE)
- **p.value**: If nonlinear is TRUE, the p value to accept as the significance of the correlation

**Details**

Evaluation of the pairwise linear correlated variables to remove is accomplished through calculating the mean correlations of each variable and selecting the variable with higher mean.

Evaluation of the pairwise linear correlated variables to remove is accomplished through calculating the mean correlations of each variable and selecting the variable with higher mean. If nonlinear = TRUE, pairwise nonlinear correlations are evaluated by fitting y as a semi-parametrically estimated function of x using a generalized additive model and testing whether or not that functional estimate is constant, which would indicate no relationship between y and x thus, avoiding potentially arbitrary decisions regarding the order in a polynomial regression.

**Value**

- Messages and a vector of correlated variables
- Messages and a vector of correlated variables
Author(s)
Jeffrey S. Evans <jeffrey_evans<at>tnc.org>
Jeffrey S. Evans <jeffrey_evans<at>tnc.org>

Examples

data(cor.data)

# Evaluate linear correlations on linear data
head( dat <- cor.data[[4]] )
pairs(dat, pch=20)
  ( cor.vars <- collinear( dat ) )

# Remove identified variable(s)
head( dat[,,-which(names(dat) %in% cor.vars)] )

# Evaluate linear correlations on nonlinear data
# using nonlinear correlation function
plot(cor.data[[1]], pch=20)
collinear(cor.data[[1]], p=0.80, nonlinear = TRUE )

data(cor.data)

# Evaluate linear correlations on linear data
head( dat <- cor.data[[4]] )
pairs(dat, pch=20)
  ( cor.vars <- collinear( dat ) )

# Remove identified variable(s)
head( dat[,,-which(names(dat) %in% cor.vars)] )

# Evaluate linear correlations on nonlinear data
# using nonlinear correlation function
plot(cor.data[[1]], pch=20)
collinear(cor.data[[1]], p=0.80, nonlinear = TRUE )

---

**combine**

**raster combine**

**Description**

Combines rasters into all unique combinations of inputs

**Usage**

combine(x, rnames = NULL, sp = FALSE)
**combine**

**Arguments**

- `x`: raster stack/brick or SpatialPixelsDataFrame object
- `rnames`: Column names to combine in raster stack or sp object
- `sp`: (FALSE/TRUE) output SpatialPixelsDataFrame

**Details**

Please note that this is not a memory safe function that utilizes rasters out of memory in the manner that the raster package does.

If `sp = TRUE` the object will be a list with "combine", containing the SpatialPixelsDataFrame with the value attribute containing the unique combinations, and "summary" with the summary table of collapsed combinations and associated attributes.

If `sp = FALSE` the a single ratified rasterLayer class object is returned with the summary table as the raster attribute table, this is most similar to the ESRI format resulting from their combine function.

**Value**

A ratified rasterLayer or a list containing a SpatialPixelsDataFrame and a data.frame of unique combinations.

**Author(s)**

Jeffrey S. Evans <jeffrey_evans@tnc.org>

**Examples**

```r
library(raster)

r1 <- raster(nrows=100, ncol=100)
r1[] <- round(runif(ncell(r1), 1,4),0)
r2 <- raster(nrows=100, ncol=100)
r2[] <- round(runif(ncell(r2), 2,6),0)
r3 <- raster(nrows=100, ncol=100)
r3[] <- round(runif(ncell(r3), 2,6),0)
r <- stack(r1,r2,r3)
names(r) <- c("LC1","LC2","LC3")

# Combine rasters in stack
( cr <- combine(r) )
levels(cr)

# Combine rasters in stack, using specific rasters
( cr <- combine(r, rnames=c("LC1","LC3")) )

# Combine rasters in stack, output SpatialPixelsDataFrame
cr.sp <- combine(r, sp = TRUE)
  head(cr.sp$summary)
  class(cr.sp$combine)

# Input SpatialPixelsDataFrame
```
Concordance test for binomial models

Description

Performs a concordance/disconcordance (C-statistic) test on binomial models.

Usage

concordance(y, p)

Arguments

y: vector of binomial response variable used in model
p: estimated probabilities from fit binomial model

Value

list object with: concordance, discordance, tied and pairs

Note

Test of binomial regression for the hypothesis that probabilities of all positives [1], are greater than the probabilities of the nulls [0]. The concordance would be 100 inverse of concordance, representing the null. The C-statistic has been show to be comparable to the area under an ROC

Results are: concordance - percent of positives that are greater than probabilities of nulls, discordance - concordance inverse of concordance representing the null class, tied - number of tied probabilities and pairs - number of pairs compared

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References

Examples

```r
data(mtcars)
dat <- subset(mtcars, select=c(mpg, am, vs))
glm.reg <- glm(vs ~ mpg, data = dat, family = binomial)
concordance(dat$vs, predict(glm.reg, type = "response"))
```

---

**conf.interval**  
*Confidence interval for mean or median*

**Description**

Calculates confidence interval for the mean or median of a distribution with unknown population variance

**Usage**

```r
conf.interval(x, cl = 0.95, stat = "mean", std.error = TRUE)
```

**Arguments**

- `x`: Vector to calculate confidence interval for
- `cl`: Percent confidence level (default = 0.95)
- `stat`: Statistic (mean or median)
- `std.error`: Return standard error (TRUE/FALSE)

**Value**

- `lci`: Lower confidence interval value
- `uci`: Upper confidence interval value
- `mean`: If `stat = "mean"`, mean value of distribution
- `mean`: Value of the mean or median
- `conf.level`: Confidence level used for confidence interval
- `std.error`: If `std.error = TRUE` standard error of distribution

**Author(s)**

Jeffrey S. Evans <jeffrey_evans@tnc.org>
Examples

```r
x <- runif(100)
cr <- conf.interval(x, cl = 0.97)
print(cr)

d <- density(x)
plot(d, type="n", main = "PDF with mean and 0.97 confidence interval")
polygon(d, col="cyan3")
abline(v=mean(x, na.rm = TRUE), lty = 2)
segments( x0=cr["lci"], y0=mean(d$y), x1=cr["uci"],
         y1=mean(d$y), lwd = 2.5,
         col = "black")
legend("topright", legend = c("mean", "CI"),
       lty = c(2,1), lwd = c(1,2.5))
```

cor.data

Various correlation structures

Description

linear and nonlinear correlated data examples

A list object with various linear and nonlinear correlation structures

Format

A list object with 4 elements containing data.frames:

- **example 1** two columns with nonlinear wave function relationship
- **example 2** two columns with simple nonlinear relationship
- **example 3** two columns with nonlinear multi-level wave function relationship
- **example 4** 4 columns with first two having linear relationship

---

correlogram

Correlogram

Description

Calculates and plots a correlogram

Usage

```r
correlogram(x, v, dist = 5000, dmatrix = FALSE, ns = 99, latlong = FALSE, ...)
```
Arguments

- **x**  
  SpatialPointsDataFrame object
- **v**  
  Test variable in x@data
- **dist**  
  Distance of correlation lags, if latlong=TRUE units are in kilometers
- **dmatrix**  
  Should the distance matrix be include in output (TRUE/FALSE)
- **ns**  
  Number of simulations to derive simulation envelope
- **latlong**  
  Coordinates are in latlong (TRUE/FALSE)
- **...**  
  Arguments passed to cor (‘pearson’, ‘kendall’ or ‘spearman’)

Value

A list object containing:

- autocorrelation is a data.frame object with the following components
- autocorrelation - Autocorrelation value for each distance lag
- dist - Value of distance lag
- lci - Lower confidence interval (p=0.025)
- uci - Upper confidence interval (p=0.975)
- CorrPlot recordedplot object to recall plot
- dmatrix Distance matrix (if dmatrix=TRUE)

Author(s)

Jeffrey S. Evans jeffrey_evans@tnc.org

Examples

```r
library(sp)
data(meuse)
coordinates(meuse) = ~x+y
zinc.cg <- correlogram(x = meuse, v = meuse@data[, 'zinc'], dist = 250, ns = 9)
```

---

cross.tab  
Class comparison between two nominal rasters

Description

Creates a labeled cross tabulation between two nominal rasters

Usage

cross.tab(x, y, values = NULL, labs = NULL, pct = FALSE, ...)

Arguments

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>rasterLayer class object</td>
</tr>
<tr>
<td>y</td>
<td>rasterLayer class object to compare to x</td>
</tr>
<tr>
<td>values</td>
<td>Expected values in both rasters</td>
</tr>
<tr>
<td>labs</td>
<td>Labels associated with values argument</td>
</tr>
<tr>
<td>pct</td>
<td>(TRUE/FALSE) return proportions rather than counts</td>
</tr>
<tr>
<td>...</td>
<td>Additional arguments</td>
</tr>
</tbody>
</table>

Value

a table with the cross tabulated counts

Note

This function returns a cross tabulation between two nominal rasters. Arguments allow for labeling the results and returning proportions rather than counts. It also accounts for asymmetrical classes between the two rasters

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


See Also

raster::crosstab

Examples

```r
library(sp)
library(raster)
data(meuse.grid)

r1 <- sp::SpatialPixelsDataFrame(points = meuse.grid[c("x", "y")],
data = meuse.grid)

lulc2010 <- raster(r1)
na.idx <- which(!is.na(lulc2010[]))
  lulc2010[na.idx] <- sample(1:5, length(na.idx), replace=TRUE)

lulc2020 <- raster(lulc2010)
lulc2020[na.idx] <- sample(1:5, length(na.idx), replace=TRUE)

( v = sort(unique(c(lulc2010[], lulc2020[])) )
  l = c("water", "urban", "forest",
        "ag", "barren")
```
cross.tab(lulc2010, lulc2020)
cross.tab(lulc2010, lulc2020, values = v, labs = l)
cross.tab(lulc2010, lulc2020, values = v, labs = l, pct=TRUE)

# Create asymmetrical classes
lulc2020[na.idx] <- sample(c(1,2,4,5), length(na.idx), replace=TRUE)
cross.tab(lulc2010, lulc2020, values = v, labs = l, pct=TRUE)

crossCorrelation  Spatial cross correlation

Description
Calculates univariate or bivariate spatial cross-correlation using local Moran’s-I (LISA), following Chen (2015)

Usage
crossCorrelation(
  x,
  y = NULL,
  coords = NULL,
  w = NULL,
  type = c("LSCI", "GSCI"),
  k = 999,
  dist.function = c("inv.power", "neg.exponent", "none"),
  scale.xy = TRUE,
  scale.partial = FALSE,
  scale.matrix = FALSE,
  alpha = 0.05,
  clust = TRUE,
  return.sims = FALSE
)

Arguments
x  Vector of x response variables
y  Vector of y response variables, if not specified the univariate statistic is returned
coords  A matrix of coordinates corresponding to (x,y), only used if w = NULL. Can also be an sp object with relevant x,y coordinate slot (ie., points or polygons)
w  Spatial neighbors/weights in matrix format. Dimensions must match (n(x),n(y)) and be symmetrical. If w is not defined then a default method is used.
type  c("LSCI","GSCI") Return Local Spatial Cross-correlation Index (LSCI) or Global Spatial cross-correlation Index (GSCI)
crossCorrelation

k Number of simulations for calculating permutation distribution under the null hypothesis of no spatial autocorrelation
dist.function ("inv.power", "neg.exponent", "none") If w = NULL, the default method for deriving spatial weights matrix, options are: inverse power or negative exponent, none is for use with a provided matrix
scale.xy (TRUE/FALSE) scale the x,y vectors, if FALSE it is assumed that they are already scaled following Chen (2015)
scale.partial (FALSE/TRUE) rescale partial spatial autocorrelation statistics
scale.matrix (FALSE/TRUE) If a neighbor/distance matrix is passed, should it be scaled using (w/sum(w))
alpha = 0.05 confidence interval (default is 95 pct)
clust (FALSE/TRUE) Return approximated lisa clusters
return.sims (FALSE/TRUE) Return randomizations vector n = k

Details

In specifying a distance matrix, you can pass a coordinates matrix or spatial object to coords or alternately, pass a distance or spatial weights matrix to the w argument. If the w matrix represents spatial weights dist.function="none" should be specified. Otherwise, w is assumed to represent distance and will be converted to spatial weights using inv.power or neg.exponent. The w distances can represent an alternate distance hypothesis (eg., road, stream, network distance) Here are example argument usages for defining a matrix.

• IF coords=x, w=NULL, dist.function= c("inv.power", "neg.exponent") A distance matrix is derived using the data passed to coords then spatial weights derived using one of the dist.function options
• IF cords=NULL, w=x, dist.function= c("inv.power", "neg.exponent") It is expected that the distance matrix specified with w represent some form of distance then the spatial weights are derived using one of the dist.function options
• IF cords=NULL, w=x, dist.function="none" It is assumed that the matrix passed to w already represents the spatial weights

Value

When not simulated k=0, a list containing:

• I Global autocorrelation statistic
• SCI A data.frame with two columns representing the xy and yx autocorrelation
• nsim value of NULL to represent p values were derived from observed data (k=0)
• p Probability based observations above/below confidence interval
• t.test Probability based on t-test
• clusters If "clust" argument TRUE, vector representing LISA clusters

when simulated (k>0), a list containing:

• I Global autocorrelation statistic
crossCorrelation

• SCI A data.frame with two columns representing the xy and yx autocorrelation
• nsim value representing number of simulations
• global.p p-value of global autocorrelation statistic
• local.p Probability based simulated data using successful rejection of t-test
• range.p Probability based on range of probabilities resulting from paired t-test
• clusters If "clust" argument TRUE. vector representing lisa clusters

References


Examples

# replicate Chen (2015)
data(chen)
( r <- crossCorrelation(x=chen["X"], y=chen["Y"], w = chen["M"],
    clust=TRUE, type = "LSCI", k=0,
    dist.function = "inv.power") )

library(sp)
library(spdep)

data(meuse)
coordinates(meuse) <- ~x+y

#### Using a default spatial weights matrix method (inverse power function)
( I <- crossCorrelation(meuse$zinc, meuse$copper,
    coords = coordinates(meuse), k=99) )
meuse$lisa <- I$SCI[,"lsci.xy"]
spplot(meuse, "lisa")

#### Providing a distance matrix
Wij <- spDists(meuse)
( I <- crossCorrelation(meuse$zinc, meuse$copper, w = Wij, k=99) )

#### Providing an inverse power function weights matrix
Wij <- spDists(meuse)
    Wij <- 1 / Wij
diag(Wij) <- 0
    Wij <- Wij / sum(Wij)
diag(Wij) <- 0
( I <- crossCorrelation(meuse$zinc, meuse$copper, w = Wij,
    dist.function = "none", k=99) )
Cosine Similarity Index

Description

Calculates the cosine similarity and angular similarity on two vectors or a matrix

Usage

csi(x, y = NULL)

Arguments

x A vector or matrix object
y If x is a vector, then a vector object

Value

If x is a matrix, a list object with: similarity and angular.similarity matrices or, if x and y are vectors, a vector of similarity and angular.similarity

Note

The cosine similarity index is a measure of similarity between two vectors of an inner product space. This index is bested suited for high-dimensional positive variable space. One useful application of the index is to measure separability of clusters derived from algorithmic approaches (e.g., k-means). It is a good common practice to center the data before calculating the index. It should be noted that the cosine similarity index is mathematically, and often numerically, equivalent to the Pearson’s correlation coefficient

The cosine similarity index is derived: s(xy) = x * y / ||x|| * ||y||, where the expected is 1.0 (perfect similarity) to -1.0 (perfect dissimilarity). A normalized angle between the vectors can be used as a bounded similarity function within [0,1] angular similarity = 1 - (cos(s)^-1/pi)

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

# Compare two vectors (centered using scale)
x=runif(100)
y=runif(100)^2
csi(as.vector(scale(x)),as.vector(scale(y)))

# Compare columns (vectors) in a matrix (centered using scale)
```r
x <- matrix(round(runif(100),0),nrow=20,ncol=5)
( s <- csi(scale(x)) )

# Compare vector (x) to each column in a matrix (y)
y <- matrix(round(runif(500),3),nrow=100,ncol=5)
x=runif(100)
csi(as.vector(scale(x)),scale(y))
```

---

### curvature

#### Surface curvature

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculates Zevenbergen &amp; Thorne, McNab’s or Bolstad’s curvature</td>
</tr>
</tbody>
</table>

#### Usage

```r
curvature(x, type = c("planform", "profile", "total", "mcnab", "bolstad"), ...)
```

#### Arguments

- **x**
  - rasterLayer object
- **type**
  - Method used c("planform", "profile", "total", "mcnab", "bolstad")
- **...**
  - Additional arguments passed to writeRaster

#### Value

raster class object of surface curvature

#### Note

The planform and profile curvatures are the second derivative(s) of the elevation surface, or the slope of the slope. Profile curvature is in the direction of the maximum slope, and the planform curvature is perpendicular to the direction of the maximum slope. Negative values in the profile curvature indicate the surface is upwardly convex whereas, positive values indicate that the surface is upwardly concave. Positive values in the planform curvature indicate that the surface is laterally convex whereas, negative values indicate that the surface is laterally concave.

Total curvature is the sigma of the profile and planform curvatures. A value of 0 in profile, planform or total curvature, indicates the surface is flat. The planform, profile and total curvatures are derived using Zevenbergen & Thorne (1987) via a quadratic equation fit to eight neighbors as such, the s (focal window size) argument is ignored.

McNab’s and Bolstad’s variants of the surface curvature (concavity/convexity) index (McNab 1993; Bolstad & Lillesand 1992; McNab 1989). The index is based on features that confine the view from the center of a 3x3 window. In the Bolstad equation, edge correction is addressed by dividing by the radius distance to the outermost cell (36.2m).
diurnal anisotropic heat index

**Description**

Simple approximation of the anisotropic diurnal heat (Ha) distribution

The Diurnal Anisotropic Heat Index is based on this equation. \( Ha = \cos(\text{amax} - a) \times \arctan(b) \)

Where; amax defines the aspect with the maximum total heat surplus, \( a \) is the aspect and \( b \) is the slope angle.
Usage

dahi(x, amax = 202.5)

Arguments

x          An elevation raster of class RasterLayer, SpatRaster or SpatialPixelsDataFrame
amax       The Alpha Max (amax) parameter in degrees defined as: minimum = 0, maximum = 360 with the default = 202.500

Value

RasterLayer class object Diurnal Anisotropic Heat Index

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


Examples

library(raster)
data(elev)
Ha <- dahi(elev)
plot(Ha)

date_seq

date sequence

Description

creates date sequence given start and stop dates

Usage

date_seq(
  start,
  end,
  step = c("day", "week", "month", "quarter", "year", "minute"),
  rm.leap = FALSE
)

daymet.point

Arguments

- **start**: Start date in "yyyy/mm/dd" character format
- **end**: End date in "yyyy/mm/dd" character format
- **step**: Time step, options are c("day", "week", "month", "quarter", "year", "minute")
- **rm.leap**: Remove extra days in leap years

Value

A date vector of class POSIXct for minute and Date for other options

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

```r
# monthly steps 1990/01/01 - 2019/12/31
d <- date_seq("1990/01/01", "2019/12/31", step="month")

# daily steps 1990/01/01 - 2019/12/31
d <- date_seq("1990/01/01", "2019/12/31", step="day")

# daily steps 1990/01/01 - 2019/12/31 with leap days removed
d <- date_seq("1990/01/01", "2019/12/31", step="day", rm.leap=TRUE)

# daily step 2008/12/29 - 2008/12/31, 2008 is leap year
d <- date_seq("2008/12/29", "2008/12/31")

# minutes step 2008/12/29 - 2008/12/31, 2008 is leap year
d <- date_seq("2008/12/29", "2008/12/31", step="minute")
```

daymet.point

**DAYMET point values**

Description

Downloads DAYMET climate variables for specified point and time-period

Usage

```r
daymet.point(
  lat,
  long,
  start.year,
  end.year,
  site = NULL,
)```
daymet.tiles

    files = FALSE,
    echo = FALSE
  )

Arguments

lat       latitude of point (decimal degrees WGS84)
long      longitude pf point (decimal degrees WGS84)
start.year First year of data
end.year   Last year of data
site      Unique identification value that is appended to data
files     (TRUE/FALSE) Write file to disk
echo      (TRUE/FALSE) Echo progress

Value

A data.frame with climate results

Note

data is available for Long -131.0 W and -53.0 W; lat 52.0 N and 14.5 N Function uses the Single
Pixel Extraction tool and returns year, yday, dayl(s), prcp (mm/day), srad (W/m^2), swe (kg/m^2),
tmax (deg c), tmin (deg c), vp (Pa) Metadata for DAYMET single pixel extraction: https://
daymet.ornl.gov/files/UserGuides/current/readme_singlepointextraction.pdf

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

  ( d <- daymet.point(lat = 36.0133, long = -84.2625, start.year = 2013,
                      end.year=2014, site = "1", files = FALSE, echo = FALSE) )

________________________________________________________________________________________

daymet.tiles  

    \textit{DAYMET Tile ID's}

________________________________________________________________________________________

Description

Returns a vector of DAYMET tile id’s within a specified extent

Usage

daymet.tiles(...)
Arguments

... ignored

Value

Vector of DAYMET tile IDS or if sp = TRUE a sp class SpatialPolygonsDataFrame

Note

Function accepts sp, raster or extent class object or bounding coordinates. All input must be in the same projection as the tile index SpatialPolygonsDataFrame. The library includes the DAYMAT tile index "DAYMET_tiles" which can be add using data(), see examples.

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

<table>
<thead>
<tr>
<th>DAYMET_tiles</th>
<th>DAYMET climate tile index</th>
</tr>
</thead>
</table>

Description

Polygon tile index for DAYMET climate data

Format

An sp SpatialPolygonsDataFrame with 404 features (rows) and 6 columns (columns):

- **Id**  Tile Index Identification
- **Area** Area of each tile
- **XMin** Minimum x geographic decimal degree coordinate
- **XMax** Maximum x geographic decimal degree coordinate
- **YMin** Minimum y geographic decimal degree coordinate
- **YMax** Maximum y geographic decimal degree coordinate

Source

https://daymet.ornl.gov/
dispersion

Dispersion (H-prime)

Description
Calculates the dispersion ("rarity") of targets associated with planning units.

Usage
dispersion(x)

Arguments
x data.frame object of target values

Value
data.frame with columns H values for each target, H, sH, sHmax

Note
The dispersion index (H-prime) is calculated $H = \sum \frac{\sqrt{p}}{\sqrt{a}}$ where; $P = \frac{\text{sum of target in planning unit}}{\text{sum of target across all planning units}}$ and $a = \frac{\text{count of planning units containing target}}{\text{number of planning units}}$

Author(s)
Jeffrey S. Evans <jeffrey_evans@tnc.org>

References

Examples

```r
library(sp)
data(pu)

d <- dispersion(pu@data[,2:ncol(pu)])
p <- d[,"H"]
clr <- c("#3288BD", "#99D594", "#E6F598", "#FEE08B", "#FC8D59", "#D53E4F")
clrs <- ifelse(p < 0.5524462, clr[1],
ifelse(p >= 0.5524462 & p < 1.223523, clr[2],
ifelse(p >= 1.223523 & p < 2.465613, clr[3],
```
ifelse(p >= 2.465613 & p < 4.76429, clr[4],
ifelse(p >= 4.76429 & p < 8.817699, clr[5],
ifelse(p >= 8.817699, clr[6], NA)))))
plot(pu, col=clrs, border=NA)
legend("topleft", legend=rev(c("Very Rare","Rare","Moderately Rare",
"Somewhat Common","Common","Over Dispersed")),
fill=clr, cex=0.6, bty="n")
box()

---

description

**Description**

Calculates the Evans (1972) Martonne's modified dissection

**Usage**

dissection(x, s = 5, ...)

**Arguments**

- `x` raster object
- `s` Focal window size
- `...` Additional arguments passed to raster::calc

**Value**

raster class object of Martonne’s modified dissection

**Note**

Dissection is calculated as: \((z(s) - \text{min}(z(s))) / (\text{max}(z(s)) - \text{min}(z(s)))\)

**Author(s)**

Jeffrey S. Evans <jeffrey_evans@tnc.org>

**Examples**

```r
library(raster)
data(elev)
d <- dissection(elev, s=3)
plot(d, main="dissection")
```
Description

Kullback-Leibler Divergence (Cross-entropy)

Usage

divergence(x, y, type = c("Kullback-Leibler", "cross-entropy"))

Arguments

- x: a vector of integer values, defining observed
- y: a vector of integer values, defining estimates
- type: Type of divergence statistic c("Kullback-Leibler", "cross-entropy")

Value

single value vector with divergence statistic

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

```r
x <- round(runif(10,1,4),0)
y <- round(runif(10,1,4),0)
divergence(x, y)
divergence(x, y, type = "cross-entropy")
```

Description

Batch download of daily gridded DAYMET climate data

Usage

download.daymet(...)
Arguments

... ignored

Details

DAYMET website: http://daymet.ornl.gov, path structure: /year/tile_year/file.nc

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


Details

Available products: treecover2000, loss, gain, lossyear, datamask, first, or last

- **treecover2000** - (Tree canopy cover for year 2000) - Tree cover in the year 2000, defined as canopy closure for all vegetation taller than 5m in height. Encoded as a percentage per output grid cell, in the range 0-100.

- **loss** - (Global forest cover loss 2000-2013) - Forest loss during the period 2000-2013, defined as a stand-replacement disturbance, or a change from a forest to non-forest state. Encoded as either 1 (loss) or 0 (no loss).

- **gain** - (Global forest cover gain 2000-2012) - Forest gain during the period 2000-2012, defined as the inverse of loss, or a non-forest to forest change entirely within the study period. Encoded as either 1 (gain) or 0 (no gain).

- **lossyear** - (Year of gross forest cover loss event) - A disaggregation of total forest loss to annual time scales. Encoded as either 0 (no loss) or else a value in the range 1-13, representing loss detected primarily in the year 2001-2013.

- **datamask** - (Data mask) - Three values representing areas of no data (0), mapped land surface (1), and permanent water bodies (2).

- **first** - (Circa year 2000 Landsat 7 cloud-free image composite) - Reference multispectral imagery from the first available year, typically 2000. If no cloud-free observations were available for year 2000, imagery was taken from the closest year with cloud-free data, within the range 1999-2012.

- **last** - (Circa year 2013 Landsat cloud-free image composite) - Reference multispectral imagery from the last available year, typically 2013. If no cloud-free observations were available for year 2013, imagery was taken from the closest year with cloud-free data, within the range 2010-2012.


Value

Downloaded Hansen forest loss tif files

Author(s)

Jeffrey S. Evans jeffrey_evans@tnc.org

References

Examples

```r
## Not run:
# Download single tile
download.hansen(tile=c('00N', '130E'), data.type=c('loss', 'lossyear'),
                download.folder=getwd())

# Batch download of multiple tiles
tiles <- list(c('00N', '140E'), c('00N', '130E'))
for( j in 1:length(tiles)){
  download.hansen(tile=tiles[[j]], data.type=c('loss'))
}

## End(Not run)
```

download.prism Download PRISM

Description

Batch download of monthly gridded PRISM climate data

Usage

download.prism(...)

Arguments

... Nonexistent parameters

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

effect.size Cohen's-d effect size

description

Description

Cohen's-d effect size with pooled sd for a control and experimental group

Usage

effect.size(y, x, pooled = TRUE, conf.level = 0.95)
Arguments

- `y`: A character or factor vector
- `x`: A numeric vector, same length as `y`
- `pooled`: Pooled or population standard deviation (TRUE/FALSE)
- `conf.level`: Specified confidence interval. Default is 0.95

Value

An effect.size class object with `x`, `y` and a data.frame with columns for effect size, lower confidence interval, lower confidence interval. The row names of the data frame represent the levels in `y`

Note

This implementation will iterate through each class in `y` and treating a given class as the experimental group and all other classes as a control case. Each class had d and the confidence interval derived. A negative d indicate directionality with same magnitude. The expected range for d is 0 - 3 d is derived; ( mean(experimental group) - mean(control group) ) / sigma(p) pooled standard deviation is derived; sqrt( ( (Ne - 1) * sigma(e)^2 + (Nc - 1) * sigma(c)^2 ) / (Ne + Nc - 2) ) where; Ne, Nc = n of experimental and control groups.

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


Examples

```r
(es <- effect.size(iris$Species, iris$Sepal.Length))
plot(es)
```

elev

Elevation raster

description

Elevation raster of Switzerland
Format

A raster RasterLayer class object:

- **resolution**: 5 arc-minute 0.00833 (10000m)
- **nrow**: 264
- **ncol**: 564
- **ncell**: 148896
- **xmin**: 5.9
- **xmax**: 10.6
- **ymin**: 45.7
- **ymax**: 47.9
- **proj4string**: +proj=longlat +ellps=WGS84

Source

http://www.diva-gis.org/Data

---

**erase.point**  
**Erase points**

Description

Removes points intersecting a polygon feature class

Usage

erase.point(y, x, inside = TRUE)

Arguments

- **y**: A SpatialPoints or SpatialPointsDataFrame
- **x**: A SpatialPolygons or SpatialPolygonsDataFrame
- **inside**: (TRUE/FALSE) Remove points inside polygon, else outside polygon

Value

A SpatialPoints or SpatialPointsDataFrame

Note

Used to erase points that intersect polygon(s). If inside=FALSE then the function results in an intersection operation where points that intersect the polygon are retained. This function effectively duplicates the ESRI ArcGIS Erase Point tool.
**explode**

Explodes multipart features

**Description**

Explodes multipart features into single part

**Usage**

`explode(x, sp = FALSE)`

**Arguments**

- `x` : sp or sf multipart (MULTIPOLYGON, MULTIPOLYGON, MULTILINE) object
- `sp` : (FALSE/TRUE) output as sp class object, else is sf class

**Value**

A single part sp or sf object (polygons or points)
Note

Multipart geometries are a data structure where a single attribute shares multiple features (polygons, points, lines). This function disaggregates the data into a one-to-one match.

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

```r
library(sf)
library(sp)

dim( p.sf <- st_read(system.file("shapes/sids.shp", package = "spData")[[1]]) )
dim( p.sf <- explode(p.sf) )
```

Description

Extracts \([x,y]\) vertices from an sp line or polygon object

Usage

```r
extract.vertices(x, as.sp = FALSE, rm.duplicates = FALSE, join = FALSE)
```

Arguments

- **x**: An sp class `SpatialPolygonsDataFrame`, `SpatialPolygons`, `SpatialLinesDataFrame` or `SpatialLines` object
- **as.sp**: (FALSE/TRUE) Output as sp `SpatialPointsDataFrame`
- **rm.duplicates**: (FALSE/TRUE) remove duplicate \((x,y)\) coordinates
- **join**: (FALSE/TRUE) Joint attributes from original object

Value

A `SpatialPointsDataFrame` or data.frame with id, x, y and merged attributes

Note

This function returns the vertices of a line or polygon object, as opposed to the polygon centroids or line start/stop coordinates available in the `@coords` slot. This requires accessing the coordinates located in the `x@polygons@Polygons` or `x@lines@Lines` slots.
focal.lmetrics

Author(s)

Jeffrey S. Evans <jeffrey.evans@tnc.org>

Examples

```r
library(sp)
library(raster)
library(sf)

# For polygons
r <- raster(xmn=-11.69, xmx=2988.31, ymn=-749.97, ymx=1650.03,
resolution=c(100,100))
r[] <- runif(ncell(r))
names(r) <- "random_process"

polys <- as(r, "SpatialPolygonsDataFrame")
polys <- polys[sample(1:nrow(polys),10),]
extract.vertices(polys, join=TRUE, rm.duplicates=TRUE)

v <- extract.vertices(polys, as.sp=TRUE, join=TRUE)
head(v@data)
plot(polys)
points(v, pch=20, cex=2, col="red")

# For lines
nc <- sf::st_read(system.file("shape/nc.shp", package="sf"))
nc <- sf::st_cast(sf::st_cast(nc, "POLYGON"), "LINESTRING")
nc <- as(nc, "Spatial")
extract.vertices(nc)
extract.vertices(nc, join=TRUE, rm.duplicates=TRUE)

v <- extract.vertices(nc, as.sp=TRUE, join=TRUE)
head(v@data)
plot(nc)
points(v, pch=20, cex=2, col="red")
```

---

### focal.lmetrics

Focal landscape metrics

#### Description

Calculates a variety of landscape metrics on integer rasters using focal approach
Usage

focal.lmetrics(...)

Arguments

... Parameters to be passed to the modern version of the function

Examples

## Not run:
library(landscapemetrics)
library(raster)

data(landscape)

s <- matrix(1, nrow = 3, ncol = 3)
( result <- do.call(stack, window_lsm(landscape, window = s,
             what = c("lsm_l_pr", "lsm_l_joinent"))) )

plot(result)

## End(Not run)

---

fuzzySum  

Fuzzy Sum

Description

Calculates the fuzzy sum of a vector

Usage

fuzzySum(x)

Arguments

x  

Vector of values to apply fuzzy sum

Value

Value of fuzzy sum

Note

The fuzzy sum is an increasing linear combination of values. This can be used to sum probabilities or results of multiple density functions.

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>
gaussian.kernel

Examples

\begin{verbatim}
p = c(0.8, 0.76, 0.87)
fuzzySum(p)
sum(p)
p = c(0.3, 0.2, 0.1)
fuzzySum(p)
sum(p)
\end{verbatim}

---

Description

Creates a Gaussian Kernel of specified size and sigma

Usage

\begin{verbatim}
gaussian.kernel(sigma = 2, n = 5)
\end{verbatim}

Arguments

- **sigma**: sigma (standard deviation) of kernel (defaults 2)
- **n**: size of symmetrical kernel (defaults to 5x5)

Value

Symmetrical (NxN) matrix of a Gaussian distribution

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

\begin{verbatim}
par(mfrow=c(2,2))
persp(gaussian.kernel(sigma=1, n=27), theta = 135,
    phi = 30, col = "grey", ltheta = -120, shade = 0.6,
    border=NA )
persp(gaussian.kernel(sigma=2, n=27), theta = 135, phi = 30,
    col = "grey", ltheta = -120, shade = 0.6, border=NA )
persp(gaussian.kernel(sigma=3, n=27), theta = 135, phi = 30,
    col = "grey", ltheta = -120, shade = 0.6, border=NA )
persp(gaussian.kernel(sigma=4, n=27), theta = 135, phi = 30,
    col = "grey", ltheta = -120, shade = 0.6, border=NA )
\end{verbatim}
Buffers data in geographic (Latitude/Longitude) projection

Arguments

- `x`: A sf or sp vector class object
- `r`: Buffer radius in meters
- `sf`: (FALSE/TRUE) Output sf class object else sp
- `...`: Additional arguments passed to `gBuffer`

Value

An sp or sf polygon class object representing buffer for each feature

Note

Projects (Latitude/Longitude) data in decimal-degree geographic projection using an on-the-fly azimuthal equidistant projection in meters centered on

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

See Also

`gBuffer` for `gBuffer` arguments

Examples

```r
library(sp)
library(raster)

s <- spsample(as(extent(61.87125, 76.64458, 23.90153, 37.27042),
    "SpatialPolygons"), n=100, type="random")
proj4string(s) <- '+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs'

b <- geo.buffer(x=s, r=1000, quadsegs=100)
plot(b[1,])
points(s[1,], pch=20, cex=2)
```
**Description**

Creates a probability density plot of y for each group of x

**Usage**

```r
group.pdf(
  x,
  y,
  col = NULL,
  lty = NULL,
  lwd = NULL,
  lx = "topleft",
  ly = NULL,
  ...
)
```

**Arguments**

- `x` Numeric, character or factorial vector of grouping variable (must be same length as y)
- `y` Numeric vector (density variable)
- `col` Optional line colors (see par, col)
- `lty` Optional line types (see par, lty)
- `lwd` Optional line widths (see par, lwd)
- `lx` Position of legend (x coordinate or 'topright', 'topleft', 'bottomright', 'bottom-left')
- `ly` Position of legend (y coordinate)
- `...` Additional arguments passed to plot

**Author(s)**

Jeffrey S. Evans <jeffrey_evans<at>tnc.org>

**References**

Examples

```r
y <- dnorm(runif(100))
x <- rep(c(1, 2, 3), length.out = length(y))
group.pdf(x = as.factor(x), y = y, main = 'Probability Density of y by group(x)',
ylab = 'PDF', xlab = 'Y', lty = c(1, 2, 3))
```

hexagons

**Hexagons**

Description

Create hexagon polygons

Usage

`hexagons(x, res = 100, ...)`

Arguments

- `x`: sp SpatialDataFrame class object
- `res`: Area of resulting hexagons
- `...`: Additional arguments passed to spsample

Value

SpatialPolygonsDataFrame OBJECT

Note

depends: sp

Examples

```r
require(sp)
data(meuse)
coordinates(meuse) <- ~x+y

hex.polys <- hexagons(meuse, res=100)
plot(hex.polys)
plot(meuse, pch = 20, add = TRUE)

# Points intersecting hexagons
hex.pts <- na.omit(over(meuse, hex.polys))
(hex.pts <- data.frame(PTID = rownames(hex.pts), hex.pts))
```
**hli**

*Heat Load Index*

**Description**
Calculates the McCune & Keon (2002) Heat Load Index

**Usage**

```r
hli(x, check = TRUE, force.hemisphere = c("none", "southern", "northern"))
```

**Arguments**

- **x**: rasterLayer class object
- **check**: (TRUE/FALSE) check for projection integrity and calculate central latitude for non-geographic projections
- **force.hemisphere**: If country is split at the equator, force southern or northern hemisphere equation c("southern", "northern")

**Value**
raster class object of McCune & Keon (2002) Heat Load Index

**Note**
Describes A southwest facing slope should have warmer temperatures than a southeast facing slope, even though the amount of solar radiation they receive is equivalent. The McCune and Keon (2002) method accounts for this by "folding" the aspect so that the highest values are southwest and the lowest values are northeast. Additionally, this method account for steepness of slope, which is not addressed in most other aspect rescaling equations. HLI values range from 0 (coolest) to 1 (hottest).

The equations follow McCune (2007) and support northern and southern hemisphere calculations. The folded aspect for northern hemispheres use (180 - (Aspect – 225) ) and for Southern hemisphere ( 180 - ( Aspect – 315) ). If a country is split at the equator you can use the force.hemisphere argument to choose which equation to use. Valid values for this argument are "southern" and "northern" with the default "none".

**Author(s)**
Jeffrey S. Evans <jeffrey_evans@tnc.org>

**References**

hsp

Hierarchical Slope Position

Examples

```r
library(raster)
data(elev)
heat.load <- hli(elev)
plot(heat.load, main="Heat Load Index")
```

Description

Calculates a hierarchical scale decomposition of topographic position index

Usage

```r
hsp(
  x,
  min.scale = 3,
  max.scale = 27,
  inc = 4,
  win = "rectangle",
  normalize = FALSE
)
```

Arguments

- `x`: Object of class raster (requires integer raster)
- `min.scale`: Minimum scale (window size)
- `max.scale`: Maximum scale (window size)
- `inc`: Increment to increase scales
- `win`: Window type, options are "rectangle" or "circle"
- `normalize`: Normalize results to 0-1 scale (FALSE | TRUE)

Value

raster class object

Note

if win = "circle" units are distance, if win = "rectangle" units are number of cells

Author(s)

Jeffrey S. Evans <jeffrey.evans@tnc.org>
hybrid.kmeans

References


Examples

```r
library(raster)
data(elev)
hsp27 <- hsp(elev, 3, 27, 4, normalize = TRUE)
plot(hsp27)
```

hybrid.kmeans

Hybrid K-means

Description

Hybrid K-means clustering using hierarchical clustering to define cluster-centers

Usage

```r
hybrid.kmeans(x, k = 2, hmethod = "ward.D", stat = mean, ...)
```

Arguments

- `x` A data.frame or matrix with data to be clustered
- `k` Number of clusters
- `hmethod` The agglomeration method used in hclust
- `stat` The statistic to aggregate class centers (mean or median)
- `...` Additional arguments passed to `kmeans`

Details

This method uses hierarchical clustering to define the cluster-centers in the K-means clustering algorithm. This mitigates some of the known convergence issues in K-means.

Value

returns an object of class "kmeans" which has a print and a fitted method

Note

options for hmethod are: "ward.D", "ward.D2", "single", "complete", "average", mcquitty", "median", "centroid"
Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


See Also

kmeans for available ... arguments and function details
hclust for details on hierarchical clustering

Examples

```r
x <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),
           matrix(rnorm(100, mean = 1, sd = 0.3), ncol = 2))

# Compare k-means to hybrid k-means with k=4
km <- kmeans(x, 4)
hkm <- hybrid.kmeans(x, k=4)

opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))
plot(x[,1],x[,2], col=km$cluster,pch=19, main="K-means")
plot(x[,1],x[,2], col=hkm$cluster,pch=19, main="Hybrid K-means")
par(opar)
```

---

**idw.smoothing**

*Inverse Distance Weighted smoothing*

**Description**

Distance weighted smoothing of a variable in a spatial point object

**Usage**

```r
idw.smoothing(x, y, d, k)
```

**Arguments**

- **x**: Object of class SpatialPointsDataFrame
- **y**: Numeric data in x@data
- **d**: Distance constraint
- **k**: Maximum number of k-nearest neighbors within d
**Impute loess**

**Value**

A vector, same length as nrow(x), of adjusted y values

**Note**

Smoothing is conducted with a weighted-mean where; weights represent inverse standardized distance lags. Distance-based or neighbour-based smoothing can be specified by setting the desired neighbour smoothing method to a specified value then the other parameter to the potential maximum. For example; a constraint distance, including all neighbors within 1000 (d=1000) would require k to equal all of the potential neighbors (n-1 or k=nrow(x)-1).

**Examples**

```r
library(sp)
data(meuse)
coordinates(meuse) <- ~x+y

# Calculate distance weighted mean on cadmium variable in meuse data
cadmium.idw <- idw.smoothing(meuse, 'cadmium', k=nrow(meuse), d = 1000)
meuse$data$cadmium.wm <- cadmium.idw

opar <- par(no.readonly=TRUE)
par(mfrow=c(2,1))
plot(density(meuse$data$cadmium), main='Cadmium')
plot(density(meuse$data$cadmium.wm), main='IDW Cadmium')
par(opar)
```

---

**Impute.loess**  
**Impute loess**

**Description**

Imputes missing data or smooths using Loess regression

**Usage**

```r
impute.loess(y, s = 0.2, smooth = FALSE)
```

**Arguments**

- `y` A vector to impute
- `s` Smoothing parameter ()
- `smooth` (FALSE/TRUE) Smooth data, else only replace NA's
Details

Performs a local polynomial regression to smooth data or to impute NA values. The minimal number of non-NA observations to reliably impute/smooth values is 6. There is not a reliably way to impute NA’s on the tails of the distributions so if the missing data is in the first or last position of the vector it will remain NA. Please note that smooth needs to be TRUE to return a smoothed vector, else only NA’s will be imputed.

Value

a vector the same length as x with NA values filled or the data smoothed (or both).

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

data(cor.data)
d <- cor.data[[1]][,2]
plot(d, type="l")
lines(impute.loess(d, s=0.3, smooth=TRUE), lwd=2, col="red")

# add some NA's
d <- d[1:100]
d[sample(30:70, 5)] <- NA
d
impute.loess(d, s=0.2)

---

insert

Insert a row or column into a data.frame

Description

Inserts a new row or column into a data.frame at a specified location

Usage

insert(x, MARGIN = 1, value = NULL, idx, name = NULL)

Arguments

- **x**: Existing data.frame
- **MARGIN**: Insert a 1 = row or 2 = column
- **value**: A vector of values equal to the length of MARGIN, if nothing specified values with be NA
- **idx**: Index position to insert row or column
- **name**: Name of new column (not used for rows, MARGIN=1)
Value

A data.frame with the new row or column inserted

Note

Where there are methods to easily add a row/column to the end or beginning of a data.frame, it is not straightforward to insert data at a specific location within the data.frame. This function allows for inserting a vector at a specific location e.g., between columns or rows 1 and 2 where row/column 2 is moved to the 3rd position and a new vector of values is inserted into the 2nd position.

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

d <- data.frame(ID=1:10, y=runif(10))

# insert row
insert(d, idx=2)
insert(d, value=c(20,0), idx=2)

# insert column
insert(d, MARGIN=2, idx=2)
insert(d, MARGIN = 2, value = rep(0,10), idx=2, name="x")

Description

Inserts new values into a vector at specified positions

This function inserts new values at specified positions in a vector. It does not replace existing values. If a single value is provided for y and l represents multiple positions y will be replicated for the length of l. In this way you can insert the same value at multiple locations.

Usage

insert.values(x, value, index)

Arguments

x A vector to insert values
value Values to insert into x
index Index position(s) to insert y values into x
Value
A vector with values of y inserted into x and the position(s) defined by the index

Author(s)
Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples
(x=1:10)

  # Insert single value in one location
  insert.values(x, 100, 2)

  # Insert multiple values in multiple locations
  insert.values(x, c(100,200), c(2,8))

  # Insert single value in multiple locations
  insert.values(x, NA, c(2,8))

is.empty

Description
evaluates empty elements in a vector
This function evaluates if an element in a vector is empty the na.empty argument allows for evaluat-
ing NA values (TRUE if NA) and all.na returns a TRUE if all elements are NA. The trim argument
trims a character string to account for the fact that c(" ") is not empty but, a vector with c(""") is
empty. Using trim = TRUE will force both to return TRUE

Usage
is.empty(x, all.na = FALSE, na.empty = TRUE, trim = TRUE)

Arguments
x     A vector to evaluate elements
all.na (FALSE / TRUE) Return a TRUE if all elements are NA
na.empty (TRUE / FALSE) Return TRUE if element is NA
trim  (TRUE / FALSE) Trim empty strings

Value
A Boolean indicating empty elements in a vector, if all.na = FALSE a TRUE/FALSE value will be
returned for each element in the vector
is.whole

Author(s)
Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

```r
is.empty( c("") )
is.empty( c(" ") )
is.empty( c(" "), trim=FALSE )

is.empty( c("",NA,1) )
is.empty( c("",NA,1), na.empty=FALSE )

is.empty( c(NA,NA,NA) )
is.empty( c(NA,NA,NA), all.na=TRUE )
is.empty( c(NA,2,NA), all.na=TRUE )

any( is.empty( c("",2,3) ) )
any( is.empty( c(1,2,3) ) )
```

---

is.whole is.whole

Description

Boolean for evaluating whole numbers

Usage

```r
is.whole(a, tol = 0.0000001)
```

Arguments

- `a` A numeric vector to evaluate, only first element will be evaluated
- `tol` numeric >= 0, differences smaller than tolerance are not reported

Value

A Boolean indicating if number is whole or float

Author(s)
Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

```r
is.whole( 1 )
is.whole( 1.5 )
is.whole( 0.5 )
```
**kde.2D**  
2-dimensional kernel density estimate

**Description**  
Calculates 2-dimensional kernel density estimate over specified extent

**Usage**  
kde.2D(...)  

**Arguments**  
... Parameters to be passed to the modern version of the function

---

**kendall**  
Kendall tau trend with continuity correction for time-series

**Description**  
Calculates a nonparametric statistic for a monotonic trend based on the Kendall tau statistic and the Theil-Sen slope modification

**Usage**  
kendall(  
  y,  
  tau = TRUE,  
  p.value = TRUE,  
  z.value = TRUE,  
  confidence = TRUE,  
  intercept = TRUE,  
  prewhiten = FALSE,  
  na.rm,  
  ...  
)

**Arguments**  
  
y  
  A vector representing a timeseries with >= 8 obs  
  tau  
  (FALSE/TRUE) return tau values  
  p.value  
  (FALSE/TRUE) return p.values  
  z.value  
  (FALSE/TRUE) return z values  
  confidence  
  (FALSE/TRUE) return 95 pct confidence levels
intercept (FALSE/TRUE) return intercept values
prewhiten (FALSE/TRUE) Apply autocorrelation correction using pre-whitening
na.rm (FALSE/TRUE) Remove NA values
... Not used

Details

This function implements Kendall’s nonparametric test for a monotonic trend using the Theil-Sen (Theil 1950; Sen 1968; Siegel 1982) method to estimate the slope and related confidence intervals. Critical values are $Z > 1.96$ representing a significant increasing trend and a $Z < -1.96$ a significant decreasing trend ($p < 0.05$). The null hypothesis can be rejected if $\tau = 0$. There is also an option for autocorrelation correction using the method proposed in Yue & Wang (2002).

Value

Depending on arguments, a vector containing:

- value 1 Theil-Sen slope, always returned
- value 2 Kendall’s tau two-sided test, if tau TRUE
- value 3 intercept for trend if intercept TRUE, not if prewhitened
- value 4 p value for trend fit if p.value TRUE
- value 5 Z value for trend fit if z.value TRUE
- value 6 lower confidence level at 95-pct if confidence TRUE, not if prewhitened
- value 7 upper confidence level at 95-pct if confidence TRUE, not if prewhitened

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


kl.divergence  

**Kullback-Leibler divergence (relative entropy)**

**Description**

Calculates the Kullback-Leibler divergence (relative entropy) between unweighted theoretical component distributions. Divergence is calculated as: \( \int [f(x) \log f(x) - \log g(x)] dx \) for distributions with densities \( f() \) and \( g() \).

**Usage**

\[
\text{kl.divergence}(\text{object, eps = 10^{-4}, overlap = TRUE})
\]

**Arguments**

- **object**: Matrix or dataframe object with \( \geq 2 \) columns
- **eps**: Probabilities below this threshold are replaced by this threshold for numerical stability.
- **overlap**: Logical, do not determine the KL divergence for those pairs where for each point at least one of the densities has a value smaller than eps.

**Value**

pairwise Kullback-Leibler divergence index (matrix)

**Author(s)**

Jeffrey S. Evans <jeffrey_evans@tnc.org>

**References**


**Examples**

\[
\begin{align*}
x & \leftarrow \text{seq(-3, 3, length=200)} \\
y & \leftarrow \text{cbind(n=dnorm(x), t=dt(x, df=10))} \\
\text{matplot(x, y, type='l')} \\
\text{kl.divergence(y)} \\
\text{# extract value for last column} \\
\text{kl.divergence(y[,1:2])[3:3]}
\end{align*}
\]
knn

Spatial K nearest neighbor

Description

Find K nearest neighbors for two spatial objects

Finds nearest neighbor in x based on y and returns rownames, index and distance. If ids is NULL, rownames of x are returned. If coordinate matrix provided, columns need to be ordered [X,Y]. If a radius for d is specified than a maximum search radius is imposed. If no neighbor is found, a neighbor is not returned

You can specify weights to act as covariates for x and y. The vectors or matrices must match row dimensions with x and y as well as columns matching between weights. In other words, the covariates must match and be numeric.

Usage

knn(
  y,
  x,
  k = 1,
  d = NULL,
  ids = NULL,
  weights.y = NULL,
  weights.x = NULL,
  indexes = FALSE
)

Arguments

y Spatial points or polygons object or coordinates matrix
x Spatial points or polygons object or coordinates matrix
k Number of neighbors
d Optional search radius
ids Optional column of ID’s in x
weights.y A vector or matrix representing covariates of y
weights.x A vector or matrix representing covariates of x
indexes (FALSE/TRUE) Return row indexes of x neighbors

Value

A data.frame with row indexes (optional), rownames, ids (optional) and distance of k

Author(s)

Jeffrey S. Evans <jeffrey.evans@tnc.org>
See Also

nn2 for details on search algorithm

Examples

```r
library(sp)
data(meuse)
coordinates(meuse) <- ~x+y

idx <- sample(1:nrow(meuse), 10)
pts <- meuse[idx,]
meuse <- meuse[-idx,]
meuse$IDS <- 1:nrow(meuse)

# Find 2 neighbors in meuse
(nn <- knn(pts, meuse, k=2, ids = "IDS", indexes = TRUE))
plot(pts, pch=19, main="KNN"
  points(meuse[nn[,1],], pch=19, col="red")

# Using covariates (weights)
wx = as.matrix(meuse@data[,1:3])
wy = as.matrix(pts@data[,1:3])

(nn <- knn(pts, meuse, k=2, ids = "IDS", indexes = TRUE,
  weights.y=wy, weights.x=wx))
plot(pts, pch=19, main="KNN"
  points(meuse[nn[,1],], pch=19, col="red")

# Using coordinate matrices
y <- coordinates(pts)
x <- coordinates(meuse)
knn(y, x, k=2)
```

---

**land.metrics**

Landscape metrics for points and polygons

Description

Calculates a variety of landscape metrics, on binary rasters, for polygons or points with a buffer distance

Usage

```r
land.metrics(...)
```

Arguments

... Parameters to be passed to the modern version of the function
local.min.max

Examples

## Not run:
library(landscapemetrics)
library(raster)

data(landscape)
points <- matrix(c(10, 5, 25, 15, 5, 25),
                 ncol = 2, byrow = TRUE)

sample_lsm(landscape, y = points, size = 10,
           level = "landscape", type = "diversity metric",
           classes_max = 3,
           verbose = FALSE)

## End(Not run)

---

local.min.max  Local minimum and maximum

Description

Calculates the local minimums and maximums in a numeric vector, indicating inflection points in the distribution.

Usage

local.min.max(x, dev = mean, plot = TRUE, add.points = FALSE, ...)

Arguments

x      A numeric vector
dev    Deviation statistic (mean or median)
plot   plot the minimum and maximum values with the distribution (TRUE/FALSE)
add.points Should all points of x be added to plot (TRUE/FALSE)
...    Arguments passed to plot

Value

A list object with:

- minima - minimum local values of x
- maxima - maximum local values of x
- mindev - Absolute deviation of minimum from specified deviation statistic (dev argument)
- maxdev - Absolute deviation of maximum from specified deviation statistic (dev argument)
Note

Useful function for identifying inflection or enveloping points in a distribution

Author(s)

Jeffrey S. Evans jeffrey_evans@tnc.org

Examples

```r
x <- rnorm(100, mean=1500, sd=800)
(lmm <- local.min.max(x, dev=mean, add.points=TRUE,
                      main="Local Minima and Maxima")

# return only local minimum values
local.min.max(x)$minima
```

Description

Bootstrap of a Local Polynomial Regression (loess)

The function fits a loess curve and then calculates a symmetric nonparametric bootstrap with a confidence region. Fitted curves are evaluated at a fixed number of equally-spaced x values, regardless of the number of x values in the data. Some replicates do not include the values at the lower and upper end of the range of x values. If the number of such replicates is too large, it becomes impossible to construct a confidence region that includes a fraction "confidence" of the bootstrap replicates. In such cases, the left and/or right portion of the confidence region is truncated.

Usage

```r
loess.boot(x, y, nreps = 100, confidence = 0.95, ...)
```

Arguments

- **x**: Independent variable
- **y**: Dependent variable
- **nreps**: Number of bootstrap replicates
- **confidence**: Fraction of replicates contained in confidence region
- **...**: Additional arguments passed to loess function
Value

list object containing

- nreps Number of bootstrap replicates
- confidence Confidence interval (region)
- span alpha (span) parameter used loess fit
- degree polynomial degree used in loess fit
- normalize Normalized data (TRUE/FALSE)
- family Family of statistic used in fit
- parametric Parametric approximation (TRUE/FALSE)
- surface Surface fit, see loess.control
- data data.frame of x,y used in model

- fit data.frame including:
  1. x - Equally-spaced x index (see NOTES)
  2. y.fit - loess fit
  3. up.lim - Upper confidence interval
  4. low.lim - Lower confidence interval
  5. stddev - Standard deviation of loess fit at each x value

Author(s)

Jeffrey S. Evans jeffrey_evans@tnc.org

References

Cleveland, WS, (1979) Robust Locally Weighted Regression and Smoothing Plots Journal of the American Statistical Association 74:829-836


Examples

n=1000
x <- seq(0, 4, length.out=n)
y <- sin(2*x)+ 0.5*x + rnorm(n, sd=0.5)
sb <- loess.boot(x, y, nreps=99, confidence=0.90, span=0.40)
plot(sb)
Description

Calculates a local polynomial regression fit with associated confidence intervals

Usage

loess.ci(y, x, p = 0.95, plot = FALSE, ...)

Arguments

y Dependent variable, vector
x Independent variable, vector
p Percent confidence intervals (default is 0.95)
plot Plot the fit and confidence intervals
...

Value

A list object with:

• loess Predicted values
• se Estimated standard error for each predicted value
• lci Lower confidence interval
• uci Upper confidence interval
• df Estimated degrees of freedom
• rs Residual scale of residuals used in computing the standard errors

Author(s)

Jeffrey S. Evans jeffrey_evans@tnc.org

References

Examples

```r
x <- seq(-20, 20, 0.1)
y <- sin(x)/x + rnorm(length(x), sd=0.03)
p <- which(y == "NaN")
y <- y[-p]
x <- x[-p]

opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
lci <- loess.ci(y, x, plot=TRUE, span=0.10)
lci <- loess.ci(y, x, plot=TRUE, span=0.30)
lci <- loess.ci(y, x, plot=TRUE, span=0.50)
lci <- loess.ci(y, x, plot=TRUE, span=0.80)
par(opar)
```

Description

Performs a logistic (binomial) or auto-logistic (spatially lagged binomial) regression using maximum likelihood or penalized maximum likelihood estimation.

It should be noted that the auto-logistic model (Besag 1972) is intended for exploratory analysis of spatial effects. Auto-logistic are know to underestimate the effect of environmental variables and tend to be unreliable (Dormann 2007). Wij matrix options under style argument - B is the basic binary coding, W is row standardized (sums over all links to n), C is globally standardized (sums over all links to n), U is equal to C divided by the number of neighbors (sums over all links to unity) and S is variance-stabilizing. Spatially lagged y defined as: W(y)ij=sum_j(Wij yj)/ sum_j(Wij) where; Wij=1/Euclidean(i,j) If the object passed to the function is an sp class there is no need to call the data slot directly via "object@data", just pass the object name.

Usage

```r
logistic.regression(
  ldata,
  y,
  x,
  penalty = TRUE,
  autologistic = FALSE,
  coords = NULL,
  bw = NULL,
  type = "inverse",
  style = "W",
  longlat = FALSE,
  ...
)
```
logistic.regression

Arguments

- **ldata**: data.frame object containing variables
- **y**: Dependent variable (y) in ldata
- **x**: Independent variable(s) (x) in ldata
- **penalty**: Apply regression penalty (TRUE/FALSE)
- **autologistic**: Add auto-logistic term (TRUE/FALSE)
- **coords**: Geographic coordinates for auto-logistic model matrix or sp object.
- **bw**: Distance bandwidth to calculate spatial lags (if empty neighbors result, need to increase bandwidth). If not provided it will be calculated automatically based on the minimum distance that includes at least one neighbor.
- **type**: Neighbor weighting scheme (see autocov_dist)
- **style**: Type of neighbor matrix (Wij), default is mean of neighbors
- **longlat**: Are coordinates (coords) in geographic, lat/long (TRUE/FALSE)
- **...**: Additional arguments passed to lrm

Value

A list class object with the following components:

- `model`: lrm model object (rms class)
- `bandwidth`: If AutoCov = TRUE returns the distance bandwidth used for the auto-covariance function
- `diagTable`: data.frame of regression diagnostics
- `coefTable`: data.frame of regression coefficients
- `Residuals`: data.frame of residuals and standardized residuals
- `AutoCov`: If an auto-logistic model, AutoCov represents lagged auto-covariance term

Author(s)

Jeffrey S. Evans jeffrey_evans@tnc.org

References


See Also

- `lrm`
- `autocov_dist`
Examples

```r
require(sp)
require(spdep)
require(rms)
data(meuse)
coordinates(meuse) <- ~x+y
meuse@data <- data.frame(DepVar=rbinom(dim(meuse)[1], 1, 0.5), meuse@data)

### Logistic model
lmodel <- logistic.regression(meuse, y='DepVar',
                                     x=c('dist','cadmium','copper'))
lmodel$model
lmodel$diagTable
lmodel$coefTable

### Logistic model with factorial variable
lmodel <- logistic.regression(meuse, y='DepVar',
                                     x=c('dist','cadmium','copper', 'soil'))
lmodel$model
lmodel$diagTable
lmodel$coefTable

### Auto-logistic model using 'autocov_dist' in 'spdep' package
lmodel <- logistic.regression(meuse, y='DepVar',
                                     x=c('dist','cadmium','copper'), autologistic=TRUE,
                                     coords=coordinates(meuse), bw=5000)
lmodel$model
lmodel$diagTable
lmodel$coefTable
est <- predict(lmodel$model, type='fitted.ind')

#### Add residuals, standardized residuals and estimated probabilities
VarNames <- rownames(lmodel$model$var)[-1]
meuse@data$AutoCov <- lmodel$AutoCov
meuse@data <- data.frame(meuse@data, Residual=lmodel$Residuals[,1],
                          StdResid=lmodel$Residuals[,2], Probs=predict(lmodel$model,
                          meuse@data[,VarNames],type='fitted'))

#### Plot fit and probabilities
resid(lmodel$model, "partial", pl="loess")
# plot residuals
resid(lmodel$model, "partial", pl=TRUE)

# global test of goodness of fit
resid(lmodel$model, "gof")

# Approx. leave-out linear predictors
lp1 <- resid(lmodel$model, "lp1")

# Approx leave-out-1 deviance
-2 * sum(meuse@data$DepVar * lp1 + log(1-plogis(lp1)))
```
max_extent

Maximum extent of multiple rasters

Description
returns a extent polygon representing maximum extent of input rasters

Usage
max_extent(x, ...)

Arguments
x raster class object
... additional raster class objects

Value
a SpatialPolygons sp class object

Note
Creates a maximum extent polygon of all specified rasters

Author(s)
Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples
library(raster)
r1 <- raster(extent(61.87125, 76.64458, 23.90153, 37.27042))
r2 <- raster(extent(67.66625, 81.56847, 20.38458, 35.67347))
r3 <- raster(extent(72.64792, 84.38125, 5.91125, 28.13347))
(e <- max_extent(r1, r2, r3))
plot(e)
plot(as(extent(r1),"SpatialPolygons"),col="red",add=TRUE)
plot(as(extent(r2),"SpatialPolygons"),col="red",add=TRUE)
plot(as(extent(r3),"SpatialPolygons"),col="red",add=TRUE)

# plot estimated probabilities at points
spplot(meuse, c('Probs'))
moments

Description

Calculate statistical moments of a distribution

Usage

moments(x, plot = FALSE)

Arguments

x numeric vector
plot plot of distribution (TRUE/FALSE)

Value

A vector with the following values:

- min Minimum
- 25th 25th percentile
- mean Arithmetic mean
- gmean Geometric mean
- hmean Harmonic mean
- median 50th percentile
- 7th5 75th percentile
- max Maximum
- stdv Standard deviation
- var Variance
- cv Coefficient of variation (percent)
- mad Median absolute deviation
- skew Skewness
- kurt Kurtosis
- nmodes Number of modes
- mode Mode (dominate)

Author(s)

Jeffrey S. Evans jeffrey_evans@tnc.org
Examples

```r
x <- runif(1000, 0, 100)
(d <- moments(x, plot=TRUE))
(mode.x <- moments(x, plot=FALSE)[16])
```

**Description**

Autocorrelation plot (Anselin 1996), following Chen (2015), aka, Moran’s-I plot (univariate or bivariate)

**Usage**

```r
morans.plot(
x,
y = NULL,
coords = NULL,
type.ac = c("xy", "yx"),
dist.function = "inv.power",
scale.xy = TRUE,
scale.morans = FALSE,
...
)
```

**Arguments**

- `x`: Vector of x response variables
- `y`: Vector of y response variables
- `coords`: A matrix of coordinates corresponding to [x,y]
- `type.ac`: Type of autocorrelation plot ("xy", "yx")
- `dist.function`: ("inv.power", "neg.exponent")
- `scale.xy`: (TRUE/FALSE) scale the x,y vectors
- `scale.morans`: (FALSE/TRUE) standardize the Moran’s index to an expected [-1 to 1]?
- `...`: Additional arguments passed to plot

**Details**

The argument "type" controls the plot for x influencing y (type="xy") or y influencing x (type="yx"). If y is not defined then the statistic is univariate and only the "xy" plot will be available. The linear relationship between x and its spatial lag (Wx) is indicative of the spatial autoregressive process, underlying the spatial dependence. The statistic can be autocorrelation (univariate or cross-correlation (bivariate). The quadrants are the zero intercept for random autocorrelation and the red
morans.plot

line represents the trend in autocorrelation. The quadrants in the plot indicate the type of spatial association/interaction (Anselin 1996). For example the upper-left quadrant represents negative associations of low values surrounded by high and the lower-right quadrant represents negative associations of high values surrounded by low.

Value

A plot of the scaled variable against its spatially lagged values.

Note

if y is not specified the univariate statistic for x is returned. the coords argument is only used if k = NULL. Can also be an sp object with relevant x,y coordinate slot (ie., points or polygons). If w = NULL, the default method for deriving spatial weights matrix, options are: inverse power or negative exponent. If scale.xy = FALSE it is assumed that they are already scaled following Chen (2015).

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


Examples

library(sp)
library(spdep)
data(meuse)
coordinates(meuse) <- ~x+y

# Autocorrelation (univariate)
morans.plot(meuse$zinc, coords = coordinates(meuse))

# Cross-correlation of: x influencing y and y influencing x
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))
morans.plot(x=meuse$zinc, y=meuse$copper, coords = coordinates(meuse),
           scale.morans = TRUE)
morans.plot(x=meuse$zinc, y=meuse$copper, coords = coordinates(meuse),
           scale.morans = TRUE, type.ac="yx")
par(opar)
**mwCorr**

*Dutilleul moving window bivariate raster correlation*

**Description**

A bivariate raster correlation using Dutilleul’s modified t-test

**Usage**

`mwCorr(...)`

**Arguments**

... Parameters to be passed to the modern version of the function

---

**nni**

*Average Nearest Neighbor Index (NNI)*

**Description**

Calculates the NNI as a measure of clustering or dispersal

The nearest neighbor index is expressed as the ratio of the observed distance divided by the expected distance. The expected distance is the average distance between neighbors in a hypothetical random distribution. If the index is less than 1, the pattern exhibits clustering; if the index is greater than 1, the trend is toward dispersion or competition. The Nearest Neighbor Index is calculated as:

- Mean Nearest Neighbor Distance (observed) $D_{nn} = \text{sum}(\text{min}(Dij)/N)$
- Mean Random Distance (expected) $D_{e} = 0.5 \sqrt{A/N}$
- Nearest Neighbor Index $\text{NNI} = D_{nn}/D_{e}$ Where; $D=$neighbor distance, $A=$Area

**Usage**

`nni(x, win = "hull")`

**Arguments**

- `x` An sp point object
- `win` Type of window 'hull' or 'extent'

**Value**

list object containing NNI = nearest neighbor index, z.score = Z Score value, p = p value, expected.mean.distance = Expected mean distance, observed.mean.distance = Observed mean distance.
nth.values

Author(s)

Jeffrey S. Evans jeffrey_evans@tnc.org

References


Examples

```r
require(sp)
data(meuse)
coordinates(meuse) <- ~x+y
nni(meuse)
```

nth.values

Nth values

Description

Returns the Nth highest or lowest values in a vector

Usage

```r
nth.values(x, N = 2, smallest = FALSE)
```

Arguments

- `x`: Numeric vector
- `N`: Number of (Nth) values returned
- `smallest` (FALSE/TRUE) Return the highest, else smallest values

Value

Numeric vector of Nth values

Note

This function returns n lowest or highest elements in a vector

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>
Examples

nth.values(1:20, N=3, smallest = TRUE)
nth.values(1:20, N=3)

Description

Calculates the inhomogeneous O-ring point pattern statistic (Wiegand & Maloney 2004)

The function K(r) is the expected number of points in a circle of radius r centered at an arbitrary point (which is not counted), divided by the intensity l of the pattern. The alternative pair correlation function g(r), which arises if the circles of Ripley's K-function are replaced by rings, gives the expected number of points at distance r from an arbitrary point, divided by the intensity of the pattern. Of special interest is to determine whether a pattern is random, clumped, or regular.

Using rings instead of circles has the advantage that one can isolate specific distance classes, whereas the cumulative K-function confounds effects at larger distances with effects at shorter distances. Note that the K-function and the O-ring statistic respond to slightly different biological questions. The accumulative K-function can detect aggregation or dispersion up to a given distance r and is therefore appropriate if the process in question (e.g., the negative effect of competition) may work only up to a certain distance, whereas the O-ring statistic can detect aggregation or dispersion at a given distance r. The O-ring statistic has the additional advantage that it is a probability density function (or a conditioned probability spectrum) with the interpretation of a neighborhood density, which is more intuitive than an accumulative measure.

Usage

o.ring(x, inhomogeneous = FALSE, ...)

Arguments

x spatstat ppp object
inhomogeneous (FALSE/TRUE) Run homogeneous (pcf) or inhomogeneous (pcfinhom)
... additional arguments passed to pcf or pcfinhom

Value

plot of o-ring and data.frame with plot labels and descriptions

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>
References


Examples

library(spatstat.core)
data(lansing)
x <- spatstat.geom::unmark(split(lansing)$maple)
o.ring(x)

oli.asw  Query AWS-OLI

Description

Query of Amazon AWS OLI-Landsat 8 cloud service

Usage

oli.asw(path, row, dates, cloud.cover = 10, processing)

Arguments

path  landsat path
row  landsat row
dates  dates, single or start-stop range in YYYY-MM-DD format
cloud.cover  percent cloud cover
processing  processing level ("L1GT" or "L1T")

Value

data.frame object with:

• entityId - Granule ID
• L = Landsat
• X = Sensor
• SS = Satellite
• PPP = WRS path
• RRR = WRS row
• YYYYMMD = Acquisition date
• yyyymmdd = Processing date
• CC = Collection number
• TX = Collection category
• acquisitionDate - POSIXct YYYY-MM-DD (eg., 2015-01-02)
• cloudCover -
• processingLevel - USGS processing level
• path - Landsat path
• row - Landsat row

Note

Amazon’s AWS cloud service is hosting OLI Landsat 8 data granules https://registry.opendata.aws/landsat-8 https://aws.amazon.com/blogs/aws/start-using-landsat-on-aws/

USGS Landsat collections: https://www.usgs.gov/core-science-systems/nli/landsat Pre-collection processing levels: "L1T", "L1GT", "L1G" Collection 1 processing levels: "L1TP", "L1GT", "L1GS" "L1T" and "L1TP" - Radiometrically calibrated and orthorectified (highest level processing) "L1GT" and "L1GT" - Radiometrically calibrated and systematic geometric corrections "L1G" and "L1GS" - Radiometrically calibrated with systematic ephemeris correction

Author(s)

Jeffrey S. Evans <jeffrey.evans@tnc.org>

Examples

## Not run:

```r
# Query path 126, row 59, 2013-04-15 to 2017-03-09, <20% cloud cover
(p126r59.oli <- oli.asw(path=126, row=59, dates = c("2013-04-15", "2017-03-09"),
  cloud.cover = 20) )

# Download images from query
  "_B11.TIF", "_BQA.TIF","_MTL.txt")
for(i in 1:length(p126r59.oli$download_url)) {
  oli.url <- gsub("/index.html","",p126r59.oli$download_url[i])
  all.bands <- paste(oli.url, paste0(unlist(strsplit(oli.url, "/"))[8], bands), sep="/")
  for(j in all.bands) {
    try(utils::download.file(url=j, destfile=basename(j), mode = "wb"))
  }
}
## End(Not run)```
optimal.k

Description

Find optimal k of k-Medoid partitions using silhouette widths

Usage

optimal.k(x, nk = 10, plot = TRUE, cluster = TRUE, clara = FALSE, ...)

Arguments

x Numeric dataframe, matrix or vector
nk Number of clusters to test (2:nk)
plot Plot cluster silhouettes(TRUE/FALSE)
cluster Create cluster object with optimal k
clara Use clara model for large data
... Additional arguments passed to clara

Value

Object of class clust "pam" or "clara"

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


See Also

pam for details on Partitioning Around Medoids (PAM)
clara for details on Clustering Large Applications (clara)

Examples

require(cluster)
x <- rbind(cbind(rnorm(10,0,0.5), rnorm(10,0,0.5)),
          cbind(rnorm(15,5,0.5), rnorm(15,5,0.5)))

clust <- optimal.k(x, 20, plot=TRUE, cluster=TRUE)
plot(silhouette(clust), col = c('red', 'green'))
plot(clust, which.plots=1, main='K-Medoid fit')
# Extract multivariate and univariate mediods (class centers)
clust$medoids
  pam(x[,1], 1)$medoids

# join clusters to data
x <- data.frame(x, k=clust$clustering)

---

**optimized.sample.variance**

*Optimized sample variance*

**Description**

Draws an optimal sample that minimizes or maximizes the sample variance

**Usage**

```r
optimized.sample.variance(x, n, type = "maximized")
```

**Arguments**

- **x**: A vector to draw a sample from
- **n**: Number of samples to draw
- **type**: Type of sample variance optimization c("maximized", "minimized")

**Value**

A data.frame with "idx" representing the index of the original vector and "y" is the value of the sampled data

**Author(s)**

Jeffrey S. Evans <jeffrey_evans@tnc.org>

**Examples**

```r
library(sp)
data(meuse)
  coordinates(meuse) <- ~x+y

n = 15
# Draw n samples that maximize the variance of y
( max.sv <- optimized.sample.variance(meuse$zinc, 15) )

# Draw n samples that minimize the variance of y
( min.sv <- optimized.sample.variance(meuse$zinc, 15, type="minimized") )
```
```r
# Plot results
plot(meuse, pch=19, col="grey")
plot(meuse[max.sv$idx[,], col="red", add=TRUE, pch=19]
plot(meuse[min.sv$idx[,], col="blue", add=TRUE, pch=19)
box()
legend("topleft", legend=c("population","maximized variance",
"minimized variance"), col=c("grey","red","blue"),
pch=c(19,19,19))

# Raster example (not memory safe)
library(raster)
r <- raster(system.file("external/test.grd", package="raster"))

# Calculate optimal sample variance and coerce to SpatialPointsDataFrame
# using xyFromCell
(min.sv <- optimized.sample.variance(getValues(r), n, type="minimized") )
min.sv <- sp::SpatialPointsDataFrame(xyFromCell(r, min.sv[,"idx"],
spatial=TRUE), data=min.sv)

(max.sv <- optimized.sample.variance(getValues(r), n )
max.sv <- sp::SpatialPointsDataFrame(xyFromCell(r, max.sv[,"idx"],
spatial=TRUE), data=max.sv)

plot(r)
plot(max.sv, col="blue", add=TRUE, pch=19)
plot(min.sv, col="red", add=TRUE, pch=19)
box()
legend("topleft", legend=c("maximized variance", "minimized variance"),
col=c("red","blue"), pch=c(19,19))
```

---

### outliers

**Outliers**

**Description**

Identify outliers using modified Z-score

**Usage**

```r
outliers(x, s = 1.4826)
```

**Arguments**

- **x**: A numeric vector
- **s**: Scaling factor for mad statistic

**Value**

value for the modified Z-score
Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References

Iglewicz, B. & D.C. Hoaglin (1993) How to Detect and Handle Outliers, American Society for Quality Control, Milwaukee, WI.

Examples

# Create data with 3 outliers
x <- seq(0.1, 5, length=100)
x[98:100] <- c(100, 55, 250)

# Calculate Z score
Z <- outliers(x)

# Show number of extreme outliers using Z-score
length(Z[Z > 9.9])

# Remove extreme outliers
x <- x[!which(Z > 9.9)]

---

**overlap**  

*Niche overlap (Warren’s-I)*

Description

Similarity Statistic for Quantifying Niche Overlap using Warren’s-I

The overlap function computes the I similarity statistic (Warren et al. 2008) of two overlapping niche estimates. Similarity is based on the Hellinger distance. It is assumed that the input data share the same extent and cells size and all values are positive.

The I similarity statistic sums the pair-wise differences between two predictions to create a single value representing the similarity of the two distributions. The I similarity statistic ranges from a value of 0, where two distributions have no overlap, to 1 where two distributions are identical (Warren et al., 2008). The function is based on code from Jeremy VanDerWal

Usage

`overlap(x, y)`

Arguments

- `x`: A matrix, rasterLayer or sp raster class object
- `y`: A matrix, rasterLayer or sp raster class object with the same dimensions of `x`
Value

A value representing the I similarity statistic

Author(s)

Jeffrey Evans <jeffrey_evans@tnc.org> and Jeremy VanDerWal

References


Examples

```r
# add degree of separation in two matrices
p1 <- abs(matrix(1:50, nr=50, nc=50) + runif(n = 2500, min = -1, max = 1))
p2 <- abs(matrix(1:50, nr=50, nc=50) + rnorm(n = 2500, mean = 1, sd = 1))

# High overlap/similarity
(I <- overlap(p1, p2))
```

Description

Creates a point sample of polygons where n is based on percent area

Usage

```r
parea.sample(
  x,  
pct = 0.1,  
join = FALSE,  
min.samp = 1,  
max.samp = NULL,  
sf = 4046.86,  
stype = "hexagonal",  
...  
)
```
Arguments

- `x`: A sp SpatialPolygonsDataFrame object
- `pct`: Percent of area sampled
- `join`: Join polygon attributed to point sample
- `min.samp`: Minimum number of samples
- `max.samp`: Maximum number of samples
- `sf`: Scaling factor (default is meters to acres conversion factor)
- `stype`: Sampling type ("random", "regular", "nonaligned", "hexagonal")
- `...`: Additional arguments passed to spsample

Value

A SpatialPointsDataFrame with polygon samples

Note

This function results in an adaptive sample based on the area of each polygon

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

```r
require(sp)
sr1=Polygons(list(Polygon(cbind(c(180114, 180553, 181127, 181477, 181294,
181007, 180409, 180162, 180114), c(332349, 332057, 332342, 333250, 333558,
333676, 332618, 332413, 332349)))),'1'))
sr2=Polygons(list(Polygon(cbind(c(180042, 180545, 180553, 180314, 179955,
179142, 179437, 179524, 179979, 180042), c(332373, 332026, 330889,
330683, 331133, 331623, 332152, 332357, 332373)))),'2'))
sr=SpatialPolygons(list(sr1,sr2))
srdf=SpatialPolygonsDataFrame(sr, data.frame(row.names=c('1','2'), PIDS=1:2))

ars <- parea.sample(srdf, pct=0.20, stype='random')
plot(ars, pch=20, add=TRUE)
```
parse.bits

Parse bits

Description

Returns specified bit value based on integer input.

Data such as MODIS the QC band are stored in bits. This function returns the value(s) for specified bit. For example, the MODIS QC flag are bits 0-1 with the bit value 00 representing the "LST produced, good quality" flag. When exported from HDF the QC bands are often in an 8 bit integer range (0-255). With this function you can parse the values for each bit to assign the flag values.

Usage

parse.bits(x, bit, depth = 8, order = c("reverse", "none"))

Arguments

x
  Integer value

bit
  A single or vector of bits to return

depth
  The depth (length) of the bit range, default is 8

order
  c("reverse", "none") sort order for the bits

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

# Return value for bit 5 for integer value 100
parse.bits(100, 5)

# Return value(s) for bits 0 and 1 for integer value 100
parse.bits(100, c(0,1))

# Return value(s) for bits 0 and 1 for integer values 0-255
for(i in 0:255) { print(parse.bits(i, c(0,1))) }

## Not run:
#### Applied Example using Harmonized Landsat Sentinel-2 QC

# Create dummy data and qc band
library(raster)
r <- raster(nrow=100, ncol=100)
r[] <- round(runif(ncell(r), 0, 1))
qc <- raster(nrow=100, ncol=100)
qc[] <- round(runif(ncell(qc), 64, 234))
# Calculate bit values from QC table
( qc_bits <- data.frame(int=0:255, 
  cloud = unlist(lapply(0:255, FUN=parse.bits, bit=1)), 
  shadow = unlist(lapply(0:255, FUN=parse.bits, bit=3)), 
  acloud = unlist(lapply(0:255, FUN=parse.bits, bit=2)), 
  cirrus = unlist(lapply(0:255, FUN=parse.bits, bit=0)), 
  aerosol = unlist(lapply(0:255, FUN=parse.bits, bit=c(7,6))) ) )

# Query the results to create a vector of integer values indicating what to mask
m <- sort(unique(qc_bits[which(qc_bits$cloud == 1), 
  which(qc_bits$shadow == 1) ])$int))

# Apply queried integer values to mask image with QA band
qc[qc %in% m] <- NA
r <- mask(r, qc)

## End(Not run)

---

**partial.cor**

*Partial and Semi-partial correlation*

**Description**

Calculates a partial or semi-partial correlation with parametric and nonparametric options.

**Usage**

```r
partial.cor(
  x, 
  y, 
  z, 
  method = c("partial", "semipartial"), 
  statistic = c("kendall", "pearson", "spearman")
)
```

**Arguments**

- `x` A vector, data.frame or matrix with 3 columns
- `y` A vector same length as `x`
- `z` A vector same length as `x`
- `method` Type of correlation: "partial" or "semipartial"
- `statistic` Correlation statistic, options are: "kendall", "pearson", "spearman"
Details

Partial and semipartial correlations show the association between two variables when one or more peripheral variables are controlled to hold them constant.

Suppose we have three variables, X, Y, and Z. Partial correlation holds constant one variable when computing the relations two others. Suppose we want to know the correlation between X and Y holding Z constant for both X and Y. That would be the partial correlation between X and Y controlling for Z. Semipartial correlation holds Z constant for either X or Y, but not both, so if we wanted to control X for Z, we could compute the semipartial correlation between X and Y holding Z constant for X.

Value
data.frame containing:

• correlation correlation coefficient
• p.value p-value of correlation
• test.statistic test statistic
• n sample size
• Method indicating partial or semipartial correlation
• Statistic the correlation statistic used

Author(s)
Jeffrey S. Evans <jeffrey.evans@tnc.org>

Examples

```r
air.flow = stackloss[,1]
water.temperature = stackloss[,2]
acid = stackloss[,3]

# Partial using Kendall (nonparametric) correlation
partial.cor(air.flow, water.temperature, acid)

scholar <- data.frame(
  HSGPA=c(3.0, 3.2, 2.8, 2.5, 3.2, 3.8, 3.9, 3.8, 3.5, 3.1),
  FGPA=c(2.8, 3.0, 2.8, 2.2, 3.3, 3.3, 3.5, 3.7, 3.4, 2.9),
  SATV =c(500, 550, 450, 400, 600, 650, 700, 550, 650, 550))

# Standard Pearson's correlations between HSGPA and FGPA
cor(scholar[,1], scholar[,2])

# Partial correlation using Pearson (parametric) between HSGPA and FGPA, controlling for SATV
partial.cor(scholar, statistic="pearson")

# Semipartial using Pearson (parametric) correlation
partial.cor(x=scholar[,2], y=scholar[,1], z=scholar[,3],
  method="semipartial", statistic="pearson")
```
plot.effect.size  

Description

Plot function for effect.size object

Usage

## S3 method for class 'effect.size'
plot(x, ...)

Arguments

x  A effect.size object
...
Additional arguments passed to plot

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

---

plot.loess.boot  

Description

Plot function for loess.boot object

Usage

## S3 method for class 'loess.boot'
plot(x, ...)

Arguments

x  A loess.boot object
...
Additional arguments passed to plot

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>
point.in.poly

References

Cleveland, WS, (1979) Robust Locally Weighted Regression and Smoothing Plots Journal of the American Statistical Association 74:829-836


Examples

\begin{verbatim}
n=1000
x <- seq(0, 4, length.out=n)
y <- sin(2*x)+ 0.5*x + rnorm(n, sd=0.5)
sb <- loess.boot(x, y, nreps = 99, confidence = 0.90, span = 0.40)
plot(sb)
\end{verbatim}

Description

Intersects point and polygon feature classes and adds polygon attributes to points

If duplicate argument is TRUE and more than one polygon intersection occurs, points will be duplicated (new row added) and all attributes joined. However, if duplicate is FALSE, with duplicate intersections, a new column for each unique intersecting polygon will be returned and the points will not be duplicated. For example, if a point intersect three polygons, three new columns will be added representing the polygons ID.

Usage

point.in.poly(x, y, sp = TRUE, duplicate = TRUE, ...)

Arguments

- \texttt{x} \hspace{1cm} \text{sp SpatialPointsDataFrame or SpatialPoints or sf point object}
- \texttt{y} \hspace{1cm} \text{sp SpatialPolygonsDataFrame or sf polygon object}
- \texttt{sp} \hspace{1cm} (TRUE/FALSE) Return an sp class object, else returns sf class object
- \texttt{duplicate} \hspace{1cm} (TRUE/FALSE) Return duplicated features with more than one polygon intersection
- \texttt{...} \hspace{1cm} Additional arguments passed to \texttt{sf::st_join}

Value

A SpatialPointsDataFrame or sf
Author(s)
Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

#### Simple one-to-one feature overlay.
```
require(sp)
data(meuse)
coordinates(meuse) = ~x+y
meuse@data$test.na <- NA
sr1=Polygons(list(Polygon(cbind(c(180114, 180553, 181127, 181477, 181294, 
181007, 180409, 180162, 180114), c(332349, 332057, 332342, 332320, 333558, 
333676, 332618, 332413, 332349)))),'10')
sr2=Polygons(list(Polygon(cbind(c(180042, 180545, 180553, 180314, 179955, 179142, 
179437, 179524, 179979, 180042), c(332373, 332026, 331426, 330889, 330683, 
331133, 331623, 332152, 332357, 332373)))),'20')
sr3=Polygons(list(Polygon(cbind(c(179110, 179907, 180433, 180712, 180752, 180329, 
179875, 179668, 179572, 179269, 178879, 178600, 178544, 179046, 179110), 
c(331086, 330620, 330494, 330265, 330075, 330233, 330336, 330004, 
329783, 329665, 329720, 329933, 330478, 331062, 331086)))),'30')
sr4=Polygons(list(Polygon(cbind(c(180304, 180403, 179632, 179420, 180304), 
c(332791, 333204, 333635, 333058, 332791)))),'40')
sr=SpatialPolygons(list(sr1,sr2,sr3,sr4))
polys=SpatialPolygonsDataFrame(sr, data.frame(row.names=c('10','20','30','40'), 
PIDS=1:4, y=runif(4)))
polys@data$pid <- polys@data$PIDS + 100
```
```
plot(polys)
plot(meuse, pch=19, add=TRUE)
# Point in polygon overlay
pts.poly <- point.in.poly(meuse, polys)
head(pts.poly@data)
```
```
# Count points in each polygon
tapply(pts.poly$cadmium, pts.poly$pid, FUN=length)
```

#### Complex many-to-one feature overlay.
```
require(sf)
p <- sf::st_polygon(list(rbind(c(0,0), c(1,0), c(1,1), c(0,1), c(0,0))))
polys <- sf::st_sf(sf::st_sfc(p, p + c(.8, .2), p + c(.2, .8)))
pts <- sf::st_sf(sf::st_sample(polys, size=100))
```
```
# Duplicates points for each new polygon, no attributes so returns IDs for features
pts.poly.dup <- point.in.poly(pts, polys)
head(pts.poly.dup@data)
```
```
# Not run:
# **** Should throw error due to lack of attributes ****
pts.poly <- point.in.poly(pts, polys, duplicate = FALSE)
```
## End(Not run)

# Coerce to sp class objects
x <- as(pts, "Spatial")
x <- SpatialPointsDataFrame(x, data.frame(IDS=1:nrow(x), pty=runif(nrow(x))))
y <- as(polys, "Spatial")
y <- SpatialPolygonsDataFrame(y, data.frame(IDS=1:nrow(y), py=runif(nrow(y))))

# Returns point attributes with column for each unique polygon
pts.poly <- point.in.poly(x, y, duplicate = FALSE)
  head(pts.poly@data)

# Duplicates points for each new polygon, joins all attributes
pts.poly.dup <- point.in.poly(x, y)
  head(pts.poly.dup@data)

# Count points in each polygon
tapply(pts.poly.dup$IDS.x, pts.poly.dup$IDS.y, FUN=length)

---

### poly.regression

**Local Polynomial Regression**

#### Description

Calculates a Local Polynomial Regression for smoothing or imputation of missing data.

This is a wrapper function for loess that simplifies data smoothing and imputation of missing values. The function allows for smoothing a vector, based on an index (derived automatically) or covariates. If the impute option is TRUE NA values are imputed, otherwise the returned vector will still have NA's present. If impute and na.only are both TRUE the vector is returned, without being smoothed but with imputed NA values filled in. The loess weight function is defined using the tri-cube weight function \( w(x) = (1-|x|^3)^3 \) where; \( x \) is the distance of a data point from the point the curve being fitted.

#### Usage

```r
poly.regression(
  y,
  x = NULL,
  s = 0.75,
  impute = FALSE,
  na.only = FALSE,
  ci = FALSE,
  ...
)
```
Arguments

y Vector to smooth or impute NA values
x Optional x covariate data (must match dimensions of y)
s Smoothing parameter (larger equates to more smoothing)
impute (FALSE/TRUE) Should NA values be imputed
na.only (FALSE/TRUE) Should only NA values be change in y
ci (FALSE/TRUE) Should confidence intervals be returned
... Additional arguments passed to loess

Value

If ci = FALSE, a vector of smoothed values, otherwise a list object with:

- loess - A vector, same length of y, representing the smoothed or imputed data
- lower.ci - Lower confidence interval
- upper.ci - Upper confidence interval

Author(s)

Jeffrey S. Evans jeffrey_evans@tnc.org

See Also

loess for loess ... model options

Examples

```r
x <- seq(-20, 20, 0.1)
y <- sin(x)/x + rnorm(length(x), sd=0.03)
p <- which(y == "NaN")
y <- y[-p]
r <- poly.regression(y, ci=TRUE, s=0.30)
plot(y,type="l", lwd=0.5, main="s = 0.10")
y.polygon <- c((r$lower.ci)[1:length(y)], (r$upper.ci)[rev(1:length(y))])
x.polygon <- c(1:length(y), rev(1:length(y)))
polygon(x.polygon, y.polygon, col="#00009933", border=NA)
lines(r$loess, lwd=1.5, col="red")

# Impute NA values, replacing only NA's
y.na <- y
y.na[c(100,200,300)] <- NA
p.y <- poly.regression(y.na, s=0.10, impute = TRUE, na.only = TRUE)
y - p.y
plot(p.y,type="l", lwd=1.5, col="blue", main="s = 0.10")
lines(y, lwd=1.5, col="red")
```
Description

Fast method for extracting raster values to polygons

This method for raster extraction uses the raster cell indices and is quite a bit faster with polygon data than other methods. This is especially true with large raster stacks (e.g., time-series). The cell indices are returned using `cellnumbers`. If ids argument is provided a column with values from the associate column are included otherwise "row_names" is returned which corresponds to the rownames in the source polygon object. Please note that if use.terra = TRUE it will coerce to a terra class if not already. With large data the coercion overhead may be worth it, providing speed gains. If the raster is a terra SpatRaster it will operate in its native class. The cells = TRUE argument will return the cell indices which could be used at a later time.

Usage

```r
polygon_extract(
  r,
  p,
  ids = NULL,
  cells = FALSE,
  asList = TRUE,
  use.terra = FALSE
)
```

Arguments

- `r` RasterLayer, RasterStack, RasterBrick or SpatRaster object
- `p` sf polygon data
- `ids` A unique id field contained in p, will be assigned to output otherwise will return rownames
- `cells` FALSE | TRUE - Return cell index ids
- `asList` TRUE | FALSE - Output list object
- `use.terra` FALSE | TRUE - Use terra for extracting indices

Value

A list object containing a data.frame for each polygons raster values, as columns. Additional columns are "row_names" or which ever column is passed to the ids argument and "cells" if cells = TRUE. If asList = FALSE a data.frame will be returned

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>
library(sf)
library(raster)

nc <- sf::st_read(system.file("shape/nc.shp", package="sf"))
nc <- sf::st_cast(nc, "POLYGON")

### multi-band
i=100; j=100
r <- do.call(raster::stack, replicate(20,
    raster::raster(matrix(runif(i*j), i, j))
    names(r) <- paste0("time", 1:nlayers(r))
    extent(r) <- extent(nc)
    proj4string(r) <- st_crs(nc)$proj4string
)

plot(r[[1]])
plot(st_geometry(nc), add=TRUE)

( e <- polygon_extract(r, nc) )
( e <- polygon_extract(r, nc, ids="CNTY_ID") )

# Column means
lapply(e, function(x) apply(x[,2:ncol(x)], MARGIN=1, FUN=mean) )

### Single band mean
( e <- polygon_extract(r[[1]], nc, ids="CNTY_ID") )
unlist(lapply(e, function(x) mean(x[,2], na.rm=TRUE) ))

# Leveraging cell ids, pulls values, calculates
# new value, and assigns to source cell using
# index from cells = TRUE

e <- polygon_extract(r, nc, cells=TRUE)
e <- data.frame(
    cells=unlist(lapply(e, function(x) as.numeric(x[,22]))),
    means=unlist(lapply(e, function(x) apply(x[,2:22], MARGIN=1, FUN=mean))
)

# copy raster and assign NA's
r2 <- r[[1]]
r2[] <- rep(NA, ncell(r))

# assign using cell indices
r2[e$cells] <- e$means
plot(r2)

# benchmark against raster extract
system.time(
    e <- polygon_extract(r, nc)
)
system.time({
  e <- raster::extract(r, nc)
})

---

**polyPerimeter**  
*Polygon perimeter*

---

**Description**

Calculates the perimeter length(s) for a polygon object

**Usage**

polyPerimeter(x)

**Arguments**

- **x**  
  sp class SpatialPolygonsDataFrame object

**Value**

A vector of polygon perimeters

**Author(s)**

Jeffrey S. Evans <jeffrey_evans@tnc.org>

**Examples**

```r
library(sp)
p1 <- Polygons(list(Polygon(cbind(c(2,4,4,1,2),c(2,3,5,4,2)))), "1")
p2 <- Polygons(list(Polygon(cbind(c(5,4,2,5),c(2,3,2,2)))), "2")
p3 <- Polygons(list(Polygon(cbind(c(4,4,5,10,4),c(5,3,2,5,5)))), "3")
polys <- SpatialPolygons(list(p1,p2,p3), 1:3)
polyPerimeter(polys)
```
pp.subsample

Point process random subsample

Description

Generates random subsample based on density estimate of observations.

The window type creates a convex hull by default or, optionally, uses the maximum extent (envelope). The resulting bandwidth can vary widely by method. The 'diggle' method is intended for bandwidth representing 2nd order spatial variation whereas the 'scott' method will represent 1st order trend. The 'geometry' approach will also represent 1st order trend. For large datasets, caution should be used with the 2nd order 'likelihood' approach, as it is slow and computationally expensive. Finally, the 'stoyan' method will produce very strong 2nd order results.

Usage

pp.subsample(
  x, 
  n, 
  window = "hull", 
  sigma = "Scott", 
  wts = NULL, 
  gradient = 1, 
  edge = FALSE 
)

Arguments

- **x**: An sp class SpatialPointsDataFrame or SpatialPoints object.
- **n**: Number of random samples to generate.
- **window**: Type of window (hull or extent).
- **sigma**: Bandwidth selection method for KDE, default is 'Scott'. Options are 'Scott', 'Stoyan', 'Diggle', 'likelihood', and 'geometry'.
- **wts**: Optional vector of weights corresponding to point pattern.
- **gradient**: A scaling factor applied to the sigma parameter used to adjust the gradient decent of the density estimate. The default is 1, for no adjustment (downweight < 1 | upweight > 1).
- **edge**: Apply Diggle edge correction (TRUE/FALSE).

Value

sp class SpatialPointsDataFrame containing random subsamples.
Note

Available bandwidth selection methods are:

- Scott - (Scott 1992), Scott’s Rule for Bandwidth Selection (1st order)
- Diggle - (Berman & Diggle 1989), Minimise the mean-square error via cross validation (2nd order)
- likelihood - (Loader 1999), Maximum likelihood cross validation (2nd order)
- geometry - Bandwidth is based on simple window geometry (1st order)
- Stoyan - (Stoyan & Stoyan 1995), Based on pair-correlation function (strong 2nd order)
- User defined - using a numeric value for sigma

Author(s)

Jeffrey S. Evans jeffrey_evans@tnc.org

References


Examples

```r
require(spatstat.core)
require(sp)
data(bei)
trees <- as(bei, 'SpatialPoints')
n=round(length(trees) * 0.10, digits=0)
trees.wrs <- pp.subsample(trees, n=n, window='hull')
plot(trees, pch=19, col='black')
plot(trees.wrs, pch=19, col='red', add=TRUE)
box()
title('10% subsample')
legend('bottomright', legend=c('Original sample', 'Subsample'),
       col=c('black','red'),pch=c(19,19))
```
print.cross.cor  
Print spatial cross correlation

Description

print method for class "cross.cor"

Usage

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

Arguments

x  Object of class cross.cor
...
Ignored

print.effect.size  
Print effect size

Description

print method for class "effect.size"

Usage

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

Arguments

x  Object of class effect.size
...
Ignored
print.loess.boot

Print Loess bootstrap model

Description

print method for class "loess.boot"

Usage

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

Arguments

x Object of class loess.boot

... Ignored

proximity.index

Proximity Index

Description

Calculates proximity index for a set of polygons

Usage

proximity.index(x, y = NULL, min.dist = 0, max.dist = 1000, background = NULL)

Arguments

x A polygon class sp or sf object

y Optional column in data containing classes

min.dist Minimum threshold distance

max.dist Maximum neighbor distance

background Optional value in y column indicating background value

Value

A vector equal to nrow(x) of proximity index values, if a background value is specified NA values will be returned in the position(s) of the specified class

Author(s)

Jeffrey S. Evans <jeffrey.evans@tnc.org>
pseudo.absence

References


Examples

library(sp)
library(rgeos)

# Create test polygons
data(meuse)
coordinates(meuse) = ~x+y
meuse_poly <- gBuffer(meuse, width = meuse$elev * 5, byid = TRUE)
meuse_poly$LU <- sample(c("forest", "nonforest"), nrow(meuse_poly), replace=TRUE)

# All polygon proximity index 1000 radius
(pidx <- proximity.index(meuse_poly, min.dist = 1))
pidx[pidx > 100] <- 100

# Class-level proximity index 1000 radius
(pidx.class <- proximity.index(meuse_poly, y = "LU", min.dist = 1))
pidx.class[pidx.class > 100] <- 100

# plot index for all polygons
meuse_poly$pidx <- pidx
spplot(meuse_poly, "pidx")

# plot index for class-level polygons
meuse_poly$cpidx <- pidx.class
spplot(meuse_poly, "cpidx")

# plot index for just forest class
forest <- meuse_poly[meuse_poly$LU == "forest",]
spplot(forest, "cpidx")

pseudo.absence

Pseudo-absence random samples

Description

Generates pseudo-absence samples based on density estimate of known locations

Usage

pseudo.absence(
x,
  n,
pseudo.absence

```r
window = "hull",
Mask = NULL,
s = NULL,
sigma = "Scott",
wts = NULL,
KDE = FALSE,
gradient = 1,
p = NULL,
edge = FALSE
)
```

### Arguments

- **x**: An sp class SpatialPointsDataFrame or SpatialPoints object
- **n**: Number of random samples to generate
- **window**: Type of window (hull OR extent), overridden if mask provided
- **Mask**: Optional rasterLayer class mask raster. The resolution of the density estimate will match mask.
- **s**: Optional resolution passed to window argument. Caution should be used due to long processing times associated with high resolution. In contrast, coarse resolution can exclude known points.
- **sigma**: Bandwidth selection method for KDE, default is 'Scott'. Options are 'Scott', 'Stoyan', 'Diggle', 'likelihood', and 'geometry'
- **wts**: Optional vector of weights corresponding to point pattern
- **KDE**: save KDE raster (TRUE/FALSE)
- **gradient**: A scaling factor applied to the sigma parameter used to adjust the gradient decent of the density estimate. The default is 1, for no adjustment (downweight < 1 | upweight > 1)
- **p**: Minimum value for probability distribution (must be > 0)
- **edge**: Apply Diggle edge correction (TRUE/FALSE)

### Details

The window type creates a convex hull by default or, optionally, uses the maximum extent (envelope). If a mask is provided the kde will represent areas defined by the mask and defines the area that pseudo absence data will be generated.

Available bandwidth selection methods are:

- Scott (Scott 1992), Scott’s Rule for Bandwidth Selection (1st order)
- Diggle (Berman & Diggle 1989), Minimize the mean-square error via cross validation (2nd order)
- likelihood (Loader 1999), Maximum likelihood cross validation (2nd order)
- geometry, Bandwidth is based on simple window geometry (1st order)
- Stoyan (Stoyan & Stoyan 1995), Based on pair-correlation function (strong 2nd order)
• User defined numeric distance bandwidth

Note; resulting bandwidth can vary widely by method. the ‘diggle’ method is intended for select-
ing bandwidth representing 2nd order spatial variation whereas the ‘scott’ method will represent
1st order trend. the ‘geometry’ approach will also represent 1st order trend. For large datasets,
caution should be used with the 2nd order ‘likelihood’ approach, as it is slow and computationally
expensive. finally, the ‘stoyan’ method will produce very strong 2nd order results.

Value

A list class object with the following components:

• sample SpatialPointsDataFrame containing random samples
• kde sp RasterLayer class of KDE estimates (IF KDE = TRUE)
• sigma Selected bandwidth of KDE

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


Hengl, T., H. Sierdsema, A. Radovic, and A. Dilo (2009) Spatial prediction of species distributions
from occurrence-only records: combining point pattern analysis, ENFA and regression-kriging.
Ecological Modelling, 220(24):3499-3511


Wiley.

statistics. John Wiley and Sons.


Examples

library(raster)
library(sp)
data(meuse)
data(meuse.grid)
coordinates(meuse) = ~x+y
coordinates(meuse.grid) = ~x+y
proj4string(meuse.grid) <- CRS("+init=epsg:28992")
gridded(meuse.grid) = TRUE
r <- raster(meuse.grid)
# Using a raster mask
pa <- pseudo.absence(meuse, n=100, window='hull', KDE=TRUE, Mask = r, 
  sigma='Diggle', s=50)
col.br <- colorRampPalette(c('blue','yellow'))
plot(pa$kde, col=col.br(10))
  plot(meuse, pch=20, cex=1, add=TRUE)
  plot(pa$sampling, col='red', pch=20, cex=1, add=TRUE)
  legend('top', legend=c('Presence', 'Pseudo-absence'), 
    pch=c(20,20),col=c('black','red'))

# With clustered data
library(sp)
library(spatstat.core)
data(bei)
trees <- as(bei, 'SpatialPoints')
trees <- SpatialPointsDataFrame(coordinates(trees), 
  data.frame(ID=1:length(trees)))
trees.abs <- pseudo.absence(trees, n=100, window='extent', KDE=TRUE)
col.br <- colorRampPalette(c('blue','yellow'))
plot(trees.abs$kde, col=col.br(10))
plot(trees, pch=20, cex=0.5, add=TRUE)
plot(trees.abs$sampling, col='red', pch=20, cex=1, add=TRUE)
legend('top', legend=c('Presence', 'Pseudo-absence'), 
  pch=c(20,20),col=c('black','red'))

---

**Biodiversity Planning Units**

**Description**
Subset of biodiversity planning units for Haiti ecoregional spatial reserve plan

**Format**
A sp SpatialPolygonsDataFrame with 5919 rows and 46 variables:

- **UNIT_ID** Unique planning unit ID
- **DR_Dr_A** Biodiversity target
- **DR_Dr_L** Biodiversity target
- **Ht_Dr_A** Biodiversity target
- **Ht_Dr_L** Biodiversity target
- **DR_Ms_A** Biodiversity target
- **DR_Ms_L** Biodiversity target
- **Ht_Ms_L** Biodiversity target
DR_LM_M  Biodiversity target
H_LM_M_L  Biodiversity target
H_LM_R_L  Biodiversity target
DR_LM_R_L  Biodiversity target
DR_Rn_L  Biodiversity target
DR_LM_R_S  Biodiversity target
DR_Rn_S  Biodiversity target
DR_Ms_S  Biodiversity target
Ht_Ms_A  Biodiversity target
DR_Ms_E  Biodiversity target
DR_Ms_I  Biodiversity target
DR_Rn_E  Biodiversity target
DR_Rn_I  Biodiversity target
H_LM_R_E  Biodiversity target
Ht_Ms_E  Biodiversity target
Ht_Rn_E  Biodiversity target
DR_Rn_A  Biodiversity target
Ht_Rn_A  Biodiversity target
Ht_Rn_I  Biodiversity target
Ht_Dr_E  Biodiversity target
Ht_Ms_S  Biodiversity target
Ht_Dr_S  Biodiversity target
Ht_Rn_L  Biodiversity target
Ht_Th_A  Biodiversity target
Ht_Th_L  Biodiversity target
Ht_Th_S  Biodiversity target
Ht_Dr_U  Biodiversity target
Ht_Dr_I  Biodiversity target
Ht_Ms_I  Biodiversity target
H_LM_M_A  Biodiversity target
H_LM_M_E  Biodiversity target
H_LM_R_A  Biodiversity target
H_LM_M_S  Biodiversity target
H_LM_R_I  Biodiversity target
H_LM_R_S  Biodiversity target
Ht_Rn_S  Biodiversity target
Ht_Ms_U  Biodiversity target
Ht_Rn_U  Biodiversity target
quadrats

Source

https://www.conservationgateway.org

References


quadrats

Description

Creates quadrat polygons for sampling or analysis

Usage

quadrats(x, s = 250, n = 100, r = NULL)

Arguments

x A sp or sf polygon object defining extent
s Radius defining single or range of sizes of quadrats
n Number of quadrats
r A rotation factor for random rotation, default is NULL

Value

an sp or sf polygon object with rotated polygon

Note

The radius (s) parameter can be a single value or a range of values, representing a randomization range of resulting quadrat sizes. The rotation (r) parameter can also be used to defined a fixed rotation or random range of quadrat rotations. You can specify each of these parameters using an explicit vector that will be sampled eg., seq(100,300,0.5)
Examples

```r
library(sp)
library(raster)
library(rgeos)
data(meuse)
coordinates(meuse) <- ~x+y
e <- gConvexHull(meuse)

# Fixed size 250 and no rotation
s <- quadrats(e, s = 250, n = 50)
spplot(s, "ID")

# Variable sizes 100-300 and rotation of 0-45 degrees
s <- quadrats(e, s = c(100,300), n = 50, r = c(0,45))
spplot(s, "ID")

# Variable sizes 100-300 and no rotation
s <- quadrats(e, s = c(100,300), n = 50)
spplot(s, "ID")
```

random.raster

Description

Create a random raster or raster stack using specified distribution

Usage

```r
random.raster(
  r = NULL,
  n.row = 50,
  n.col = 50,
  n.layers = 1,
  x = seq(1, 10),
  min = 0,
  max = 1,
  mean = 0,
  sd = 1,
  p = 0.5,
  s = 1.5,
  distribution = c("random", "normal", "seq", "binomial", "gaussian")
)
```

Arguments

- **r** Optional existing raster defining nrow/ncol
**random.raster**

- **n.row**  Number of rows
- **n.col**  Number of columns
- **n.layers**  Number of layers in resulting raster stack
- **x**  A vector of values to sample if distribution is "sample"
- **min**  Minimum value of raster
- **max**  Maximum value of raster
- **mean**  Mean of centered distribution
- **sd**  Standard deviation of centered distribution
- **p**  p-value for binomial distribution
- **s**  sigma value for Gaussian distribution
- **distribution**  Available distributions, c("random", "normal", "seq", "binomial", "gaussian", "sample")

**Details**

Options for distributions are for random, normal, seq, binomial, gaussian and sample raster(s)

**Value**

RasterLayer or RasterStack object with random rasters

**Author(s)**

Jeffrey S. Evans <jeffrey_evans@tnc.org>

**Examples**

```r
library(raster)

# Using existing raster to create random binomial
r <- raster(system.file("external/rlogo.grd", package="raster"))
r <- random.raster(r, distribution="binomial")

# default; random, nrows=50, ncols=50, nlayers=1
rr <- random.raster(n.layer=5)

# specified; binomial, nrows=20, ncols=20, nlayers=5
rr <- random.raster(n.layer=5, n.col=20, n.row=20,
        distribution="binomial")

# specified; gaussian, nrows=50, ncols=50, nlayers=1
rr <- random.raster(n.col=50, n.row=50, s=8,
        distribution="gaussian")

# specified; sample, nrows=50, ncols=50, nlayers=1
rr <- random.raster(n.layer=1, x=c(2,6,10,15), distribution="sample" )
freq(rr)
```
raster.change  

**Raster change between two nominal rasters**

### Description

Compares two categorical rasters with a variety of statistical options.

This function provides a variety of statistics for comparing two classified maps. Valid options are:

- kappa - Cohen’s Kappa
- wkappa - Cohen’s Weighted Kappa (not yet implemented)
- t.test - Two-tailed paired t-test
- cor - Persons Correlation
- entropy - Delta entropy
- cross-entropy - Cross-entropy loss function
- divergence - Kullback-Leibler divergence (relative entropy)

Kappa and t-test values < 0 are reported as 0. For a weighted kappa, a matrix must be provided that correspond to the pairwise weights for all values in both rasters. Delta entropy is derived by calculating Shannon’s on each focal window then differencing them \((e(x) - e(y))\).

### Usage

```r
raster.change(
  x, y,
  d = c(3, 3),
  stat = c("kappa", "wkappa", "t.test", "cor", "entropy", "cross-entropy",
          "divergence"),
  w = NULL,
  out.raster = NULL,
  mask = FALSE,
  force.memory = FALSE
)
```

### Arguments

- **x**  
  First raster for comparison, rasterLayer class object

- **y**  
  Second raster for comparison, rasterLayer class object

- **d**  
  Rectangular window size, must be odd but not necessarily square

- **stat**  
  Statistic to use in comparison, please see details for options.

- **w**  
  Weights if stat="kappa", must represent same classes as input rasters

- **out.raster**  
  Optional output raster

- **mask**  
  (FALSE/TRUE) mask output to original rasters

- **force.memory**  
  (FALSE/TRUE) Force in memory processing, may fail with insufficient RAM
Value

A raster layer or stack object one of the following layers:

- kappa Kappa or Weighted Kappa statistic (if stat = "kappa")
- correlation Paired t.test statistic (if stat = "cor")
- entropy Delta entropy (if stat = "entropy")
- divergence Kullback-Leibler divergence (if stat = "divergence")
- cross.entropy Cross-entropy (if stat = "cross.entropy")
- t.test Paired t.test statistic (if stat = "t.test")
- p.value p-value of the paired t.test statistic (if stat = "t.test")

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


Examples

```r
library(sp)
library(raster)
data(meuse.grid)

r1 <- sp::SpatialPixelsDataFrame(points = meuse.grid[,c("x", "y")],
                                  data = meuse.grid)

r1 <- raster(r1)
na.idx <- which(!is.na(r1[]))
na.idx <- round(runif(length(na.idx), 1,5),0)

r2 <- sp::SpatialPixelsDataFrame(points = meuse.grid[,c("x", "y")],
                                  data = meuse.grid)

r2 <- raster(r2)
na.idx <- round(runif(length(na.idx), 1,5),0)

s = 11
(r.kappa <- raster.change(r1, r2, d = s, mask = TRUE) )
(r.ttest <- raster.change(r1, r2, d = s, stat="t.test", mask = TRUE) )
(r.ent <- raster.change(r1, r2, d = s, stat="entropy", mask = TRUE) )
(r.cor <- raster.change(r1, r2, d = s, stat="cor", mask = TRUE) )
(r.ce <- raster.change(r1, r2, d = s, stat = "cross-entropy", mask = TRUE) )
(r.kl <- raster.change(r1, r2, d = s, stat = "divergence", mask = TRUE) )

opar <- par(no.readonly=TRUE)
```
raster.deviation

**Raster local deviation from the global trend**

**Description**

Calculates the local deviation from the raster, a specified global statistic or a polynomial trend of the raster.

The deviation from the trend is derived as \([y-hat - y]\), where; \(y-hat\) is the \(N\)th-order polynomial.

Whereas the deviation from a global statistic is \([y - y-hat]\), where; \(y-hat\) is the local (focal) statistic.

The `global = TRUE` argument allows one to evaluate the local deviation from the global statistic \([\text{stat}(x) - y-hat]\), where; \(\text{stat}(x)\) is the global value of the specified statistic and \(y-hat\) is the specified focal statistic.

**Usage**

```r
raster.deviation(x, type = "trend", s = 3, degree = 1, global = FALSE)
```

**Arguments**

- **x**
  - raster object

- **type**
  - The global statistic to represent the local deviation options are: "trend", "min", "max", "mean", "median"

- **s**
  - Size of matrix (focal window), not used with type="trend"

- **degree**
  - The polynomial degree if type is trend, options are 1 and 2.

- **global**
  - Use single global value for deviation or cell-level values (FALSE/TRUE). Argument is ignored for type="trend"

**Value**

raster class object of the local deviation from the raster or specified global statistic

**Author(s)**

Jeffrey S. Evans <jeffrey.evans@tnc.org>
References


Examples

library(raster)
data(elev)

# local deviation from first-order trend, global mean and raw value
r.dev.trend <- raster.deviation(elev, type="trend", degree=1)
r.dev.mean <- raster.deviation(elev, type="mean", s=5)
r.gdev.mean <- raster.deviation(elev, type="mean", s=5, global=TRUE)

opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
plot(elev, main="original")
plot(r.dev.trend, main="dev from trend")
plot(r.dev.mean, main="dev of mean from raw values")
plot(r.gdev.mean, main="local dev from global mean")
par(opar)

raster.downscale  Raster Downscale

Description

Downscales a raster to a higher resolution raster using a robust regression

Usage

raster.downscale(
  x,
  y,
  p = NULL,
  n = NULL,
  filename = FALSE,
  scatter = FALSE,
  ...
)

Arguments

- **x**: Raster class object representing independent variable(s)
- **y**: Raster class object representing dependent variable
- **p**: Percent sample size
- **n**: Fixed sample size
- **filename**: Name of output raster
- **scatter**: (FALSE/TRUE) Optional scatter plot
- **...**: Additional arguments passed to predict

Value

A list object containing:

- downscaled raster (omitted if filename is defined)
- model rlm model object
- MSE Mean Square Error
- AIC Akaike information criterion

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

```r
## Not run:
library(raster)
elev <- raster::getData(quote(alt), country=quote(SWZ), mask=TRUE)
tmax <- raster::getData(quote(worldclim), var=quote(tmax), res=10,
                        lon=8.25, lat=46.8)
tmax <- crop(tmax[[1]], extent(elev))
# Downscale temperature
tmax.ds <- raster.downscale(elev, tmax, scatter=TRUE)
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
plot(tmax, main="Temp max")
plot(elev, main="elevation")
plot(tmax.ds$downscale, main="Downscaled Temp max")
par(opar)

## End(Not run)
```
**raster.entropy**  

**Raster Entropy**

**Description**

Calculates entropy on integer raster (i.e., 8 bit 0-255)

Entropy calculated as:  \( H = -\sum(P_i \ln(P_i)) \) where:  
- \( P_i \), Proportion of one value to total values  
- \( n(p)/m \) and \( m \), Number of unique values.  
Expected range: 0 to \( \log(m) \)  
H=0 if window contains the same value in all cells. H increases with the number of different values in the window.

Maximum entropy is reached when all values are different, same as \( \log(m) \)  
\[
\text{max.ent} \leftarrow \text{function(x)} \rightarrow \log(\text{length(unique(x)))}
\]

**Usage**

```r
raster.entropy(
  x,
  d = 5,
  categorical = FALSE,
  global = FALSE,
  filename = FALSE,
  ...
)
```

**Arguments**

- **x**  
  Object of class raster (requires integer raster)
- **d**  
  Size of matrix (window)
- **categorical**  
  Is the data categorical or continuous (FALSE/TRUE)
- **global**  
  Should the model use a global or local \( n \) to calculate entropy (FALSE/TRUE)
- **filename**  
  Raster file written to disk
- **...**  
  Optional arguments passed to writeRaster or dataType

**Value**

raster class object or specified format raster written to disk

**References**

Examples

```r
require(raster)
  r <- raster(ncols=100, nrows=100)
  r[] <- round(runif(ncell(r), 1,8), digits=0)

  rEnt <- raster.entropy(r, d=5, categorical = TRUE, global = TRUE)
  opar <- par(no.readonly=TRUE)
  par(mfcol=c(2,1))
  plot(r)
  plot(rEnt)
  par(opar)
```

---

### raster.gaussian.smooth

**Gaussian smoothing of raster**

**Description**

Applies a Gaussian smoothing kernel to smooth raster.

**Usage**

```r
raster.gaussian.smooth(x, sigma = 2, n = 5, type = mean, ...)
```

**Arguments**

- `x`: raster object
- `sigma`: standard deviation (sigma) of kernel (default is 2)
- `n`: Size of the focal matrix, single value (default is 5 for 5x5 window)
- `type`: The statistic to use in the smoothing operator (suggest mean or sd)
- `...`: Additional arguments passed to raster::focal

**Value**

raster class object of the local distributional moment

**Note**

This is a simple wrapper for the focal function, returning local statistical moments

**Author(s)**

Jeffrey S. Evans <jeffrey_evans@tnc.org>
Examples

```r
library(raster)
r <- raster(nrows=500, ncols=500, xmn=571823, xmx=616763,
            ymn=4423540, ymx=4453690)
proj4string(r) <- CRS("+proj=utm +zone=12 +datum=NAD83 +units=m +no_defs")
r[] <- runif(ncell(r), 1000, 2500)
r <- focal(r, focalWeight(r, 150, "Gauss") )

# Calculate Gaussian smoothing with sigma(s) = 1-4
g1 <- raster.gaussian.smooth(r, sigma=1, nc=11)
g2 <- raster.gaussian.smooth(r, sigma=2, nc=11)
g3 <- raster.gaussian.smooth(r, sigma=3, nc=11)
g4 <- raster.gaussian.smooth(r, sigma=4, nc=11)

opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
plot(g1, main="Gaussian smoothing sigma = 1")
plot(g2, main="Gaussian smoothing sigma = 2")
plot(g3, main="Gaussian smoothing sigma = 3")
plot(g4, main="Gaussian smoothing sigma = 4")
par(opar)
```

describe(raster.invert)

**Invert raster**

Inverts (flip) the values of a raster

**Usage**

```r
raster.invert(x)
```

**Arguments**

- `x` : raster object

**Value**

raster class object with inverted (flipped) raster values

**Note**

Inverts raster values using the formula: `(((x - max(x)) * -1) + min(x)`

**Author(s)**

Jeffrey S. Evans <jeffrey_evans@tnc.org>
Examples

```r
library(raster)
r <- raster(nrows=500, ncols=500, xmn=571823, xmx=616763,
            ymn=4423540, ymx=4453690)
r[] <- runif(ncell(r), 1, 100)
r <- focal(r, focalWeight(r, 150, "Gauss")

r.inv <- raster.invert(r)

opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))
plot(r, main="original raster")
plot(r.inv, main="inverted raster")
par(opar)
```

---

`raster.kendall`  
*Kendall tau trend with continuity correction for raster time-series*

Description

Calculates a nonparametric statistic for a monotonic trend based on the Kendall tau statistic and the Theil-Sen slope modification.

Usage

```r
raster.kendall(x, intercept = FALSE, p.value = FALSE, z.value = FALSE, confidence = FALSE, tau = FALSE, ...)
```

Arguments

- `x` A rasterStack object with at least 5 layers
- `intercept` (FALSE/TRUE) return a raster with the pixel wise intercept values
- `p.value` (FALSE/TRUE) return a raster with the pixel wise p.values
- `z.value` (FALSE/TRUE) return a raster with the pixel wise z.values
- `confidence` (FALSE/TRUE) return a raster with the pixel wise 95 pct confidence levels
- `tau` (FALSE/TRUE) return a raster with the pixel wise tau correlation values
- `...` Additional arguments passed to the raster overlay function
Details

This function implements Kendall’s nonparametric test for a monotonic trend using the Theil-Sen (Theil 1950; Sen 1968; Siegel 1982) method to estimate the slope and related confidence intervals.

Value

Depending on arguments, a raster layer or rasterBrick object containing:

- raster layer 1 slope for trend, always returned
- raster layer 2 intercept for trend if intercept TRUE
- raster layer 3 p value for trend fit if p.value TRUE
- raster layer 4 z value for trend fit if z.value TRUE
- raster layer 5 lower confidence level at 95 pct, if confidence TRUE
- raster layer 6 upper confidence level at 95 pct, if confidence TRUE
- raster layer 7 Kendall’s tau two-sided test, reject null at 0, if tau TRUE

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


See Also

kendallTrendTest for model details
overlay for available ... arguments

Examples

library(raster)
r.logo <- stack(system.file("external/rlogo.grd", package="raster"),
    system.file("external/rlogo.grd", package="raster"),
    system.file("external/rlogo.grd", package="raster"))

# Calculate trend slope with p-value and confidence level(s)
# ("slope","intercept", "p.value","z.value", "LCI","UCI","tau")
k <- raster.kendall(r.logo, p.value=TRUE, z.value=TRUE,
    intercept=TRUE, confidence=TRUE,
    tau=TRUE)

plot(k)
raster.mds  

**Raster multidimensional scaling (MDS)**

**Description**

Multidimensional scaling of raster values within an N x N focal window

An MDS focal function. If only one value provided for \( s \), then a square matrix (window) will be used. If window.median = FALSE then the center value of the matrix is returned and not the median of the matrix.

**Usage**

```r
raster.mds(r, s = 5, window.median = FALSE, ...)
```

**Arguments**

- **r**  
  Raster layer

- **s**  
  Window size (may be a vector of 1 or 2) of n x n dimension.

- **window.median**  
  (TRUE/FALSE) Return the median of the MDS matrix values.

- **...**  
  Additional arguments passed to raster::focal

**Value**

A raster class object or raster written to disk

**Author(s)**

Jeffrey S. Evans <jeffrey_evans@tnc.org>

**References**


**Examples**

```r
library(raster)
r <- raster(system.file("external/rlogo.grd", package="raster"))
r <- r / cellStats(r, "max")

diss <- raster.mds(r)
diss.med <- raster.mds(r, window.median = TRUE)

opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
plot(r)
```
raster.modified.ttest  Dutilleul moving window bivariate raster correlation

Description

A bivariate raster correlation using Dutilleul’s modified t-test

This function provides a bivariate moving window correlation using the modified t-test to account for spatial autocorrelation. Point based subsampling is provided for computation tractability. The hexagon sampling is recommended as it is good at capturing spatial process that includes nonstationarity and anisotropy.

Usage

raster.modified.ttest(
  x,
  y,
  x.idx = 1,
  y.idx = 1,
  d = "AUTO",
  sub.sample = FALSE,
  type = "hexagon",
  p = 0.1,
  size = NULL
)

Arguments

x  x raster for correlation, SpatialPixelsDataFrame or SpatialGridDataFrame object
y  y raster for correlation, SpatialPixelsDataFrame or SpatialGridDataFrame object
x.idx  Index for the column in the x raster object
y.idx  Index for the column in the y raster object
d  Distance for finding neighbors
sub.sample  Should a sub-sampling approach be employed (TRUE/FALSE)
type  If sub.sample = TRUE, what type of sample (random or hexagon)
p  If sub.sample = TRUE, what proportion of population should be sampled
size  Fixed sample size
Value

A SpatialPixelsDataFrame or SpatialPointsDataFrame with the following attributes:

- corr Correlation
- Fstat The F-statistic calculated as [degrees of freedom * unscaled F-statistic]
- p.value p-value for the test
- moran.x Moran’s-I for x
- moran.y Moran’s-I for y

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


See Also

modified.ttest for test details

Examples

```r
## Not run:
library(gstat)
library(sp)

data(meuse)
data(meuse.grid)
coordinates(meuse) <- ~x + y
coordinates(meuse.grid) <- ~x + y

# GRID-1 log(copper):
v1 <- variogram(log(copper) ~ 1, meuse)
x1 <- fit.variogram(v1, vgm(1, "Sph", 800, 1))
G1 <- krige(zinc ~ 1, meuse, meuse.grid, x1, nmax = 30)
gridded(G1) <- TRUE
G1@data = as.data.frame(G1@data[, -2])

# GRID-2 log(elev):
v2 <- variogram(elev ~ 1, meuse)
x2 <- fit.variogram(v2, vgm(.1, "Sph", 1000, .6))
G2 <- krige(elev ~ 1, meuse, meuse.grid, x2, nmax = 30)
gridded(G2) <- TRUE
G2@data <- as.data.frame(G2@data[, -2])
G2@data[, 1] <- G2@data[, 1]
```
corr <- raster::modified.ttest(G1, G2)
    plot(raster::raster(corr,1))

corr.rand <- raster::modified.ttest(G1, G2, sub.sample = TRUE, type = "random")
corr.hex <- raster::modified.ttest(G1, G2, sub.sample = TRUE, d = 500, size = 1000)
    head(corr.hex@data)
    bubble(corr.hex, "corr")

## End(Not run)

---

**raster.moments**  
**Raster moments**

**Description**
Calculates focal statistical moments of a raster

**Usage**
raster.moments(x, type = "mean", s = 3, p = 0.75)

**Arguments**
- **x**: raster object
- **type**: The global statistic to represent the local deviation options are: "min", "min", "mean", "median", "var", "sd", "mad", "kurt", "skew", "quantile"
- **s**: Size of matrix (focal window), can be single value or two values defining the [x,y] dimensions of the focal matrix
- **p**: if type="quantile", the returned percentile.

**Value**
raster class object of the local distributional moment

**Note**
This is a simple wrapper for the focal function, returning local statistical moments

**Author(s)**
Jeffrey S. Evans <jeffrey_evans@tnc.org>
raster.transformation

Examples

library(raster)

r <- raster(nrows=100, ncols=100, xmn=571823, xmx=616763,
            ymn=4423540, ymx=4453690)

proj4string(r) <- CRS("+proj=utm +zone=12 +datum=NAD83 +units=m +no_defs")

r[] <- runif(ncell(r), 1000, 2500)

r <- focal(r, focalWeight(r, 150, "Gauss") )

# Calculate 10th percentile for 3x3 window

r.p10 <- raster.moments(r, type="quantile", p=0.10)

raster.transformation

Statistical transformation for rasters

Description

Transforms raster to a specified statistical transformation

Transformation option details:

- norm - (Normalization (0-1): if min(x) < 0 \( \frac{(x - \text{min}(x))}{(\text{max}(x) - \text{min}(x))} \) 
- rstd - (Row standardize) (0-1): if min(x) >= 0 \( \frac{x}{\text{max}(x)} \) This normalizes data with negative distributions
- std - (Standardize) \( \frac{(x - \text{mean}(x))}{\text{sdv}(x)} \)
- stretch - (Stretch) \( \frac{(x - \text{min}(x)) \times \text{max.stretch}}{(\text{max}(x) - \text{min}(x)) + \text{min.stretch}} \) This will stretch values to the specified minimum and maximum values (e.g., 0-255 for 8-bit)
- nl - (Natural logarithms) if min(x) > 0 \( \log(x) \)
- slog - (Signed log 10) (for skewed data): if min(x) >= 0 ifelse(abs(x) <= 1, 0, sign(x)*log10(abs(x)))
- sr - (Square-root) if min(x) >= 0 \( \sqrt{x} \)

Usage

raster.transformation(x, trans = "norm", smin = 0, smax = 255)

Arguments

x          raster class object
trans      Transformation method: "norm", "rstd", "std", "stretch", "nl", "slog", "sr" (please see notes)
smin      Minimum value for stretch
smax      Maximum value for stretch
raster.vol

Value

raster class object of transformation

Author(s)

Jeffrey S. Evans jeffrey_evans@tnc.org

Examples

library(raster)
  r <- raster(nrows = 100, ncols = 100, xmn = 571823, xmx = 616763,
             ymn = 4423540, ymx = 4453690)
  r[] <- runif(ncell(r), 1000, 2500)

  # Postive values so, can apply any transformation
  for( i in c("norm", "rstd", "std", "stretch", "nl", "slog", "sr")) {
    print( raster.transformation(r, trans = i) )
  }

  # Negative values so, can't transform using "nl", "slog" or "sr"
  r[] <- runif(ncell(r), -1, 1)
  for( i in c("norm", "rstd", "std", "stretch", "nl", "slog", "sr")) {
    try( print( raster.transformation(r, trans = i) ) )
  }

raster.vol

Raster Percent Volume

Description

Calculates a percent volume on a raster or based on a systematic sample

Usage

raster.vol(x, p = 0.95, sample = FALSE, spct = 0.05)

Arguments

  x          raster class object
  p          percent raster-value volume
  sample     base volume on systematic point sample (TRUE/FALSE)
  spct       sample percent, if sample (TRUE)
Value

if sample (FALSE) binary raster object with 1 representing designated percent volume else, if sample (TRUE) n sp SpatialPointsDataFram object with points that represent the percent volume of the sub-sample

Note

Since this model needs to operate on all of the raster values, it is not memory safe

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

```r
require(raster)
  r <- raster(ncols=100, nrows=100)
  r[] <- runif(ncell(r), 0, 1)
  r <- focal(r, w=focalWeight(r, 6, "Gauss"))
  r[sample(1000, 1:ncell(r))] <- NA

  # full raster percent volume
  p30 <- raster.vol(r, p=0.30)
  p50 <- raster.vol(r, p=0.50)
  p80 <- raster.vol(r, p=0.80)

  opar <- par(no.readonly=TRUE)
  par(mfrow=c(2,2))
  plot(r, col=cm.colors(10), main="original raster")
  plot(p30, breaks=c(0,0.1,1), col=c("cyan","red"), legend=FALSE, main="30% volume")
  plot(p50, breaks=c(0,0.1,1), col=c("cyan","red"), legend=FALSE, main="50% volume")
  plot(p80, breaks=c(0,0.1,1), col=c("cyan","red"), legend=FALSE, main="80% volume")
  par(opar)
```

---

**raster.Zscore**

*Modified z-score for a raster*

Description

Calculates the modified z-score for all cells in a raster

Usage

```r
raster.Zscore(x, p.value = FALSE, file.name = NULL, ...)
```
Arguments

- **x**: A raster class object
- **p.value**: Return p-value rather than z-score raster (FALSE/TRUE)
- **file.name**: Name of raster written to disk
- **...**: Additional arguments passed to writeRaster

Value

raster class object or raster written to disk

Note

Since this functions needs to operate on all of the raster values, it is not memory safe

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

```r
library(raster)
r <- raster(nrows=824, ncols=767, xmn=2451905, xmx=3218905,
           ymn=-2744771, ymx=-1920771, resolution = 5000)
r[] <- runif(ncell(r), 0, 1)
# Modified z-score
z <- raster.Zscore(r)
# P-value
p <- raster.Zscore(r, p.value = TRUE)
```

---

**rasterCorrelation**  
*Raster correlation*

Description

Perform a simple moving window correlation between two rasters

Usage

```r
rasterCorrelation(x, y, s = 3, type = "pearson", file.name = NULL, ...)
```
Arguments

- **x**: raster class object for \( x \)
- **y**: raster class object for \( y \)
- **s**: Scale of window. Can be a single value, two values for uneven window or a custom matrix. Must be odd number (eg., \( s=3 \), for 3x3 window or \( s=c(3,5) \) for 3 x 5 window)
- **type**: Type of output, options are: "pearson", "spearman",
- **file.name**: Name of output raster (optional)
- **...**: Additional arguments passed to writeRaster

Value

raster class object or raster written to disk

Note

Depends: raster

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

```r
library(raster)
b <- brick(system.file("external/rlogo.grd", package="raster"))
x <- b[[1]]
y <- b[[3]]
r.cor <- rasterCorrelation(x, y, s = 5, type = "spearman")
plot(r.cor)
```

rasterDistance  

**Raster Distance**

Description

Calculates the Euclidean distance between a set of points and the cells in a raster. This is a drop-in replacement for the raster distanceFromPoints function using the RANN algorithm for calculating distance, resulting in a large improvement in processing speed.

Usage

```r
rasterDistance(x, y, reference = NULL, scale = FALSE)
```
**Arguments**

- **x** rasterLayer, sp SpatialPoints or sf POINTS object
- **y** sp SpatialPoints or sf POINTS object
- **reference** A raster to use as a reference if x is points object
- **scale** (FALSE/TRUE) Perform a row standardization on results

**Value**

A distance raster of class rasterLayer

**Note**

This replicates the raster distanceFromPoints function but uses the Arya & Mount Approximate Near Neighbor (ANN) C++ library for calculating distances. Where this results in a notable increase in performance it is not memory safe, needing to read in the entire raster and does not use the GeographicLib (Karney, 2013) spheroid distance method for geographic data.

**Author(s)**

Jeffrey S. Evans <jeffrey_evans@tnc.org>

**References**


**See Also**

distanceFromPoints, distance

**Examples**

```r
library(raster)

r <- raster(ncol=100,nrow=100)
r[] <- sample(c(0,1), ncell(r), replace = TRUE)

majority <- function(x){
m <- table(x)
names(m)[which.max(m)][1]
}
r <- focal(r, matrix(1,11,11, byrow=TRUE), majority)

pts <- rasterToPoints(r, spatial=TRUE)
cls <- pts[pts$layer == "1",]
d <- rasterDistance(pts, cls, reference = r, scale=TRUE)
dev.new(height=8,width=11)
plot(d)
points(cls,pch=19,cex=0.5)
```
**remove.holes**

**Remove polygon holes**

**Description**

Removes all holes (null geometry) in polygon sp class objects

**Usage**

```r
remove.holes(x)
```

**Arguments**

- `x`  
  SpatialPolygons or SpatialPolygonsDataFrame class object

**Value**

SpatialPolygonsDataFrame object with all holes removed

**Note**

A hole is considered a polygon within a polygon representing null geometry

**Author(s)**

Jeffrey S. Evans <jeffrey_evans@tnc.org>

**Examples**

```r
library(sp)
Sr1 = Polygon(cbind(c(2,4,4,1,2),c(2,3,5,4,2)))
Sr2 = Polygon(cbind(c(5,4,2,5),c(2,3,2,2)))
Sr3 = Polygon(cbind(c(4,4,5,10,4),c(5,3,2,5,5)))
Sr4 = Polygon(cbind(c(5,6,6,5,5),c(4,4,3,3,4)), hole = TRUE)
polys <- SpatialPolygons(list(Polygons(list(Sr1), "s1"),  
                        Polygons(list(Sr2), "s2"),  
                        Polygons(list(Sr3, Sr4), "s3/4")), 1:3)

opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))
plot(polys, col = 1:3, main="with hole")
plot(remove.holes(polys), col = 1:3, main="with hole removed")
par(opar)
```
**rm.ext**

*Remove extension*

**Description**

Removes file extension (and path) from string

**Usage**

```r
rm.ext(x)
```

**Arguments**

- `x` A character vector representing a file with extension

**Value**

The file name with extension and file path stripped off

**Examples**

```r
rm.ext("C:/path/file.txt")
```

---

**rotate.polygon**

*Rotate polygon*

**Description**

Rotates polygon by specified angle

**Usage**

```r
rotate.polygon(
  p,
  angle = 45,
  sp = FALSE,
  anchor = c("center", "lower.left", "upper.right")
)
```

**Arguments**

- `p` A polygon object of sf or sp class
- `angle` Rotation angle in degrees
- `sp` (FALSE | TRUE) Output sp class object
- `anchor` Location to rotate polygon on options are "center", "lower.left" and "upper.right"
Value

an sp or sf polygon object with rotated polygon

Note

The anchor is the location that the rotation is anchored to. The center is the centroid where the lower.left and upper.right are based on the min or max of the coordinates respectively.

Examples

```r
library(sp)
library(rgeos)

data(meuse)
coordinates(meuse) <- ~x+y

e <- gConvexHull(meuse)
e30 <- rotate.polygon(e, angle=30, sp=TRUE)

plot(e, main="rotated 30 degrees")
plot(e30, add=TRUE)
```

```
Description

The Trigonometric Stage (1978) \([\text{slope} \times \cos(\text{aspect})]\) or \([\text{slope} \times \sin(\text{aspect})]\)
An a priori assumption of a maximum in the NW quadrant (45 azimuth) and a minimum in the SW quadrant can be replaced by an empirically determined location of the optimum without repeated calculations of the regression fit. In addition it is argued that expressions for the effects of aspect should always be considered as terms involving an interaction with slope (Stage, 1976)
For slopes from 0 bounded from -1 to 1. Greater than 100 out of the -1 to 1 range.
An alternative for slopes with values approaching infinity is to take the square root of slope/100 to reduce the range of values. By default this model test all values greater than 100 to 101

Usage

```r
sa.trans(
  slope,
  aspect,
  type = "cos",
  slp.units = "degrees",
  asp.units = "degrees"
)
```
sample.annulus

Arguments

slope  slope values in degrees, radians or percent
aspect aspect values in degrees or radians
type  Type of transformation, options are: "cos", "sin"
slp.units Units of slope values, options are: "degrees", "radians" or "percent"
asp.units Units of aspect values, options are: "degrees" or "radians"

Value

A vector of the modeled value

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


Examples

sa.trans(slope = 48.146, aspect = 360.000)

library(raster)
data(elev)
sa <- raster::terrain(elev, opt=c("slope", "aspect"), unit="degrees")
scosa <- raster::overlay(sa[[1]], sa[[2]], fun = sa.trans)

______________________________
sample.annulus                  Sample annulus
______________________________

Description

Creates sample points based on annulus with defined inner and outer radius

Usage

sample.annulus(x, r1, r2, n = 10, ...)

Arguments

x  sp SpatialPoints or SpatialPointsDataFrame class object
r1 Numeric value defining inner radius of annulus (in projection units)
r2 Numeric value defining outer radius of annulus (in projection units)
n  Number of samples
... Additional arguments passed to spsample
Value

sp SpatialPointsDataFrame OBJECT

Note

Function can be used for distance based sampling. This is a sampling method that can be used to capture spatially lagged variation.

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

```r
library(sp)
library(rgeos)
data(meuse)
coordinates(meuse) <- ~x+y
proj4string(meuse) <- CRS("+init=epsg:28992")
xy <- meuse[,]

rs100 <- sample.annulus(xy, r1=50, r2=100, n = 50, type = "random")
rs200 <- sample.annulus(xy, r1=100, r2=200, n = 50, type = "random")

plot(rs200, pch=20, col="red")
points(rs100, pch=20, col="blue")
points(xy, pch=20, cex=2, col="black")
box()
legend("topright", legend=c("50-100m", "100-200m", "source"),
       pch=c(20,20,20), col=c("blue","red","black"))
```

---

**Systematic or random point sample of line(s)**

Description

Creates a systematic or random point sample of an sp SpatialLinesDataFrame object based on distance spacing, fixed size or proportional size.

The sdist argument will produce an evenly spaced sample, whereas n produces a fixed sized sample. The p (proportional) argument calculates the percent of the line-length. The LID column in the @data slot corresponds to the row.names of the SpatialLinesDataFrame object.
sample.line

Usage

```
sample.line(
  x,
  d = 100,
  p = NULL,
  n = NULL,
  type = "regular",
  longlat = FALSE,
  min.samp = 1,
  ...
)
```

Arguments

- `x`: sp class SpatialLinesDataFrame object
- `d`: Sample distance. For regular sample.
- `p`: Proportional sample size (length * p), expected value is 0-1. For regular or random.
- `n`: Fixed sample size. For regular or random
- `type`: Defines sample type. Options are "regular" or "random". A regular sample results in a systematic, evenly spaced sample.
- `longlat`: TRUE/FALSE is data in geographic units, if TRUE distance is in kilometers
- `min.samp`: Minimal number of sample points for a given line (default is 1 point)
- `...`: Additional argument passed to spsample

Value

sp SpatialPointsDataFrame object.

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

```r
require(sp)
sp.lines <- SpatialLines(list(Lines(list(Line(cbind(c(1,2,3),c(3,2,2)))),
  ID="2")))
sp.lines <- SpatialLinesDataFrame( sp.lines, data.frame(ID=1:2,
  row.names=c(1,2)) )

opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
  # Create systematic sample at 20 km spacing
  reg.sample <- sample.line(sp.lines, d = 20, type = "regular",
                             longlat = TRUE)
plot(sp.lines)
plot(reg.sample, pch = 20, add = TRUE)
```
# Create fixed size (n = 20) systematic sample
reg.sample <- sample.line(sp.lines, n = 20, type = "regular",
longlat = TRUE)
plot(sp.lines)
plot(reg.sample, pch = 20, add = TRUE)
box()
title("systematic n = 20")

# Create fixed size (n = 20) random sample
rand.sample <- sample.line(sp.lines, n = 20, type = "random",
longlat = TRUE)
plot(sp.lines)
plot(rand.sample, pch = 20, add = TRUE)
box()
title("rand n = 20")

# Create proportional (p = 0.10) random sample
rand.sample <- sample.line(sp.lines, p = 0.10, type = "random",
longlat = TRUE)
plot(sp.lines)
plot(rand.sample, pch = 20, add = TRUE)
box()
title("rand p = 0.10")
par(opar)

---

**sample.poly**  
**Sample Polygons**

**Description**
Creates an equal sample of n for each polygon in an sp Polygon class object

**Usage**
sample.poly(x, n = 10, type = "random", ...)

**Arguments**

- **x**  
  sp class SpatialPolygons or SpatialPolygonsDataFrame object

- **n**  
  Number of random samples

- **type**  
  Type of sample with options for: "random", "regular", "stratified", "nonaligned", "hexagonal", "clustered", "Fibonacci". See "spsample" for details.

- **...**  
  Additional arguments passed to spsample
**sampleTransect**  

**Sample transect**

**Description**

Creates random transects from points and generates sample points along each transect

**Usage**

```r
sampleTransect(x, min.length, max.length, id = NULL, ...)
```

**Arguments**

- **x**  
  A sp point object
- **min.length**  
  Minimum length of transect(s)
- **max.length**  
  Maximum length of transect(s)
- **id**  
  A unique identification column in x
- **...**  
  Additional arguments passed to sample.line

**Note**

Function create random direction and length transects and then creates a point sample along each transect. The characteristic of the sample points are defined by arguments passed to the sample.line function.

**Value**

sp SpatialPointsDataFrame object

**Author(s)**

Jeffrey S. Evans <jeffrey_evans@tnc.org>

**Examples**

```r
library(raster)
library(sp)
p <- raster(nrow=10, ncol=10)
p[] <- runif(ncell(p)) * 10
p <- rasterToPolygons(p, fun=function(x){x > 9})
s <- sample.poly(p, n = 5, type = "random")
plot(p)
plot(s, pch = 20, add = TRUE)
box()
title("Random sample (n=5) for each polygon")
```
Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

```r
library(sp)
data(meuse)
coordinates(meuse) <- ~x+y
proj4string(meuse) <- CRS("+init=epsg:28992")
meuse <- meuse[sample(1:nrow(meuse),10),]

transects <- sampleTransect(meuse, min.length=200, max.length=500, min.samp = 3)
plot(transects$transects)
plot(transects$samples, pch=20, add=TRUE)
```

sar

**Surface Area Ratio**

Description

Calculates the Berry (2002) Surface Area Ratio based on slope

Usage

```r
sar(x, s = NULL, ...)
```

Arguments

- `x` raster object
- `s` cell resolution (default is NULL, not needed if projection is in planar units)
- `...` Additional arguments passed to raster::calc

Value

raster class object of Berry (2002) Surface Area Ratio

Note

SAR is calculated as: \( \text{resolution}^2 \times \cos(\text{degrees(slope)} \times (\pi / 180)) \)

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References

Examples

```r
library(raster)
data(elev)
surface.ratio <- sar(elev, s=90)
plot(surface.ratio)
```

Description

Displays release notes

Usage

```r
se.news(...)```

Arguments

```r
...
```

not used

Description

Calculates variety of two-class sample separability metrics

Available statistics:

- M-Statistic (Kaufman & Remer 1994) - This is a measure of the difference of the distributional peaks. A large M-statistic indicates good separation between the two classes as within-class variance is minimized and between-class variance maximized (M <1 poor, M >1 good).
- Bhattacharyya distance (Bhattacharyya 1943; Harold 2003) - Measures the similarity of two discrete or continuous probability distributions.
- Jeffries-Matusita distance (Bruzzone et al., 2005; Swain et al., 1971) - The J-M distance is a function of separability that directly relates to the probability of how good a resultant classification will be. The J-M distance is asymptotic to $\chi^2$, where values of $\chi^2$ suggest complete separability.
- Divergence and transformed Divergence (Du et al., 2004) - Maximum likelihood approach. Transformed divergence gives an exponentially decreasing weight to increasing distances between the classes.
Usage

separability(
  x, y,
  plot = FALSE,
  cols = c("red", "blue"),
  clabs = c("Class1", "Class2"),
  ...
)

Arguments

  x    X vector
  y    Y vector
  plot plot separability (TRUE/FALSE)
  cols colors for plot (must be equal to number of classes)
  clabs labels for two classes
  ... additional arguments passes to plot

Value

A data.frame with the following separability metrics:

  • B - Bhattacharryya distance statistic
  • JM - Jeffries-Matusita distance statistic
  • M - M-Statistic
  • D - Divergence index
  • TD - Transformed Divergence index

Author(s)

Jeffrey S. Evans jeffrey_evans@tnc.org

References

Examples

```r
norm1 <- dnorm(seq(-20,20,length=5000),mean=0,sd=1)
norm2 <- dnorm(seq(-20,20,length=5000),mean=0.2,sd=2)
separability(norm1, norm2)

s1 <- c(1362,1411,1457,1735,1621,1621,1791,1863,1863,1838)
s2 <- c(1362,1411,1457,10030,1621,1621,1791,1863,1863,1838)
separability(s1, s2, plot=TRUE)
```

---

**sg.smooth**  
*Savitzky-Golay smoothing filter*

---

**Description**

Smoothing of time-series data using Savitzky-Golay convolution smoothing

**Usage**

```r
sg.smooth(x, f = 4, l = 51, d = 1, na.rm, ...)
```

**Arguments**

- `x`  
  A vector to be smoothed
- `f`  
  Filter type (default 4 for quartic, specify 2 for quadratic)
- `l`  
  Convolution filter length, must be odd number (default 51). Defines degree of smoothing
- `d`  
  First derivative (default 1)
- `na.rm`  
  NA behavior
- `...`  
  not used

**Value**

A vector of the smoothed data equal to length of x. Please note; NA values are retained

**Author(s)**

Jeffrey S. Evans <jeffrey_evans<at>tnc.org>

**References**

Examples

```r
y <- c(0.112220988, 0.055554941, 0.013333187, 0.055554941, 0.063332640, 0.014444285, 0.015555384, 0.057777140, 0.059999339, 0.034444068, 0.058888242, 0.136665165, 0.038888458, 0.096665606, 0.141109571, 0.015555384, 0.012222088, 0.012222088, 0.072221428, 0.052221648, 0.087776810, 0.014444285, 0.033332966, 0.012222088, 0.032221869, 0.059999339, 0.011110989, 0.042221759, 0.029999670, 0.018888680, 0.098888780, 0.016666483, 0.031110767, 0.061110441, 0.022221979, 0.073332526, 0.012222088, 0.016666483, 0.012222088, 0.122220881, 0.134442955, 0.094443403, 0.128888745, 0.045555055, 0.152220547, 0.071110331, 0.018886800, 0.022221979, 0.029999670, 0.035555165, 0.014444285, 0.049999449, 0.074443623, 0.06888135, 0.062221535, 0.032221869, 0.095554501, 0.143331751, 0.121109776, 0.06554835, 0.074443623, 0.043332856, 0.017777583, 0.016666483, 0.036666263, 0.152220547, 0.032221869, 0.009999890, 0.009999890, 0.021110879, 0.025555275, 0.099998899, 0.015555384, 0.086665712, 0.008888791, 0.062221535, 0.044443958, 0.081110224, 0.015555384, 0.089999085, 0.082221314, 0.056666043, 0.013333187, 0.048888352, 0.075554721, 0.025555275, 0.056666043, 0.146665052, 0.118887581, 0.125554174, 0.024444176, 0.124443069, 0.012222088, 0.126665279, 0.048888352, 0.046666153, 0.141109571, 0.015555384, 0.114443190)
```

```r
plot(y, type="l", lty = 3, main="Savitzky-Golay with l = 51, 25, 10")
lines(sg.smooth(y),col="red", lwd=2)
lines(sg.smooth(y, l = 25),col="blue", lwd=2)
lines(sg.smooth(y, l = 10),col="green", lwd=2)
```

#### function applied to a raster stack and sp object

```r
library(raster)
random.raster <- function(r=50, c=50, l=10, min=0, max=1){
  do.call(stack, replicate(l, raster(matrix(runif(r*c, min, max),r,c))))
}
r <- random.raster()
```

# raster stack example

```r
( r.sg <- calc(r, sg.smooth) )
```

# sp SpatialPixelsDataFrame example

```r
r.sp <- as(r, "SpatialPixelsDataFrame")
r.sp@data <- as.data.frame(t(apply(r.sp@data, MARGIN=1, FUN=sg.smooth)))
```

---

**shannons**

**Shannon’s Diversity (Entropy) Index**

**Description**

Calculates Shannon’s Diversity Index and Shannon’s Evenness Index

**Usage**

```r
shannons(x, counts = TRUE, ens = FALSE, margin = "row")
```
Arguments

- `x` data.frame object containing counts or proportions
- `counts` Are data counts (TRUE) or relative proportions (FALSE)
- `ens` Calculate effective number of species (TRUE/FALSE)
- `margin` Calculate diversity for rows or columns. c("row", "col")

Value

data.frame with "H" (Shannon's diversity) and "evenness" (Shannon's evenness where H / max(sum(x))) and ESN

Note

The expected for H is 0-3+ where a value of 2 has been suggested as medium-high diversity, for evenness is 0-1 with 0 signifying no evenness and 1, complete evenness.

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


Examples

# Using Costa Rican ant diversity data from Roth et al. (1994)
data(ants)

# Calculate diversity for each covertype ("col")
shannons(ants[,2:ncol(ants)], ens = TRUE, counts = FALSE, margin = "col")

# Calculate diversity for each species ("row")
ant.div <- shannons(ants[,2:ncol(ants)], ens = TRUE, counts = FALSE, 
margin = "row")
row.names(ant.div) <- ants[,1]
ant.div
Description

Shift a vector by specified positive or negative lag

Usage

shift(x, lag = 1, pad = NA)

Arguments

x  A vector
lag  Number of lagged offsets, default is 1
pad  Value to fill the lagged offset with, default is NA

Value

a vector, length equal to x, with offset length filled with pad values

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

x <- 1:10

shift(x, 1)  # shift positive (from beginning of vector) by 1
shift(x, -1)  # shift negative (from end of vector) by 1
shift(x, 5, 0)  # Shift by 5 and fill (pad) with 0


Description

Uses row imputation to identify "k" ecological similar observations
similarity

Usage

similarity(
  x,
  k = 4,
  method = "mahalanobis",
  frequency = TRUE,
  scale = TRUE,
  ID = NULL
)

Arguments

x       data.frame containing ecological measures
k       Number of k nearest neighbors (kNN)
method  Method to compute multivariate distances c("mahalanobis", "raw", "euclidean", "ica")
frequency Calculate frequency of each reference row (TRUE/FALSE)
scale    Scale multivariate distances to standard range (TRUE/FALSE)
ID       Unique ID vector to use as reference ID's (rownames). Must be unique and same length as number of rows in x

Value

data.frame with k similar targets and associated distances. If frequency = TRUE the freq column represents the number of times a row (ID) was selected as a neighbor.

Note

This function uses row-based imputation to identify k similar neighbors for each observation. Has been used to identify offsets based on ecological similarity.

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


Examples

library(sp)
data(pu)
kNN <- similarity(pu@data[2:ncol(pu)], k = 4, frequency = FALSE,
                  ID = pu@data$UNIT_ID)
smooth.time.series <- similarity(pu@data[2:ncol(pu)], k = 4, frequency = TRUE, ID = pu@data$UNIT_ID)
p <- kNN$freq
clr <- c("#3288BD", "#99D594", "#E6F598", "#FEE08B", "#FC8D59", "#D53E4F")
p <- ifelse(p <= 0, clr[1],
  ifelse(p > 0 & p < 10, clr[2],
    ifelse(p >= 10 & p < 20, clr[3],
      ifelse(p >= 20 & p < 50, clr[4],
        ifelse(p >= 50 & p < 100, clr[5],
          ifelse(p >= 100, clr[6], NA)))))
plot(pu, col=p, border=NA)
legend("topleft", legend=c("None","<10","10-20","20-50","50-100",">100"),
  fill=clr, cex=0.6, bty="n")
box()

smooth.time.series  Smooth Raster Time-series

Description
Smooths pixel-level data in raster time-series and can impute missing (NA) values.

Usage
smooth.time.series(x, f = 0.8, smooth.data = FALSE, ...)

Arguments
- x: A raster stack/brick or sp object with a @data slot
- f: Smoothing parameter (see loess span argument)
- smooth.data: (FALSE/TRUE) Smooth all of the data or just impute NA values
- ...: Additional arguments passed to raster calc (for writing results to disk)

Details
This function uses a LOESS regression to smooth the time-series (using the smooth.data = TRUE argument). If the data is smoothed, it will be replaced by a loess estimate of the time-series (estimated distribution at the pixel-level). The results can dramatically be effected by the choice of the smoothing parameter (f) so caution is warranted and the effect of this parameter tested. Alternately, with smooth.data = FALSE, the function can be used to impute missing pixel data (NA) in raster time-series (stacks/bricks).

Value

A raster stack or brick pr data.frame object with imputed NA values or smoothed data.

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

See Also

loess for details on the loess regression
calc for details on additional (...) arguments

Examples

```r
## Not run:
random.raster <- function(r=50, c=50, l=10, min=0, max=1){
  do.call(stack, replicate(l, raster(matrix(runif(r*c, min, max),r,c))))
} # <- random.raster()

#### Smooth time-series using raster stack/brick
r.smooth <- smooth.time.series(r, f = 0.6, smooth.data = TRUE)

#### sp SpatialPixelsDataFrame example
r <- as(r, "SpatialPixelsDataFrame")
# extract pixel 100 for plotting
y <- as.numeric(r@data[100,])

# Smooth data
r@data <- smooth.time.series(r, f = 0.6, smooth.data = TRUE)

# plot results
plot(y, type="l")
lines(as.numeric(r@data[100,]), col="red")
legend("bottomright", legend=c("original","smoothed"),
  lty=c(1,1), col=c("black","red"))

# coerce back to raster stack object
r <- stack(r)

## End(Not run)
```
Description

An isotropic image gradient operator using a 3x3 window.

The Sobel-Feldmanh operator is a discrete differentiation operator, deriving an approximation of the gradient of the intensity function. An abrupt discontinuity in the gradient function represents edges, making this a common approach for edge detection. The Sobel-Feldman operator is based on convolving the image with a small, separable, and integer matrix in the horizontal and vertical directions. The operator uses two 3x3 kernels which are convolved with the original image to calculate approximations of the derivatives - one for horizontal changes, and one for vertical. Where x is defined here as increasing in the right-direction, and y as increasing in the down-direction. At each pixel in the raster, the resulting gradient can be combined to give the gradient intensity, using: \( \sqrt{G_x^2 G_y^2} \). This can be expanded into the gradient direction using \( \arctan(G_x/G_y) \).

Usage

sobal(x, method = "intensity", ...)

Arguments

x A raster class object

method Type of operator ("intensity", "direction", "edge")

... Additional arguments passed to raster::overlay or, if method="edge", raster::focal
(if you want a file written to disk use filename = "" argument)

Value

A raster class object or raster written to disk

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


Examples

library(raster)
r <- brick(system.file("external/rlogo.grd", package="raster"))
s.int <- sobal(r[[1]])
s.dir <- sobal(r[[1]], method = "direction")
s.edge <- sobal(r[[1]], method = "edge")

opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
plot(r[[1]])
plot(s.int, main="intensity")
plot(s.dir, main="direction")
sp.kde

Spatial kernel density estimate

Description

A weighted or unweighted Gaussian Kernel Density estimate for spatial data

Usage

```
sp.kde(
  x,
  y = NULL,
  bw = NULL,
  newdata = NULL,
  nr = NULL,
  nc = NULL,
  standardize = FALSE,
  scale.factor = NULL,
  mask = TRUE
)
```

Arguments

- **x**: sp SpatialPointsDataFrame object
- **y**: Optional values, associated with x coordinates, to be used as weights
- **bw**: Distance bandwidth of Gaussian Kernel, must be units of projection
- **newdata**: A Rasterlayer, any sp class object or c[xmin,xmax,ymin,ymax] vector to estimate the kde extent
- **nr**: Number of rows used for creating grid. If not defined a value based on extent or existing raster will be used
- **nc**: Number of columns used for creating grid. If not defined a value based on extent or existing raster will be used
- **standardize**: Standardize results to 0-1 (FALSE/TRUE)
- **scale.factor**: Optional numeric scaling factor for the KDE (eg., 10000), to account for small estimate values
- **mask**: (TRUE/FALSE) mask resulting raster if newdata is provided

Value

Raster class object containing kernel density estimate
library(sp)
library(raster)
data(meuse)
coordinates(meuse) <- ~x+y

# Unweighted KDE (spatial locations only)
pt.kde <- sp.kde(x = meuse, bw = 1000, standardize = TRUE,
                 nr=104, nc=78, scale.factor = 10000 )

# Plot results
plot(pt.kde, main="Unweighted kde")
points(meuse, pch=20, col="red")

#### Using existing raster(s) to define grid ####

# Weighted KDE using cadmium and extent with row & col to define grid
e <- c(178605, 181390, 329714, 333611)
cadmium.kde <- sp.kde(x = meuse, y = meuse$cadmium, bw = 1000,
                      nr = 104, nc = 78, newdata = e,
                      standardize = TRUE,
                      scale.factor = 10000 )
plot(cadmium.kde)
points(meuse, pch=19)

# Weighted KDE using cadmium and raster object to define grid
r <- raster::raster(raster::extent(c(178605, 181390, 329714, 333611)),
                     nrow=104, ncol=78)
r[] <- rep(1,ncell(r))
cadmium.kde <- sp.kde(x = meuse, y = meuse$cadmium, bw = 1000,
                      newdata = r, standardize = TRUE,
                      scale.factor = 10000 )
plot(cadmium.kde)
points(meuse, pch=19)

# Weighted KDE using cadmium and SpatialPixelsDataFrame object to define grid
data(meuse.grid)
coordinates(meuse.grid) = ~x+y
proj4string(meuse.grid) <- CRS("+init=epsg:28992")
girded(meuse.grid) = TRUE
cadmium.kde <- sp.kde(x = meuse, y = meuse$cadmium, bw = 1000,
                      newdata = meuse.grid, standardize = TRUE,
                      scale.factor = 10000 )
plot(cadmium.kde)
points(meuse, pch=19)
Description

Removes row or column NA's in sp object

Usage

sp.na.omit(x, col.name = NULL, margin = 1)

Arguments

x Object of class SpatialPointsDataFrame OR SpatialPolygonsDataFrame
col.name The name of a specific column to remove NA's from
margin Margin (1,2) of data.frame 1 for rows or 2 for columns

Note

This function will remove all NA's in the object or NA's associated with a specific column.

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

library(sp)
data(meuse)
coordinates(meuse) <- ~x+y

# Display rows with NA
meuse@data[!complete.cases(meuse@data),]

# Remove all NA's in rows (and associated points)
meuse2 <- sp.na.omit(meuse)
dim(meuse)
dim(meuse2)

# Plot deleted points in red
plot(meuse, col='red', pch=20)
plot(meuse2, col='black', pch=20, add=TRUE)

# Remove NA's associated with specific column
meuse2 <- sp.na.omit(meuse, col.name = "om")
head(meuse@data)
head(meuse2@data)
**spatial.select**

**Spatial Select**

**Description**
Performs a spatial select (feature subset) between a polygon(s) and other feature class

Performs a spatial select of features based on an overlay of a polygon (x), which can represent multiple features, and a polygon, point or line feature classes (y). User can specify a partial or complete intersection, using within argument, or within a distance, using distance argument, predicated on the query polygon. This function is similar to ArcGIS/Pro spatial select. Please note that for point to point neighbor selections use the knn function.

**Usage**

```r
spatial.select(
  x,
  y = NULL,
  distance = NULL,
  predicate = c("intersect", "contains", "covers", "touches", "proximity", "contingency"),
  neighbors = c("queen", "rook")
)
```

**Arguments**

- **x**  
  An sp or sf polygon(s) object that defines the spatial query

- **y**  
  A sp or sf feature class that will be subset by the query of x

- **distance**  
  A proximity distance of features to select (within distance)

- **predicate**  
  Spatial predicate for intersection

- **neighbors**  
  If predicate = "contingency" type of neighbors options are c("queen", "rook")

**Value**

An sp object representing a subset of y based on the spatial query of x or, if predicate = contingency a sparse matrix representing neighbor indexes

**Note**
Valid spatial predicates include: intersect, touches, covers, contains, proximity and contingency. See [DE-9IM topology model](https://en.wikipedia.org/wiki/DE-9IM) for detailed information on data predicates.

**Author(s)**

Jeffrey S. Evans <jeffrey_evans@tnc.org>
See Also

- **gIntersects** for details on intersect predicate
- **gContains** for details on contain predicate
- **gCovers** for details on covers predicate
- **gTouches** for details on touches predicate
- **gWithinDistance** for details on proximity predicate


Examples

```r
library(raster)
library(sp)

data(meuse)
  coordinates(meuse) <- ~x+y

spolys <- hexagons(meuse, res=100)
p <- raster(extent(spolys), res=800)
p[] <- runif(ncell(p)) * 10
  p <- rasterToPolygons(p, fun=function(x){x > 6})

### On polygons
sub.int <- spatial.select(p, spolys, predicate = "intersect")
sub.contains <- spatial.select(p, spolys, predicate = "contains")
sub.cov <- spatial.select(p, spolys, predicate = "covers")
sub.touches <- spatial.select(p, spolys, predicate = "touches")
sub.prox <- spatial.select(p, spolys, distance=100, predicate = "proximity")

opar <- par(no.readonly=TRUE)
par(mfrow=c(2,3))
  plot(spolys, main="all data")
  plot(p, add=TRUE)
  plot(sub.int, main="intersects")
  plot(p, add=TRUE)
  plot(sub.contains, main="contains")
  plot(p, add=TRUE)
  plot(sub.cov, main="covers")
  plot(p, add=TRUE)
  plot(sub.touches, main="touches")
  plot(p, add=TRUE)
  plot(sub.prox, main="Proximity 100m distance")
  plot(p, add=TRUE)
par(opar)

### On points
### note; touches is not relevant for points and intersect/contains/covers
### yield the same results
sub.int <- spatial.select(p, meuse, predicate = "intersect")
sub.contains <- spatial.select(p, meuse, predicate = "contains")
sub.prox <- spatial.select(p, meuse, distance=200, predicate = "proximity")
```
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
plot(meuse, main="all data", pch=20)
plot(p, add=TRUE)
plot(sub.int, main="intersects", pch=20)
plot(p, add=TRUE)
plot(sub.contains, main="contains", pch=20)
plot(p, add=TRUE)
plot(sub.prox, main="Proximity 200m distance", pch=20)
plot(p, add=TRUE)
par(opar)

### For rook or queen polygon contingency
spolys <- as(sf::st_make_grid(sf::st_sfc(sf::st_point(c(0,0)),
                                 sf::st_point(c(3,3))), n = c(3,3)), "Spatial")

spatial.select(spolys, predicate = "contingency")
spatial.select(spolys, predicate = "contingency", neighbors = "rook")

spectral.separability spectral separability

Description

Calculates spectral separability by class

Available statistics:

- Bhattacharyya distance (Bhattacharyya 1943; Harold 2003) measures the similarity of two
  discrete or continuous probability distributions.
- Jeffries-Matusita (default) distance (Bruzzone et al., 2005; Swain et al., 1971) is a scaled (0-
  2) version of Bhattacharyya. The J-M distance is asymptotic to 2, where 2 suggest complete
  separability.

Usage

spectral.separability(x, y, jeffries.matusita = TRUE)

Arguments

- x: data.frame, matrix or vector of spectral values must, match classes defined in y
- y: A vector or factor with grouping classes, must match row wise values in x
- jeffries.matusita (TRUE/FALSE) Return J-M distance (default) else Bhattacharyya

Value

A matrix of class-wise Jeffries-Matusita or Bhattacharyya distance separability values
Author(s)

Jeffrey S. Evans jeffrey_evans@tnc.org

References


Examples

```r
# Create example data
require(MASS)
d <- 6 # Number of bands
n.class <- 5 # Number of classes
n <- rep(1000, 5)
mu <- round(matrix(rnorm(d*n.class, 128, 1),
    ncol=n.class, byrow=TRUE), 0)
x <- matrix(double(), ncol=d, nrow=0)
classes <- integer()
for (i in 1:n.class) {
  f <- svd(matrix(rnorm(d^2), ncol=d))
sigma <- t(f$v) %*% diag(rep(10, d)) %*% f$v
  x <- rbind(x, mvrnorm(n[i], mu[, i], sigma))
classes <- c(classes, rep(i, n[i]))
}
colnames(x) <- paste0("band", 1:6)
classes <- factor(classes, labels=c("water", "forest",
  "shrub", "urban", "ag"))

# Separability for multi-band (multivariate) spectra
spectral.separability(x, classes)

# Separability for single-band (univariate) spectra
spectral.separability(x[,1], classes)
```

spherical.sd

Spherical Variance or Standard Deviation of Surface

Description

Derives the spherical standard deviation of a raster surface
spherical.sd

Usage

spherical.sd(r, d, variance = FALSE, ...)

Arguments

r
Raster class object

d
Size of focal window or a matrix to use in focal function

variance
(FALSE|TRUE) Output spherical variance rather than standard deviation

... Additional arguments passed to calc (can write raster to disk here)

Details

Surface variability using spherical variance/standard deviation. The variation can be assessed using the spherical standard deviation of the normal direction within a local neighborhood. This is found by expressing the normal directions on the surfaces cells in terms of their displacements in a Cartesian (x,y,z) coordinate system. Averaging the x-coordinates, y-coordinates, and z-coordinates separately gives a vector (xb, yb, zb) pointing in the direction of the average normal. This vector will be shorter when there is more variation of the normals and it will be longest–equal to unity–when there is no variation. Its squared length is (by the Pythagorean theorem) given by: R^2 = xb^2 + yb^2 + zb^2 where; x = \cos(aspect) * \sin(slope) and xb = nXn focal mean of x y = \sin(aspect) * \sin(slope) and yb = nXn focal mean of y z = \cos(slope) and zb = nXn focal mean of z

The slope and aspect values are expected to be in radians. The value of (1 - R^2), which will lie between 0 and 1, is the spherical variance. and it’s square root can be considered the spherical standard deviation.

Value
rasterLayer class object of the spherical standard deviation

Author(s)

Jeffrey S. Evans <jeffrey_evans<at>tnc.org>

See Also

focal for details on focal function
calc for details on ... arguments

Examples

library(raster)
data(elev)

ssd <- spherical.sd(elev, d=5)
slope <- terrain(elev, opt='slope')
aspect <- terrain(elev, opt='aspect')
srr <- hillShade(slope, aspect, 40, 270)
plot(hill, col=grey(0:100/100), legend=FALSE,
     main='terrain spherical standard deviation')
plot(ssd, col=rainbow(25, alpha=0.35), add=TRUE)

---

**srr**  
*Surface Relief Ratio*

### Description
Calculates the Pike (1971) Surface Relief Ratio

### Usage
`srr(x, s = 5, ...)`

### Arguments
- `x`: raster object
- `s`: Focal window size
- `...`: Additional arguments passed to `raster::calc`

### Value
raster class object of Pike’s (1971) Surface Relief Ratio

### Note
Describes rugosity in continuous raster surface within a specified window. The implementation of SRR can be shown as: \( \frac{\text{mean}(x) - \text{min}(x)}{\text{max}(x) - \text{min}(x)} \)

### Author(s)
Jeffrey S. Evans <jeffrey_evans@tnc.org>

### Examples
```r
library(raster)
data(elev)
r.srr <- srr(elev, s=5)
plot(r.srr, main="Surface Relief Ratio")
```
## stratified.random

**Stratified random sample**

### Description

Creates a stratified random sample of an sp class object

### Usage

```r
stratified.random(x, strata, n = 10, reps = 1, replace = TRUE)
```

### Arguments

- **x**: sp class SpatialDataFrame object (point, polygon, line, pixel)
- **strata**: Column in @data slot with stratification factor
- **n**: Number of random samples
- **reps**: Number of replicates per strata
- **replace**: Sampling with replacement (TRUE|FALSE)

### Value

sp SpatialDataFrame object (same as input feature) containing random samples

### Note

If replace=FALSE features are removed from consideration in subsequent replicates. Conversely, if replace=TRUE, a feature can be selected multiple times across replicates. Not applicable if rep=1.

**Depends**: sp

### Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

### References

subsample.distance

Examples

```r
require(sp)
data(meuse)
coordinates(meuse) <- ~x+y

# Create stratified variable using quartile breaks
x1 <- cut(meuse@data[, "cadmium"], summary(meuse@data[, "cadmium"])[-4],
           include.lowest=TRUE)
levels(x1) <- seq(1,nlevels(x1),1)
x2 <- cut(meuse@data[, "lead"], summary(meuse@data[, "lead"])[-4],
           include.lowest=TRUE)
levels(x2) <- seq(1,nlevels(x2),1)
meuse@data <- cbind(meuse@data, STRAT=paste(x1, x2, sep='.' )

# 2 replicates and replacement
ssample <- stratified.random(meuse, strata='STRAT', n=2, reps=2)

# 2 replicates and no replacement
ssample.nr <- stratified.random(meuse, strata='STRAT', n=2, reps=2,
                                replace=FALSE)

# n=1 and reps=10 for sequential numbering of samples
ssample.ct <- stratified.random(meuse, strata='STRAT', n=1, reps=10,
                                replace=TRUE)

# Counts for each full strata (note; 2 strata have only 1 observation)
tapply(meuse@data$STRAT, meuse@data$STRAT, length)

# Counts for each sampled strata, with replacement
tapply(ssample@data$STRAT, ssample@data$STRAT, length)

# Counts for each sampled strata, without replacement
tapply(ssample.nr@data$STRAT, ssample.nr@data$STRAT, length)

# Counts for each sampled strata, without replacement
tapply(ssample.ct@data$STRAT, ssample.ct@data$STRAT, length)

# Plot random samples colored by replacement
ssample@data$REP <- factor(ssample@data$REP)
spplot(ssample, 'REP', col.regions=c('red','blue'))
```

subsample.distance

Distance-based subsampling

Description

Draws a minimum, and optional maximum constrained, distance sub-sampling
Usage

```r
subsample.distance(
  x,
  size,
  d,
  d.max = NULL,
  replacement = FALSE,
  latlong = FALSE,
  echo = FALSE
)
```

Arguments

- **x** A spatial polygons or points sp object
- **size** Subsample size
- **d** Minimum sampling distance
- **d.max** Maximum sampling distance
- **replacement** (FALSE/TRUE) Subsample with replacement
- **latlong** (FALSE/TRUE) Is the data in a geographic projection
- **echo** (FALSE/TRUE) Print min and max sample distances

Value

A subsampled spatial polygons or points sp object

Note

This function provides a distance constrained subsample of existing point or polygon data

Author(s)

Jeffrey S. Evans <jeffrey.evans@tnc.org>

Examples

```r
library(sp)
data(meuse)
coordinates(meuse) <- ~ x+y

# Subsample with a 500m minimum sample spread
sub.meuse <- subsample.distance(meuse, size = 10, d = 500, echo = TRUE)
plot(meuse, pch=19, main="min dist = 500")
points(sub.meuse, pch=19, col="red")

# Check distances
dm <- spDists(sub.meuse)
```
summary.cross.cor

Summary of spatial cross correlation

Description

summary method for class "cross.cor"

Usage

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

Arguments

  object Object of class cross.cor
  ... Ignored

summary.effect.size

Summary of effect size

Description

Summary method for class "effect.size".

Usage

## S3 method for class 'effect.size'
summary(object, ...)
**Arguments**

- `object` Object of class `effect.size`
- `...` Ignored

**Summary.loess.boot**

*Summarizing Loess bootstrap models*

**Description**

Summary method for class "loess.boot".

**Usage**

```r
## S3 method for class 'loess.boot'
summary(object, ...)
```

**Arguments**

- `object` Object of class `loess.boot`
- `...` Ignored

---

**swvi**

*Senescence weighted Vegetation Index (swvi)*

**Description**

Modified Soil-adjusted Vegetation Index (MSAVI) or Modified Triangular Vegetation Index 2 (MTVI) weighted by the Normalized difference senescent vegetation index (NDSVI).

The intent of this index is to correct the MSAVI or MTVI index for bias associated with senescent vegetation. This is done by:

- deriving the NDSVI;
- applying a threshold to limit NDSVI to values associated with senescent vegetation;
- converting the index to inverted weights \((-1 \times \text{NDSVI} / \text{sum(NDSVI)})\);
- applying weights to MSAVI or MTVI

The MSAVI formula follows the modification proposed by Qi et al. (1994), often referred to as MSAVI2. MSAVI index reduces soil noise and increases the dynamic range of the vegetation signal. The implemented modified version (MSAVI2) is based on an inductive method that does not use a constant $L$ value, in separating soil effects, and highlights healthy vegetation. The MTVI(2) index follows Haboudane et al., (2004) and represents the area of a hypothetical triangle in spectral space that connects (1) green peak reflectance, (2) minimum chlorophyll absorption, and (3) the NIR shoulder. When chlorophyll absorption causes a decrease of red reflectance, and leaf tissue abundance causes an increase in NIR reflectance, the total area of the triangle increases. It is good
for estimating green LAI, but its sensitivity to chlorophyll increases with an increase in canopy
density. The modified version of the index accounts for the background signature of soils while
preserving sensitivity to LAI and resistance to the influence of chlorophyll.

The Normalized difference senescent vegetation index (NDSVI) follows methods from Qi et a.,
(2000). The senescence is used to threshold the NDSVI. Values less then this value will be NA. The
threshold argument is used to apply a threshold to MSAVI. The default is NULL but if specified
all values (MSAVI <= threshold) will be NA. Applying a weight.factor can be used to change the
influence of the weights on MSAVI.

Usage

```r
text
```

Arguments

- `red`: Red band (0.636 - 0.673mm), landsat 5&7 band 3, OLI (landsat 8) band 4
- `nir`: Near infrared band (0.851 - 0.879mm) landsat 5&7 band 4, OLI (landsat 8) band 5
- `swir`: Short-wave infrared band 1 (1.566 - 1.651mm), landsat 5&7 band 5, OLI (landsat 8) band 6
- `green`: Green band if MTVI = TRUE
- `mtvi`: (FALSE | TRUE) Use Modified Triangular Vegetation Index 2 instead of MSAVI
- `senescence`: The critical value, in NDSVI, representing senescent vegetation
- `threshold`: Threshold value for defining NA based on < p
- `weight.factor`: Apply partial weights (w * weight.factor) to the NDSVI weights
- `...`: Additional arguments passed to raster calc function

Value

```r
rasterLayer class object of the weighted MSAVI metric
```

Author(s)

Jeffrey S. Evans jeffrey_evans@tnc.org
References


Examples

```r
## Not run:
library(raster)
library(RStoolbox)

data(lsat)
lsat <- radCor(lsat, metaData = readMeta(system.file(  
"external/landsat/LT52240631988227CLB02_MTL.txt",  
package="RStoolbox")), method = "apref")

# Using Modified Soil-adjusted Vegetation Index (MSAVI)
( wmsavi <- swvi(red = lsat[[3]], nir = lsat[[4]], swir = lsat[[5]]) )  
plotRGB(lsat, r=6,g=5,b=2, scale=1, stretch="lin")
plot(wmsavi, legend=FALSE, col=rev(terrain.colors(100, alpha=0.35)), add=TRUE )

# Using Modified Triangular Vegetation Index 2 (MTVI)
( wmtvi <- swvi(red = lsat[[3]], nir = lsat[[4]], swir = lsat[[5]],
               green = lsat[[3]], mtvi = TRUE) )
plotRGB(lsat, r=6,g=5,b=2, scale=1, stretch="lin")
plot(wmtvi, legend=FALSE, col=rev(terrain.colors(100, alpha=0.35)), add=TRUE )

## End(Not run)
```

time_to_event

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns the time (sum to position) to a specified value</td>
</tr>
</tbody>
</table>

The time to event represents the sum of positions, in the vector, until the specified value is found. For example, (0,0,1) would be 3 or 2 with up.to=TRUE. The int argument allows for rounding a continuous variable. Since it may be difficult to fine an exact match to a floating point value rounding mitigates the problem. If you want a specific rounding value (e.g., 1 decimal place) you can apply it to x first then pass it to the function.
time_to_event

Usage

time_to_event(x, y = 1, dir = c("LR", "RL"), int = FALSE, up.to = FALSE)

Arguments

x A vector, representing time-series, to evaluate
y Threshold value for return position for
dir Direction of evaluation c("LR", "RL")
int FALSE | TRUE - Evaluate as integer (rounds to 0 decimal places)
up.to FALSE | TRUE - Return value before event

Value

A vector value representing the time to event

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

library(raster)

# Binomial instance
time_to_event(c(0,0,0,0,1,0,0,0,0,0,0,0,1,0))
time_to_event(c(0,0,0,0,1,0,0,0,0,1,0), up.to = TRUE)
time_to_event(c(0,0,0,0,1,0,0,0,1,0), dir="RL")

r <- do.call(raster::stack, replicate(20,raster::raster(matrix(sample(c(0,1), 100, replace=TRUE), 100, 100))))
( t2e <- calc(r, fun=time_to_event) )

# Continuous threshold instance
( x <- runif(100, 0,7) )
time_to_event(x, y = 5, int=TRUE)

r <- do.call(raster::stack, replicate(20,raster::raster(matrix( runif(1000,0,7), 100, 100))))
t2e <- function(x) { time_to_event(x, y=5, int=TRUE) }
( t2e <- calc(r, fun=time_to_event) )
topo.distance  Topographic distance

Description

Calculates topographic corrected distance for a SpatialLinesDataFrame object

Usage

topo.distance(x, r, echo = FALSE)

Arguments

x  sp SpatialLinesDataFrame object
r  raster class elevation raster
echo (FALSE/TRUE) print progress to screen

Value

Vector of corrected topographic distances same length as nrow(x)

Note

This function corrects straight-line (euclidean) distances for topographic-slope effect.

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

library(sp)
library(raster)
library(GeNetIt)

# create example data
data(elev)
  r <- projectRaster(elev, res=c(1000,1000),
    crs="+proj=aea +lat_1=29.5 +lat_2=42.5")
  e <- extent(616893.6,714697.3,5001027,5080542)
  elev <- crop(r,e)
  names(elev) <- "elev"
  pts <- sampleRandom(elev, 10, sp=TRUE)
  pts$ID <- LETTERS[seq( from = 1, to = nrow(pts) )]

  graph <- GeNetIt::knn.graph(pts, row.names=pts$data[,"ID"])
  proj4string(graph) <- proj4string(elev)
  head(graph$data)
plot(elev)
plot(graph, cex=0.5, add=TRUE)
plot(pts,pch=19,col="red",add=TRUE)

# Calculate topographical distance
(tdist <- topo.distance(graph, elev))

# Increase in corrected distance
tdist - graph$length

# Percent increase in corrected distance
(((tdist - graph$length) / graph$length) * 100

---

**tpi**  
*Topographic Position Index (tpi)*

**Description**
Calculates topographic position using mean deviations

**Usage**
tpi(x, scale = 3, win = "rectangle", normalize = FALSE, zero.correct = FALSE)

**Arguments**
- **x** A raster class object
- **scale** focal window size (n-cell x n-cell for rectangle or distance for circle)
- **win** Window type. Options are "rectangle" and "circle"
- **normalize** Apply deviation correction that normalizes to local surface roughness
- **zero.correct** Apply correction for zero values in matrix weights

**Value**
A raster class object of tpi metric

**Author(s)**
Jeffrey S. Evans <jeffrey.evans@tnc.org>

**References**
### Examples

```r
library(raster)
data(elev)

# calculate tpi and plot
tpi7 <- tpi(elev, scale=7)
tpi025 <- tpi(elev, win = "circle", scale=0.025)
tpi025.zc <- tpi(elev, win = "circle", scale=0.025, zero.correct = TRUE)

opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
  plot(elev, main="original raster")
  plot(tpi7, main="tpi 7x7")
  plot(tpi025, main="tpi Circular window d=0.025")
  plot(tpi025, main="tpi Circular window d=0.025, zero correct")
par(opar)
```

---

**trasp**  
*Solar-radiation Aspect Index*

### Description


Roberts and Cooper (1989) rotates (transforms) the circular aspect to assign a value of zero to land oriented in a north-northeast direction, (typically the coolest and wettest orientation), and a value of one on the hotter, dryer south-southwesterly slopes. The result is a continuous variable between 0 - 1. The metric is defined as: \( \text{trasp} = \frac{1 - \cos((\pi/180)(a-30))}{2} \) where; \( a \) = aspect in degrees.

### Usage

```r
trasp(x, ...)
```

### Arguments

- **x**  
  raster object

- **...**  
  Additional arguments passed to raster::calc

### Value

raster class object of Roberts and Cooper (1989) Solar-radiation Aspect Index

### Author(s)

Jeffrey S. Evans <jeffrey.evans@tnc.org>
References


Examples

```r
library(raster)
data(elev)
s <- trasp(elev)
plot(s)
```

Description

Calculated specified trend line of x,y

Usage

```r
trend.line(x, y, type = "linear", plot = TRUE, ...)
```

Arguments

- **x**: Vector of x
- **y**: Vector of y
- **type**: Trend line types are: 'linear', 'exponential', 'logarithmic', 'polynomial'
- **plot**: plot results (TRUE/FALSE)
- **...**: Additional arguments passed to plot

Value

A list class object with the following components:

- for type = 'linear' x is slope and y is intercept
- for type = 'exponential', 'logarithmic', or 'polynomial' x is original x variable and y is vector of fit regression line

Author(s)

Jeffrey S. Evans jeffrey_evans@tnc.org
Examples

```r
x <- 1:10
y <- jitter(x^2)

opar <- par(no.readonly=TRUE)
par(mfcol=c(2,2))
trend.line(x,y,type='linear',plot=TRUE,pch=20,main='Linear')
trend.line(x,y,type='exponential',plot=TRUE,pch=20,main='Exponential')
trend.line(x,y,type='logarithmic',plot=TRUE,pch=20,main='Logarithmic')
trend.line(x,y,type='polynomial',plot=TRUE,pch=20,main='Polynomial')
par(opar)
```

---

**tri**  
*Terrain Ruggedness Index*

**Description**


The algebraic approximation is considerably faster. However, because inclusion of the center cell, the larger the scale the larger the divergence of the minimum value.

Recommended ranges for classifying Topographic Ruggedness Index:

- 0-80 - level terrain surface.
- 81-116 - nearly level surface.
- 117-161 - slightly rugged surface.
- 162-239 - intermediately rugged surface.
- 240-497 - moderately rugged surface.
- 498-958 - highly rugged surface.
- gt 959 - extremely rugged surface.

**Usage**

```r
tri(r, s = 3, exact = TRUE, file.name = NULL, ...)
```

**Arguments**

- `r`  
  RasterLayer class object

- `s`  
  Scale of window. Must be odd number, can represent 2 dimensions (eg., `s=c(3,5)` would represent a 3 x 5 window)

- `exact`  
  Calculate (TRUE/FALSE) the exact TRI or an algebraic approximation.

- `file.name`  
  Name of output raster (optional)

- `...`  
  Additional arguments passed to writeRaster
Value
raster class object or raster written to disk

Author(s)
Jeffrey S. Evans jeffrey_evans@tnc.org

References

Examples

```
library(raster)
data(elev)
 ( tri.ext <- tri(elev) )
 ( tri.app <- tri(elev, exact = FALSE) )
plot(stack(tri.ext, tri.app))
```

---

### vrm

**Vector Ruggedness Measure (VRM)**

---

Description
Implementation of the Sappington et al., (2007) vector ruggedness measure

Usage

```
vrm(x, s = 3, file.name = NULL, ...)
```

Arguments

- `x`  
  Elevation raster class object
- `s`  
  Scale of window. Must be odd number, can represent 2 dimensions (eg., `s=c(3,5)` would represent a 3 x 5 window)
- `file.name`  
  Name of output raster (optional)
- `...`  
  Additional arguments passed to writeRaster

Value
raster class object or raster written to disk
Note

This function measures terrain ruggedness by calculating the vector ruggedness measure

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


Examples

```r
library(raster)
data(elev)
vr3 <- vrm(elev)
vr5 <- vrm(elev, s=5)
plot(stack(vr3, vr5))
```

Description

Removes extreme outliers using a winsorization transformation

Winsorization is the transformation of a distribution by limiting extreme values to reduce the effect of spurious outliers. This is done by shrinking outlying observations to the border of the main part of the distribution.

Usage

```r
winsorize(
x,
min.value = NULL,
max.value = NULL,
p = c(0.05, 0.95),
na.rm = FALSE
)
```
Arguments

- **x**: A numeric vector
- **min.value**: A fixed lower bounds, all values lower than this will be replaced by this value. The default is set to the 5th-quantile of x.
- **max.value**: A fixed upper bounds, all values higher than this will be replaced by this value. The default is set to the 95th-quantile of x.
- **p**: A numeric vector of 2 representing the probabilities used in the quantile function.
- **na.rm** (FALSE/TRUE) should NAs be omitted?

Value

A transformed vector the same length as x, unless na.rm is TRUE, then x is length minus number of NA's

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

References


Examples

```r
set.seed(1234)
x <- rnorm(100)
winsorize(x)

plot(x, type="l", main="Winsorization transformation")
lines(winsorize(x), col="red", lwd=2)
legend("bottomright", legend=c("Original distribution","With outliers removed"),
1ty=c(1,1), col=c("black","red"))

# Behavior with NA value(s)
x[4] <- NA
winsorize(x)  # returns x with original NA's
winsorize(x, na.rm=TRUE)  # removes NA's
```
wt.centroid

Weighted centroid

Description

Creates centroid of [x,y] coordinates based on a weights field

Usage

wt.centroid(x, p, sp = TRUE)

Arguments

x sp SpatialPointsDataFrame class object
p Weights column in x@data slot
sp Output sp SpatialPoints class object (TRUE | FALSE)

Value

A vector or an sp class SpatialPoints object of the weighted coordinate centroid

Note

The weighted centroid is calculated as: [Xw]=[X]*[p], [Yw]=[Y]*[p], [sXw]=SUM[Xw], [sYw]=SUM[Yw], [sP]=SUM[p] wX=[sXw]/[sP], wY=[sYw]/[sP] where; X=X COORDINATE(S), Y=Y COORDINATE(S), p=WEIGHT

Depends: sp

Examples

require(sp)
data(meuse)
coordinates(meuse) = ~x+y
wt.copper <- wt.centroid(meuse, 'copper', sp=TRUE)
wt.zinc <- wt.centroid(meuse, 'zinc', sp=TRUE)
plot(meuse, pch=20, cex=0.75, main='Weighted centroid(s)')
points(wt.copper, pch=19, col='red', cex=1.5)
points(wt.zinc, pch=19, col='blue', cex=1.5)
box()
legend('topleft', legend=c('all','copper', 'zinc'),
pch=c(20,19,19),col=c('black','red','blue'))
Description

Polygon zonal statistics of a raster

Usage

zonal.stats(x, y, stats = c("min", "mean", "max"))

Arguments

x  Polygon object of class SpatialPolygonsDataFrame
y  rasterLayer object of class raster
stats  Statistic or function

Value

data.frame, nrow(x) and ncol of function results

Note

This function calculates the zonal statistics between a polygon vector object and a raster. This provides the advantage of being able to accept any custom function, passed to the 'stats' argument. Please note that any custom function needs to have a 'na.rm' argument.

Author(s)

Jeffrey S. Evans <jeffrey_evans@tnc.org>

Examples

library(raster)
library(sp)

# skewness function
skew <- function(x, na.rm = FALSE) {
  if (na.rm) x <- x[!is.na(x)]
  sum( (x - mean(x))^3 ) / ( length(x) * sd(x) * 3 )
}

# percent x >= p function
pct <- function(x, p=0.30, na.rm = FALSE) {
  if ( length(x[x >= p]) < 1 ) return(0)
  if ( length(x[x >= p]) == length(x) ) return(1)
  else return( length(x[x >= p]) / length(x) )
}
# create some example data
p <- raster(nrow=10, ncol=10)
p[] <- runif(ncell(p)) * 10
p <- rasterToPolygons(p, fun=function(x){x > 9})
r <- raster(nrow=100, ncol=100)
r[] <- runif(ncell(r))
plot(r)
plot(p, add=TRUE, lwd=4)

# run zonal statistics using skew and pct functions
z.skew <- zonal.stats(x = p, y = r, stats = "skew")
z.pct <- zonal.stats(x=p, y=r, stats = "pct")
( z <- data.frame(ID = as.numeric(as.character(row.names(p@data))),
  SKEW=z.skew, PCT=z.pct) )
Index

annulus.matrix, 5
ants, 6
autocov_dist, 72

background, 6
bearing.distance, 8
breeding.density, 9
built.index, 10

calc, 149, 158
cellnumbers, 97
cgls_urls, 12
chae, 14
chen, 15
clara, 83
class.comparison, 16
classBreaks, 18
collinear, 19
combine, 20
concordance, 22
conf.interval, 23
cor.data, 24
correlogram, 24
cross.tab, 25
crossCorrelation, 27
crosstab, 26
csi, 30
curvature, 31
dahi, 32
date_seq, 33
daymet.point, 34
daymet.tiles, 35
DAYMET_tiles, 36
dispersion, 37
dissection, 38
distance, 131
distanceFromPoints, 131
divergence, 39
download.daymet, 39
download.hansen, 40
download.prism, 42
effect.size, 42
elev, 43
erase.point, 44
explode, 45
extract.vertices, 46
focal, 158
focal.lmetrics, 47
fuzzySum, 48
gaussian.kernel, 49
gBuffer, 50
gContains, 155
gCovers, 155
geo.buffer, 50
gIntersects, 155
group.pdf, 51
g_touches, 155
gWithinDistance, 155

hclust, 56
hexagons, 52
hli, 53
hsp, 54
hybrid.kmeans, 55

idw.smoothing, 56
impute.loess, 57
insert, 58
insert.values, 59
is.empty, 60
is.whole, 61

distinguish, 131
kl.divergence, 64
kmeans, 55, 56

kde.2D, 62
kendall, 62
kendallTrendTest, 121
k2, 51
kmeans, 55, 56

179
INDEX

knn, 65
land.metrics, 66
local.min.max, 67
loess, 96, 149
loess.boot, 68
loess.ci, 70
logistic.regression, 71
lrm, 72
max.extent, 74
modified.ttest, 124
moments, 75
morans.plot, 76
mwCorr, 78
nn2, 66
nni, 78
nth.values, 79
o.ring, 80
oli.asw, 81
optimal.k, 83
optimized.sample.variance, 84
outliers, 85
overlap, 86
overlay, 121
pam, 83
parea.sample, 87
parse.bits, 89
partial.cor, 90
plot.effect.size, 92
plot.loess.boot, 92
point.in.poly, 93
poly.regression, 95
polygon_extract, 97
polyPerimeter, 99
pp.subsample, 100
print.cross.cor, 102
print.effect.size, 102
print.loess.boot, 103
proximity.index, 103
pseudo.absence, 104
pu, 107
quadrats, 109
random.raster, 110
raster.change, 112
raster.deviation, 114
raster.downscale, 115
raster.entropy, 117
raster.gaussian.smooth, 118
raster.invert, 119
raster.kendall, 120
raster.mds, 122
raster.modified.ttest, 123
raster.moments, 125
raster.transformation, 126
raster.vol, 127
raster.Zscore, 128
rasterCorrelation, 129
rasterDistance, 130
remove.holes, 132
rm.ext, 133
rotate.polygon, 133
sa.trans, 134
sample.annulus, 135
sample.line, 136
sample.poly, 138
sampleTransect, 139
sar, 140
se.news, 141
separability, 141
sg.smooth, 143
shannons, 144
shift, 146
similarity, 146
smooth.time.series, 148
sobal, 149
sp.kde, 151
sp.na.omit, 153
spatial.select, 154
spectral.separability, 156
spherical.sd, 157
srr, 159
stratified.random, 160
subsample.distance, 161
summary.cross.cor, 163
summary.effect.size, 163
summary.loess.boot, 164
swvi, 164
time_to_event, 166
topo.distance, 168
tpi, 169
trasp, 170
INDEX

- trend.line, 171
- tri, 172
- vrm, 173
- winsorize, 174
- writeRaster, 32
- wtcentroid, 176
- zonal.stats, 177