Package ‘lbmech’

July 9, 2023

Type Package

Title A Package to Study the Mechanics of Landscape and Behavior

Version 0.5.1

Author XXXXX

Maintainer XXXXX <XXXXX>

Description A geospatial package to study the adaptive mechanics of landscape and behavior, including at present: (1) energetic and efficient least-cost analysis; (2) local indicators of dispersion; and (3) interpolation of agricultural productivity data.

License CC BY-NC 4.0

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Imports stringr,
igraph,
quantreg,
reshape2,
stats,
utils,
methods,
sf,
geosphere,
tmaptools,
gtools,
elevatr,
fst,
raster,
geodata,
rworldmap,
scales,
matrixStats,
ggplot2,
cowplot,
leaflet,
archive

Depends terra,
data.table,
R (>= 2.10)

Suggests R.utils,
rmarkdown

R topics documented:

`approxMap`	*Create an approximate mapping function*

Description

Define a function that generates an approximation function for a given subset of data. This is designed to be used to predict values in one data.table using values in another via the j slot. See examples.

Usage

```
approxMap(
  dt,
  id.val,
  x = "x",
  y = "y",
  id.col = "ID",
  rand.val = NULL,
  rand.range = c(0, 1),
  FUN = stats::approxfun,
  ...
)
```

Arguments

`dt`	A data.table containing the observed data
`id.val`	The value to look up in `dt[[id.col]]`. When used in a data.table `target`'s `j` slot, this would be the name of the column shared between `dt` and `target` without quotes.
`x`	A character string representing the name of the independent variable column in `dt`. Default is `x = 'x'`.
`y`	A character string representing the name of the dependent variable column in `dt`. Default is `y = 'y'`.
`id.col`	A character string representing the name of the group column in `dt`. Default is `id.col = 'ID'`
`rand.val`	A character vector with values present in `dt[[id.col]]` for which to return a random value. Used for psuedo-observations where the outcome is known regardless of the value of the predicted variable.
`rand.range`	A vector of two numbers repretenting t he range of potential values for `rand.range`. Default is `rand.range = c(0,1)`.
`FUN`	The approximation function. Default is `approxfun`, but any function that accepts a two-column data.frame is acceptable.
`...`	Additional parameters to pass to `FUN`.

Examples

```
# Generate a data.table with different observations for different categories
data <- data.table(ID = rep(c("A","B","C"), each = 10),
                   x  = runif(30),
                   y  = runif(30, min = 10, max = 20))

# Create a target data.table
target <- data.table(ID = rep(c("A","B","C"), each = 101),
                     x = rep(seq(0,1,length.out=101), each = 3))

target[, y := approxMap(data, id.val = ID, rule = 2, na.rm=TRUE)(x),by='ID']
```

average *Compute weighted means and medians.*

Description

Calculate the weighted average that maximizes the maximum likelihood estimator

Usage

```
average(x, w = rep(1, length(x)), type = "mean", na.rm = FALSE)
```

Arguments

x	A numeric vector with the values to be averaged.
w	A vector of weights of the same length as x.
type	A character string, either: 'mean' or 'L2' (the default), such that a weighted mean is computed using weightedMean; or 'median' or 'L1', such that a weighted median is computed using weightedMedian.
na.rm	Should NA values be ignored? Default is na.rm = FALSE.

Value

A numeric value, representing the weighted mean or median

Examples

```
# Create dummy values
x = runif(10,1,100)
w = runif(10,1,10)

# Compute weighted mean
average(x, w, type = 'mean')

# Compute weighted median
average(x, w, type = 'median')
```

`calculateCosts` *Calculate movement costs according to a cost function*

Description

A function that for a given world of possible movement calculates the transition cost for each in terms of a user-defined cost function

Usage

```
calculateCosts(
  tiles = NULL,
  costFUN = energyCosts,
  dir = tempdir(),
  costname = deparse(substitute(costFUN)),
  ...
)
```

Arguments

`tiles`	A character vector–such as the output to `whichTiles`—containing the unique tile IDs for sectors that should be in the workspace. Default is `NULL`.
`costFUN`	A cost function such as (`timeCosts` or `energyCosts`). The input to such a function should be a `data.table` object with column names present in the makeWorld file (initially `c('x_i','x_f','y_i','y_f','z_i','z_f','dz','dl','dr')`), and the output should be an data.table object with the same number of rows and the desired output variables as the only column names. Constants can be passed in the `...` slot. Default is `costFUN = energyCosts`
`dir`	A filepath to the directory being used as the workspace, the same one instantiated with `defineWorld`. Default is `tempdir()` but unless the analyses will only be performed a few times it is highly recommended to define a permanent workspace. `whichTiles`—containing the unique tile IDs for sectors that should be in the workspace. Default is `NULL`.
`costname`	A name to save the cost call parametrs. Default is the name of the costFUN variable.
`...`	Additional parameters to pass to `costFUN`

Value

An `.fst` file for each sector named after its sector id stored in the `/World/Diff` directory, and/or a data.table object (depending on the output parameter) containing a data.table with at least eight columns

(1) `$from` The "x,y"-format coordinates of each possible origin cell

(2) `$to` The "x,y"-format coordinates of each possible destination cell

(3) `$dz` The change in elevation between the `from` and `to` cells

(4) `$x_i` The numeric x coordinate of the origin cell

(5) `$y_i` The numeric y coordinate of the origin cell

(6) `$dl` The planimetric distance between the `from` and `to` cells

(7) `$dr` The 3D distance between the `from` and `to` cells

(8+) The output cost variables

Examples

```
# Generate a DEM
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"

# Export it so it doesn't just exist on the memory
dir <- tempdir()
writeRaster(dem, paste0(dir,"/DEM.tif"),overwrite=TRUE)


# Import raster, get the grid
dem <- rast(paste0(dir,"/DEM.tif"))
grid <- makeGrid(dem = dem, nx = n, ny = n, sources = TRUE)

# Select all tiles that exist between x = (12000,16000) and y = (32000,36000)
tiles <- ext(c(12000,16000,32000,36000))
tiles <- as.polygons(tiles)
crs(tiles) <- crs(grid)
tiles <- whichTiles(region = tiles, polys = grid)

# Make a world but limit it to the DEM grid size
defineWorld(source = grid, cut_slope = 0.5,
            res = res(dem), dir = dir, overwrite=TRUE)

# Calculate the costs within the world
calculateCosts(tiles = tiles, dir = dir,
m = 70, v_max = 1.5, BMR = 76, k = 3.5, s = 0.05, l_s = 1,
L = 0.8)
```

`defineWorld` *Define the topographic landscape*

Description

Function that defines the grid that can be traversed–the "world"–as well as the cells that can be accessed from each individual cell. This is the most time-intensive function.

Usage

```
defineWorld(
  source,
  source_id = "TILEID",
  z_min = NULL,
  grid = NULL,
  proj = NULL,
  grid_id = "TILEID",
  cut_slope = Inf,
  directions = 16,
  neighbor_distance = 10,
  z_fix = NULL,
  unit = "m",
  vals = "location",
  precision = 2,
  dist = "proj",
  r = 6378137,
  f = 1/298.257223563,
  b = 6356752.3142,
  FUN = NULL,
  sampling = "bilinear",
  water = FALSE,
  water_speed = "speed",
  uv = TRUE,
  priority = "water",
  overwrite = FALSE,
  filt = 0,
  dir = tempdir(),
  cols = c("x_i", "y_i", "dz", "dl", "dr"),
  ...
)
```

Arguments

`source`	An object of class SpatVector, or potential inputs to `makeGrid` that define the paths to the original source files of the desired rasters in the same format as the output of the `makeGrid(sources = TRUE)`. If the object is NOT a already a polygon of the appropriate format, set `source_id = NULL` and add the necessary `makeGrid` parameter
`source_id`	A character string representing the name of the column in the source polygon containing the unique Tile IDs. Default is `source_id = 'TILEID'`
`z_min`	The minimum allowable elevation. Useful if DEM source includes ocean bathymetry as does the SRTM data from AWS. Default is `z_min = NULL`, but set to `0` for SRTM data.
`grid`	An object of class SpatVector representing the partitioning grid for the maximum possible travel area. Smaller grids increase the amount of area that can be read into the memory, but require more I/O operations. Default is `grid = source`.
`proj`	A crs object or character string representing the output projection. Default projection is `proj = crs(polys)` unless a 'z_fix' or 'proj' is provided, in which case the latter is ignored. Great care should be employed to ensure that the projection is conformal and in meters.
`grid_id`	A character string representing the name of the column in the `grid` polygon containing the unique Tile IDs. Default is `tile_id = 'TILEID'`
`cut_slope`	A number representing the dimensionless maximum slope of ascent/descent. To ignore, set `cut_slope = Inf`.

`directions`	One of the integers `c(4, 8, 16)`, the character string `'bishop'`,or a neighborhood matrix. Default is `directions = 16`, implying that all 'knight and one-cell queen moves' are permissible movements on the grid. See `adjacent`.
`neighbor_distance`	An integer representing the distance in meters that tiles are buffered. In other words, to ensure that all transitions in the 'world' are recorded, files for each tile will contain a number of observations that fall outside of the tile in other ones. Default is 100 m, but adjust on raster size.
`z_fix`	A SpatRaster or Raster* that will define the resolution, origin, and projection information for the entire "world" of possible movement. Note that it does NOT need the same extent. Default resolution is 5, and offset is 0. Default projection is `proj = crs(polys)` unless a ‘z_fix‘ or ‘proj‘ is provided, in which case the latter is ignored. Great care should be employed to ensure that the projection is conformal and in meters.
`unit`	One of `c("m", "km", "ft", "mi")`, representing the unit of the DEM. All will be converted to meters, which is the default.
`vals`	A character string or a SpatRaster or Raster* object. Ignored unless the `polys` parameter is a polygon NOT the output of the `makeGrid` function as the default is the character string `'location'`, AND the appropriate world `.gz` file is NOT present in the workspace directory. In which case it must represent either the original DEM or a character string with the column representing the DEM filepath or URL.
`precision`	An integer representing the number of decimals to retain in the x and y directions. For grid sizes with nice, round numbers precisions can be low. This factor is controled by `rast`. Default is 2.
`dist`	A character string representing the way distances should be calculated. Default is `dist = 'proj'` for the default projection units, but the following geodesic methods are also available: `c('haversine','cosine','karney','meeus','vincentyEllipsoid'` The developer recommends `'karney'`.
`r`	The earth's radius. Employed only if one of the geodesic methods is used. Default is `r = 6378137` for WGS1984.
`f`	The earth's ellipsoidal flattening. Employed only if `dist %in% c('karney','meeus','vincentyEll` Default is `f=1/298.257223563` for WGS1984
`b`	The earth's semiminor axis. Employed only if `dist = 'vincentyEllipsoid'`. Default is `b=6356752.3142` for WGS1984.
`FUN`	Function to deal with overlapping values for overlapping sectors. Default is NA, which uses `merge`. To use `mosaic`, provide a compatible function.
`sampling`	How to resample rasters. Default is `'bilinear'` interpolation, although `'ngb'` nearest neighbor is available for categorical rasters.
`water`	Optional. One of (1) SpatialPolygon* or SpatVector polygon representing the area covered by water, in which case `water_speed` must also be provided; (2) A RasterLayer or single-layer SpatRaster representing the speed of water at a given location. Flow direction will be calculated from the world's DEM; or (3) A two layer SpatRaster or Raster* object, representing either the horizontal/vertical components of water velocity (in m/s) or the absolute water speed and flow direction (in that order; see `uv`).
`water_speed`	A character representing the column name in `water` that contains the average speed of the water in m/s. Required if `water` is a polygon.

`uv`	Logical. If `TRUE` (the default), a two-layer raster input for `water` is taken to be the horizontal and vertical components of water velocity. If false, a two-layer raster input for `water` is taken to be the water speed and flow direction.
`priority`	One of `'water'` (the default), or `'land'`, indicating whether land values or water values should take precedence when values for a given location exist in both datasets.
`overwrite`	If a directory with a `World` subdirectory already exists, should the latter be overwritten? Default is `overwrite = FALSE`.
`filt`	Numeric. Size of moving window to apply a low-pass filter. Default is `filt = 0`. Ignored unless the tiles need to be generated from the raw source files.
`dir`	A filepath to the directory being used as the workspace. Default is `tempdir()` but unless the analyses will only be performed a few times it is highly recommended to define a permanent workspace.
`cols`	A character vector. Default is c("x_i","y_i","dz","dl","dr"), and dictates which columns are returned but final values for x and y and final/initial z values are also available
`...`	Additional arguments to pass to `fix_z`.

Details

It first checks to see if the required files contained within a `'/World/'` directory have already been created in the `dir` workspace. If not it creates them; if the `'/World/'` directory exists though, then an error is thrown (default) OR the user has the option to overwrite the directory.

The default parameters are sufficient for a workflow involving calculating costs with the `calculateCosts` function.

Value

A `/World/` directory, containing `/Loc/`, `/Diff/`, and `/Raw/`, directories where cropped and transformed files will be stored, and `/callVars.gz/`, `/z_sources/`, `/z_grid/`, `/z_fix.fst/`, and `/z_fix.fstproj/` files defining the world.

Examples

```
# Generate a DEM
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"

# Export it so it doesn't just exist on the memory
dir <- tempdir()
writeRaster(dem, paste0(dir,"/DEM.tif"),overwrite=TRUE)


# Import raster, get the grid
dem <- rast(paste0(dir,"/DEM.tif"))
grid <- makeGrid(dem = dem, nx = n, ny = n, sources = TRUE)
```

```
# Select all tiles that exist between x = (12000,16000) and y = (32000,36000)
tiles <- ext(c(12000,16000,32000,36000))
tiles <- as.polygons(tiles)
crs(tiles) <- crs(grid)
tiles <- whichTiles(region = tiles, polys = grid)

# Make a world but limit it to the DEM grid size
defineWorld(source = grid, cut_slope = 0.5,
            res = res(dem), dir = dir, overwrite=TRUE)
```

`dispersionIndex` *Calculate Gini and Inoua indexes*

Description

Calculate weighted versions of the Gini and Inoua (2021) indexes as originally defined in the econometric literature, using the half mean relative distance method.

Usage

```
dispersionIndex(
  x,
  index = "gini",
  w = rep(1, length(x)),
  weight.mean = TRUE,
  inverse = FALSE,
  max.cross = .Machine$integer.max,
  pb = FALSE
)

gini(...)

inoua(...)
```

Arguments

`x`	A vector of values
`index`	A character string, either `'gini'`, or `'inoua'`, representing whether distances are calculated in L1 or L2 space, respectively
`w`	A vector of weights with the same length as `x`
`weight.mean`	Logical. Should the mean values be weighted, or does the global depend exclusively on the observations? Default is `TRUE`.
`inverse`	Logical. Should the value for the inverse weights be calculated as well using binary decomposition? Default is `FALSE`. This rarely makes sense if the weights are population-based, but it does if they're probability-based.
`max.cross`	When processing, what is the maximum number of rows that an internal data.table can have? This is generally not a concern unless the number of observations approaches `sqrt(.Machine$integer.max)`–usually about 2^31 for most systems. Lower values result in a greater number of chunks thus allowing larger data.sets to be calculated

`pb`	Logical. Should a progress bar be displayed? Default is `FALSE`, although if a large dataset is processed that requires adjusting `max.cross` this can be useful
`...`	Parameters to pass on to `dispersionIndex`.

Value

A numeric of length 1 (if `inverse = FALSE`) or 2 (if `inverse = TRUE`) representing the requested index.

References

Inoua, Sabiou (2021). "Beware the Gini Index! A New Inequality Measure." *ESI Working Paper* 21-18, https://digitalcommons.chapman.edu/esi_working_papers/355/.

Examples

```
# Generate dummy observations
x <- runif(10, 1, 100)
n <- runif(10, 0, 10)

# Calculate Gini index
gini(x)

# Calculate weighted Inoua index
inoua(x, w = n)
```

downsampleXYZ *Downsample high-resolution GPS data*

Description

Downsample high-resolution *(x,y,t)* or *(x,y,z,t)* data to match a minimum spatial resolution.

Usage

```
downsampleXYZ(
  data,
  t_step,
  t_cut = t_step * 10,
  x = "x",
  y = "y",
  z = NULL,
  t = "t",
  ID = "ID"
)
```

Arguments

`data`	A data.frame or something coercible to a data.table containing all observations

`t_step`	The smallest allowable median time interval between observations. Any track with a median value below this will be resampled to a track of equally-spaced observations with time difference `t_step`. This should be selected based on the parameters that will be used to generate a `world` object such that each successive step is likely to fall outside of the range of possible raster transitions. For example, for a given source `dem` in `makeGrid`, a given `distances = 16` in `makeWorld` (implying that `contiguity = 2`), and an estimated animal maximum velocity of `v_max = 1.5` m/s, t_step should be at least `t_step = res(dem) / v_max * (contiguity + 1) * sqrt(2)`
`t_cut`	A numeric. Any gap exceeding this value in seconds in a given track will be treated as the start of a new segment. Default is `t_step * 10`.
`x`	A character string representing the data column containing the 'x' coordinates in any projection.
`y`	A character string representing the data column containing the 'y' coordinates in any projection.
`z`	Optional. A character string representing the data column containing the 'z' elevations.
`t`	A character string representing the data column containing the time values, either as a numeric of successive seconds or as a date time string
`ID`	A character string representing the data column containing the unique ID for each observed trajectory. In other words, each set of points for each continuous observation for each observed individual would merit a unique id.

Value

A data.table, containing the original input data but with all overly-high resolution tracks down-sampled to an acceptable rate of observations and column names prepared for the `getVelocity` function.

Examples

```
# Generate fake data with x,y coordinates, z elevation, and a t
# column representing the number of seconds into the observation
data <- data.table(x = runif(10000,10000,20000),
                   y = runif(10000,30000,40000),
                   z = runif(10000,0,200),
                   t = 1:1000,
                   ID = rep(1:10,each=1000))

# Set the minimum value at 3 seconds
data <- downsampleXYZ(data = data, t_step = 3, z = 'z')
```

`energyCosts` *Calculate time and energy costs*

Description

A function that for a given world of possible movement calculates the transition cost for each in terms of a pre-defined time, work, and energy cost functions. `energyCosts` calls `timeCosts` if columns named `'dt'` and `'dl_t'` are not present in the input data.table

Usage

```
timeCosts(DT, v_max, k, s, row_speed = NULL, water = FALSE)

energyCosts(
  DT,
  method = "kuo",
  m = NULL,
  BMR = NULL,
  g = 9.81,
  epsilon = 0.2,
  l_s = NULL,
  L = NULL,
  gamma = NULL,
  time = timeCosts,
  water = FALSE,
  row_work = NULL,
  ...
)
```

Arguments

`DT`	A data.table containing at minimum columns 'dz' representing the change in elevation and 'dl' representing planimetric distance
`v_max`	The maximum velocity of the animal moving across the landscape, in meters per second; see `getVelocity`.
`k`	The topographic sensitivity factor; see `getVelocity`.
`s`	The dimensionless slope of maximum velocity; see `getVelocity`.
`row_speed`	How fast can a person move over water? Default is `row_speed = NULL`, but required if `water` are provided.
`water`	Logical. If `FALSE` (the default), movement costs are calculated as if over land. If `water = TRUE`, movement costs are calculated considering moving water.
`method`	A character string for the method that energy costs per unit stride should be calculated. One of `method %in% c('kuo','heglund','pontzer','oscillator')`; see references.
`m`	The mass of the animal moving across the landscape, in kilograms.
`BMR`	The base metabolic rate of the object moving across the landscape in Joules per second.
`g`	The acceleration due to gravity, in meters per second per second. Default is g = `9.81` m/s^2, as for the surface of planet Earth.
`epsilon`	The biomechanical efficiency factor for an animal moving across the landscape. Default is `epsilon = 0.2`.
`l_s`	The average stride length, in meters. Required for `method = 'kuo'`, `'pontzer'` or `'oscillator'`, ignored for `'heglund'`
`L`	The average leg length. Required for `method = 'kuo'`, ignored for `'heglund'`, `'pontzer'` and `'oscillator'`.
`gamma`	The fractional maximal deviation from average velocity per stride. Required for `method = 'oscillator'`, ignored otherwise

`time`	The method by which time costs should be calculated by `energyCosts` should `c('dt','dl_t')` not be column names in the input data.table. Default is `time = timeCosts`.
`row_work`	How much work in joules per second does a person use to move over water? Default is `row_work = NULL`, but required if `water` is provided.
`...`	Additional parameters to pass to `timeCosts`

Value

For `timeCosts`, A data.table object with two columns:

(1) `$dl_t` The predicted walking speed in meters per second when walking between the `from` and `to` cells

(2) `$dt` The predicted amount of time spent walking between the `from` and `to` cells

For `energyCosts`, a data.table object with five columns:

(1) `$dt` The predicted amount of time spent walking between the `from` and `to` cells

(2) `$dU_l` The predicted work against gravitational potential energy in Joules when walking between the `from` and `to` cells

(3) `$dK_l` The predicted kinematic work in Joules when walking between the `from` and `to` cells

(4) `$dW_l` The total predicted energy lost due to biomechanical work when walking between the `from` and `to` cells.

(5) `$dE_l` The net metabolic expenditure exerted when walking between the `from` and `to` cells.

References

Heglund, N. C., Cavagna, G. A., and Taylor, C. R. (1982). "Energetics and mechanics of terrestrial locomotion. III. Energy changes of the centre of mass as a function of speed and body size in birds and mammals." *Journal of Experimental Biology* 97(1):41-56. doi:10.1242/jeb.97.1.41.

Kuo, Arthur D. (2007). "The six determinants of gait and the inverted pendulum analogy: A dynamic walking perspective." *Human Movement Science* 26(4):617-656. doi:10.1016/j.humov.2007.04.003.

Pontzer, Herman (2016). "A unified theory for the energy cost of legged locomotion" *Biology Letters* 12(2):20150935. doi:10.1098/rsbl.2015.0935

Examples

```
# Generate a DEM
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"

# Export it so it doesn't just exist on the memory
dir <- tempdir()
writeRaster(dem, paste0(dir,"/DEM.tif"),overwrite=TRUE)


# Import raster, get the grid
```

```
dem <- rast(paste0(dir,"/DEM.tif"))
grid <- makeGrid(dem = dem, nx = n, ny = n, sources = TRUE)

# Select all tiles that exist between x = (12000,16000) and y = (32000,36000)
tiles <- ext(c(12000,16000,32000,36000))
tiles <- as.polygons(tiles)
crs(tiles) <- crs(grid)
tiles <- whichTiles(region = tiles, polys = grid)

# Make a world but limit it to the DEM grid size
defineWorld(source = grid, cut_slope = 0.5,
            res = res(dem), dir = dir, overwrite=TRUE)

# Calculate the energetic and temporal costs
calculateCosts(costFUN = energyCosts,
tiles = tiles, dir = dir,
m = 70, v_max = 1.5, BMR = 76, k = 3.5, s = 0.05, l_s = 1,
L = 0.8)
```

fix_z *Define the sampling grid*

Description

Create a raster that can be used to define the resolution, origin, and projection to be employed for all least-cost analyses. If a source DEM has such properties you may use that.

Usage

```
fix_z(proj, res = 5, dx = 0, dy = 0)
```

Arguments

proj	A crs object or character string containing projection information. Should be conformal and in meters.
res	A numeric of length one or two nrepresenting the spatial resolution. Default is 5.
dx	The horizontal offset from the origin (see origin). Default is 0 (this does not correspond to an origin of zero however).
dy	The vertical offset from the origin (see origin). Default is 0 (this does not correspond to an origin of zero however).

Value

A SpatRaster object consisting of four cells, with resolution res and the origin at x = nx and y = ny.

Examples

```
projection <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"
z_fix <- fix_z(res = 2, proj = projection)
```

`getCoords` *Get "x,y" coordinates in appropriate format*

Description

Function to get the coordinates in "x,y" format for a given set of points

Usage

```
getCoords(
  data,
  proj = NULL,
  x = "x",
  y = "y",
  z_fix = NULL,
  precision = 2,
  ...
)
```

Arguments

`data`	An object of class data.table or something coercible to it containing the coordinates needing conversion, or a SpatialPointsDataFrame.
`proj`	A crs object or character string representing the output projection. Required unless `z_fix` is provided in which case `proj` is ignored.
`x`	A character vector representing the column containing the 'x' coordinates. Required if `data` is not SpatialPointsDataFrame.
`y`	A character vector representing the column containing the 'y' coordinates. Required if `data` is not SpatialPointsDataFrame.
`z_fix`	A SpatRaster with the same origin and resolution as the `z_fix` used to generate the 'world' with `makeWorld`.
`precision`	An integer representing the number of decimals to retain in the x and y directions. For grid sizes with nice, round numbers precisions can be low. This factor is controled by `rast` and must be the same as the one used to generate the 'world' with `makeWorld`. Default is 2.
`...`	Additional arguments to pass to `fix_z`.

Value

A vector containing the requested coordinates in appropriate format in the same order as the input data.

Examples

```
# Generate a DEM
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"

# Generate five random points that fall within the DEM
points <- data.table(x = runif(5, ext(dem)[1], ext(dem)[2]),
                     y = runif(5, ext(dem)[3], ext(dem)[4]))

# Get the coordinates
points$Cell <- getCoords(points, z_fix = dem)
```

getCosts *Get cost of travel*

Description

A function that calculates the cost to travel between two sets of points. This can be between two circumscribed sets of points, or one circumscribed one ('nodes') and all other points on the landscape.

Usage

```
getCosts(
  region,
  from,
  to = NULL,
  id = "ID",
  dir = tempdir(),
  x = "x",
  y = "y",
  destination = "pairs",
  polygons = "centroid",
  costs = c("dt", "dW_l", "dE_l"),
  direction = "both",
  output = "object",
  outname = deparse(substitute(from)),
  overwrite = FALSE,
  ...
)
```

Arguments

`region`	A SpatVector, Spatial* or Raster* representing the area of maximum movement
`from`	One of data.frame, data.table, SpatVector, SpatialPointsDataFrame, or SpatialPolygonsDataFrame representing the origin locations. If `to = NULL` and `destination = 'pairs'`, this will also be used as the `to` parameter. If it is a polygon, the location will be controlled by the `polygons` parameter.
`to`	One of data.frame, data.table, SpatVector SpatialPointsDataFrame, or SpatialPolygonsDataFrame representing the origin locations. Optional if `destination = 'pairs'`, ignored if `destination = 'all'`.

`id`	Character string representing the column containing the unique IDs for for each location in the `from` (and `to`) objects.
`dir`	A filepath to the directory being used as the workspace. Default is `tempdir()` but unless the analyses will only be performed a few times it is highly recommended to define a permanent workspace.
`x`	A character vector representing the column containing the 'x' coordinates. Required if `data` is not Spatial*.
`y`	A character vector representing the column containing the 'y' coordinates. Required if `data` is not Spatial*.
`destination`	One of `'pairs'` or `'all'`. If `'pairs'`, a distance matrix will be generated between every pair of locations in `from`, or every pair of locations between `from` and `to`. If `'all'`, rasters will be generated for each node representing the cost to travel to every cell in the given `world`.
`polygons`	One of `c('polygon','centroid','center')`. Ignored unless `from` and/or `to` are polygons. If `polygons = 'centroid'` (the default), the destinations are calculated to the centroid of the polygon whether or not it lies within the polygon itself. If `polygons = 'center'`, distances are calculated to the point within the polygon closest to the centroid. If `polygons = 'polygons'`, distances are calculated to any point within the polygon—in essence, the polygon acts as a giant node permitting costless movement within its bounds. This is generally not consistent with real-world scenarios and for that reason is not the default.
`costs`	A character vector containing any combination of the strings of differential values present in the environment (see `calculateCosts` function; default is `costs = c("dt","dW_l","dE_l")` anticipating the use of `calculateCosts(costFUN = energyCosts)`.
`direction`	A character vector containing one or both of `c("in","out")` or the singular string 'both'. This determines whether costs to or from the nodes are calculated. Ignored for `destination = 'pairs'`.
`output`	A character vector containing one or both of `c("object","file")`. If `"object"` is included, then a list of RasterStacks will be returned. If `"file"` is included, then the appropriate cost rasters will be saved to the workspace directory `dir`. Ignored if `destination = 'pairs'`.
`outname`	A character vector describing the name of the set of input nodes for raster analysis. Ignored unless `'file' %in% output`, in which case the files will be stored in a file named as such. Default is the name of the `from` input, which can lead to files being overwritten if vector sources are arbitrarily changed.
`overwrite`	Should any output files with the same outname be overwritten? Default is `overwrite = FALSE`.
`...`	Additional arguments to pass to `importWorld`.

Details

There are four possible workflows:

(1) If you simply desire the distance between two sets of points, provide entries for `from` and `to` (or just `from` if the interest is in all distances between locations in that object). Output is a distance matrix. The computational time for this operation is comparable to generating a raster for the distance to *all* cells in the world (unless all of the locations in the object are close to each other). So unless the operation is to be done multiple times, it is highly recommended to generate the rasters as below and extract values

(2) If you wish to generate a RasterStack of costs from and/or to all nodes in the `from` object, set the `output = 'object'` and `destination = 'all'`.

(3) You may also save the rasters as a series of `.tif` files in the same workspace directory as the transition `.gz` tensor files and the cropped/downloaded DEMs. This allows us to use `getCosts` within a loop for large numbers of origin nodes without running into random access memory limitations. Do this by setting `output = 'file'` and `destination = 'all'`.

(4) You may perform (2) and (3) simultaneously by setting `output == c('file','object')` and `destination = 'all'`.

Value

A list, with entries for each combination of selected `costs` and `directions`. The object class of the list entries depends on the `destination` and `output` parameters:

(1) If `destination = 'pairs'`, entries are distance matrices. (2) If `destination = 'all'` and `'object' %in% output`, entries are RasterStacks with each Raster* representing the cost to/from each node. (3) If `destination = 'all'` and `output == 'file'`, no list is output.

Moreover, if `file %in% output`, then the cost rasters are saved in the workspace directory.

Examples

```
#### Example 1:
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"

# Export it so it doesn't just exist on the memory
dir <- tempdir()
writeRaster(dem, paste0(dir,"/DEM.tif"),overwrite=TRUE)


# Import raster, get the grid
dem <- rast(paste0(dir,"/DEM.tif"))
grid <- makeGrid(dem = dem, nx = n, ny = n, sources = TRUE)

# Select all tiles that exist between x = (12000,16000) and y = (32000,36000)
tiles <- ext(c(12000,16000,32000,36000))
tiles <- as.polygons(tiles)
crs(tiles) <- crs(grid)
tiles <- whichTiles(region = tiles, polys = grid)

region <- grid[8,]
# Generate five random points that fall within the region
points <- data.table(ID = 1:5,
                     x = runif(5, ext(region)[1], ext(region)[2]),
                     y = runif(5, ext(region)[3], ext(region)[4]))

# Make a world but limit it to the DEM grid size
defineWorld(source = grid, cut_slope = 0.5,
            res = res(dem), dir = dir, overwrite=TRUE)
```

```
# Get time and work costs between points
costMatrix <- getCosts(grid[8,], from = points, dir = dir,
                       costs = c('dt','dW_l'), costFUN = energyCosts,
                       m = 70, v_max = 1.5, BMR = 76, k = 3.5, s = 0.05, l_s = 1,
                       L = 0.8)

#### Example 2:
# Calculate the cost rasters to travel to ad from the center of a polygon
costRasters <- getCosts(grid[8,], from = grid[8,], dir = dir, destination = 'all',
                       polygons = 'center',
                       costs = c('dt','dW_l'), costFUN = energyCosts,
                       m = 70, v_max = 1.5, BMR = 76, k = 3.5, s = 0.05, l_s = 1,
                       L = 0.8)
```

getData *Access specific datasets*

Description

Access various datasets used in vignettes and studies associated with the broader `lbmech` project

Usage

```
getData(data, name = NULL, dir = tempdir(), timeout = 999)
```

Arguments

`data`	A character string indicating the name of the dataset to access. See details below.
`name`	For values of `data` that download data (see details), what will be the file/directory name of the downloaded items?
`dir`	Directory to which downloaded data will be saved. Default is `tempdior()`.
`timeout`	How many seconds before downloads time out? Default is 999. Temporarily overrides value in `getOptions("timeout")`.

Details

`data = 'baleares-currents'` imports ocean current data around Mallorca, Menorca, and Cabrera on June 13, 2022. Data originally downloaded from E.U. Copernicus Marine Service Information. See Clementi et al. (2021). An internet connection is **not** needed, and parameters `name` and `dir` are ignored.

`data = 'baleares-gps'` downloads GPS tracks for human hikes in the Balearic Islands in GPX format from `https://osf.io/77n9t/`. See Lera et al. (2017). The `name` parameter will define the folder name in the `dir` directory to which the `.gpx` files are saved. Default `name = 'gpx'`. `getData` will **not** import these tracks, for that, use `importGPX`. An internet connection is needed.

`data = 'baleares-places'` imports a SpatVector with twelve locations on Mallorca, Menorca, and Cabrera in the Balearic Islands.

Value

Various, depending on `data` selection:

`data = 'baleares-currents'` returns a SpatRaster with ocean current surface velocities in m/s.

`data = 'baleares-gpx'` does not return any object, but creates a sub-directory `name` in directory `dir` with 15,373 GPX files, of which 15,371 can be successfully imported using importGPX.

`data = 'baleares-places'` imports a SpatVector with twelve points.

References

Clementi, E., Aydogdu A., Goglio, A. C., Pistoia J., Escudier R., Drudi M., Grandi A., et al. (2021). Mediterranean Sea Physics Analysis and Forecast (CMEMS MED-Currents, EAS6 System). *Copernicus Marine Service*. doi:10.25423/CMCC/MEDSEA_ANALYSISFORECAST_PHY_006_013_EAS7.

Lera I., Perez T., Guerrero C., Eguiluz V. M., Juiz C. (2017). Analysing human mobility patterns of hiking activities through complex network theory. *PLoS ONE* 12(5): e0177712. doi:10.1371/journal.pone.0177712

Examples

```
# Import ocean current data for the Balearic Islands

currents <- getData('baleares-currents')
```

`getDistortion` *Get geodesic distortion for rasters*

Description

Calculate the geodesic distortion on a raster

Usage

```
getDistortion(z)
```

Arguments

`z`	An input SpatRaster with a known projection

Value

A SpatRaster with three layers: (1) `'lx'` with the horizontal pixel size

(2) `'ly'` with the vertical pixel size

(3) `'A'` with the pixel area

Examples

```
# Generate dummy dem, assign it a projection
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"

# Get the geodesic distorsion
distort <- getDistortion(dem)
```

`getEnv` *Download and extract environmental data for known locations.*

Description

Get environmental variables from trusted, global datasets, and optionally extract it for point locations

Usage

```
getEnv(
  region = NULL,
  locs = NULL,
  x = "x",
  y = "y",
  dem = NULL,
  proj = NULL,
  z_fix = NULL,
  zoom = 10,
  z_min = 0,
  filt = 0,
  slope = FALSE,
  twi = FALSE,
  depth = FALSE,
  soils = NULL,
  climate = NULL,
  acc = 15,
  vars = NULL,
  overwrite = FALSE,
  dir = tempdir(),
  ...
)
```

Arguments

`region`	A SpatRaster, Raster*, SpatVector, Spatial* or character string representing the area of interest. Required unless `locs` are provided. If `region = NULL` but `locs` are provided, then `region = ext(locs)`.

`locs`	(Optional) A data.frame or something coercible to a data.table containing all observations to which data should be extracted.
`x`	A character string representing the data column containing the 'x' coordinates in meters or degrees. Ignored if data is of class SpatVector or Spatial.
`y`	A character string representing the data column containing the 'y' coordinates in meters or degrees. Ignored if data is of class SpatVector or Spatial, and ignored for distance calculations if `dl` is provided.
`dem`	(Optional) A SpatRaster or Raster object to use for slope and topographic wetness calculations. Default is `dem = NULL`, which will download the appropriate SRTM data using `get_elev_raster(z = zoom)`
`proj`	A crs object or character string representing the projection information for x,y coordinates. If z_fix is provided, default is for `proj = z_fix`. Ignored if locs if `locs` is SpatVector or Spatial. If `locs` is a data.table or data.frame and `proj = NULL`, the function will attempt `proj = crs(dem)` as a last resort.
`z_fix`	A raster with the origin, projection, and resolution of the desired output rasters.
`zoom`	Considered only if `dem = NULL` The zoom level to be downloaded. See documentation for the `z` parameter in `get_elev_raster` for further information.
`z_min`	The minimum allowable elevation. Useful if DEM source includes ocean bathymetry as does the SRTM data from AWS. Default is `z_min = 0`, but set to `NULL` to disable.
`filt`	Numeric. Size of moving window to apply a low-pass filter to terrain-based metrics (`slope` VIA DEM, `twi` directly)
`slope`	Logical. Should the slope at each input location be calculated? Default uses SRTM data from `get_elev_raster`, but this can be overwritten by providing a `dem`
`twi`	Logical. Should the topgraphic wetness index be calculated? Default uses the provided `dem` as a slope source, and HydroSHEDS 15s arcsecond data as a flow accumulation source, however this can be altered by modifying the `acc` parameter. All data are corrected for geodesic distorsion, which can be time-consuming if `acc = 3`
`depth`	Logical. Should the depth to bedrock be obtained? If TRUE, SoilGrids v. 1.0 data are downloaded from ISRIC SoilGrids. Unfortunately, these data are only provided globally and therefore the download is 1.34 GB.
`soils`	A character vector containing zero, all or, some of `c('sand','silt','clay','cec','soc')` representing soil's parts per million sand, silt, and clay content, the cation exchange capacity, and the organic carbon (respectively). Data are downloaded from ISRIC SoilGrids using their OGC Web Service API allowing the download to be fairly efficient.
`climate`	A character vector containing zero, all or, some of c("tmin","tavg","tmax","prec","srad") representing the minimum average temperature, average temperature, maximum average temperature, precipitation, and solar radiation. Data are downloaded from WorldClim v. 2 using `worldclim_country`.
`acc`	One of (1) A Raster or SpatRaster representing the flow accumulation in units of incoming per cell, or (2) an integer equal to one of `acc = c(3, 15, 30)`, representing the resolution in arcseconds to download from HydroSHEDS. Default is `acc = 15`. Note that the 3s data is 2.29 GB.
`vars`	A SpatRaster object (or something coercible to it using the `rast` function) of one or multiple layers containing custom environmental data. Data with names equivalent to a parameter that has been activated will be ignored.

`overwrite`	Should the
`dir`	A filepath to the directory being used as the workspace. Default is `tempdir()` but unless the analyses will only be performed a few times it is highly recommended to define a permanent workspace so that files only need to be downloaded once.
`...`	Additional parameters to pass to `fix_z`.

Value

A folder in the ‘dir‘ with the requested datasets.

Examples

```
# Get coordinates for Mexico City in longlat
data <- data.table(x = -99.1332,
                   y = 19.4326)

## Not run due to size of downloads
# getEnv(data, filt = 3)
```

`getMap` *Download or crop necessary DEMs*

Description

A function that checks if the DEMs for a given set of sectors exist in the workspace, and if not downloads them or crops them from a larger file

Usage

```
getMap(
  tiles,
  polys,
  tile_id = "TILEID",
  vals = "location",
  z_min = NULL,
  filt = 0,
  verbose = FALSE,
  dir = tempdir()
)
```

Arguments

`tiles`	A character vector–such as the output to `whichTiles`—containing the unique tile IDs for sectors that should be in the workspace.
`polys`	A polygon of class SpatVector representing the partitioning grid for the maximum possible area, in the same format as the output of the `makeGrid` function.
`tile_id`	a character string representing the name of the column in the polys polygon containing the unique Tile IDs. Default is `tile_id = 'TILEID'`

vals	A character string or a SpatRast or Raster* object. Optional if the polys polygon is the output of the makeGrid function as the default is the character string 'location'. If no DEM was provided when makeGrid was initially run (i.e. polys$location == NA), then the function will use get_elev_raster to download data. If not from makeGrid, the vals parameter should be set to the column name containing the URL or filepath to the DEM for that sector.
z_min	The minimum allowable elevation. Useful if DEM source includes ocean bathymetry as does the SRTM data from AWS. Default is z_min = NULL, but set to 0 for SRTM data.
filt	Numeric. Size of moving window to apply a low-pass filter. Default is filt = 0.
verbose	Should the number of remaining sectors be printed? Default is FALSE
dir	A filepath to the directory being used as the workspace. Default is tempdir() but unless the analyses will only be performed a few times it is highly recommended to define a permanent workspace.

Value

Function does not return any objects, but sets up the workspace such that the necessary DEM files are downloaded/cropped and accessible.

Examples

```
# Generate a DEM, export it
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"

dir <- tempdir()
writeRaster(dem, paste0(dir,"/DEM.tif"),overwrite=TRUE)


# Import raster, get the grid
dem <- rast(paste0(dir,"/DEM.tif"))
grid <- makeGrid(dem = dem, nx = n, ny = n, sources = TRUE)


# Generate five random points that fall within the grid
points <- data.table(x = runif(5, ext(dem)[1], ext(dem)[2]),
                     y = runif(5, ext(dem)[3], ext(dem)[4]))


# Run whichTiles and getMap to prepare appropriate sector files
tile_list <- whichTiles(region = points, polys = grid)
getMap(tiles = tile_list, polys = grid, dir = dir)
```

getPaths *Get least-cost paths*

Description

Get the shortest path for a given trip that requires travel through a set of nodes. Use is like `getCosts`, but with `nodes` and `order` parameters and no `from` or `to`.

Usage

```
getPaths(
  region,
  nodes,
  id = "ID",
  order = NULL,
  x = "x",
  y = "y",
  costs = "all",
  polygons = "centroid",
  dir = tempdir(),
  ...
)
```

Arguments

`region`	A SpatVector, Spatial* or Raster* representing the area of maximum movement
`nodes`	One of data.frame, data.table, SpatVector, SpatialPointsDataFrame, or SpatialPolygonsDataFrame representing the node locations. If it is a polygon, the location will be controlled by the `polygons` parameter.
`id`	A character string representing the column containing each `node` location's unique ID.
`order`	A character vector containing the desired path in order of visited nodes by ID. Required if For example, to visit "A" then "B" then "C" then "A" the vector would be `c("A","B","C","A")`. If this is not provided, the function assumes that `nodes` is already sorted in the desired order.
`x`	A character vector representing the column containing the 'x' coordinates. Required if `data` is not Spatial*.
`y`	A character vector representing the column containing the 'y' coordinates. Required if `data` is not Spatial*.
`costs`	A character vector containing any combination of the strings of differential values present in the environment (see `calculateCosts` function; default is `costs = c("dt","dW_l","dE_l")` anticipating the use of `calculateCosts(costFUN = energyCosts)`.
`polygons`	One of `c('polygon','centroid','center')`. Ignored unless `nodes` are polygons. If `polygons = 'centroid'` (the default), the destinations are calculated to the centroid of the polygon whether or not it lies within the polygon itself. If `polygons = 'center'`, distances are calculated to the point within the polygon closest to the centroid. If `polygons = 'polygons'`, distances are calculated to *any* point within the polygon—in essence, the polygon acts as a giant node

	permitting costless movement within its bounds. This is generally not consistent with real-world scenarios and for that reason is not the default.
`dir`	A filepath to the directory being used as the workspace. Default is `tempdir()` but unless the analyses will only be performed a few times it is highly recommended to define a permanent workspace.
`...`	Additional parameters to pass to `importWorld`.

Value

SpatialVector lines representing least-cost paths. For each cost, each entry within the SpatialLinesDataFrame object represents a single leg of the journey, sorted in the original path order. If `length(costs)` == 1, only a SpatVector is returned. If `length(costs)` > 1 a list of SpatVectors with one slot for each `cost` is returned.

Examples

```
# Generate a DEM
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"

# Export it so it doesn't just exist on the memory
dir <- tempdir()
writeRaster(dem, paste0(dir,"/DEM.tif"),overwrite=TRUE)


# Import raster, get the grid
dem <- rast(paste0(dir,"/DEM.tif"))
grid <- makeGrid(dem = dem, nx = n, ny = n, sources = TRUE)
region <- grid[8, ]

# Select all tiles that exist between x = (12000,16000) and y = (32000,36000)
tiles <- ext(c(12000,16000,32000,36000))
tiles <- as.polygons(tiles)
crs(tiles) <- crs(grid)
tiles <- whichTiles(region = tiles, polys = grid)

# Make a world but limit it to the DEM grid size
calculateCosts(tiles = tiles, dir = dir,
m = 70, v_max = 1.5, BMR = 76, k = 3.5, s = 0.05, l_s = 1,
L = 0.8)

# Generate five random points that fall within the region
points <- data.table(ID = 1:5,
                     x = runif(5, ext(region)[1], ext(region)[2]),
                     y = runif(5, ext(region)[3], ext(region)[4]))


# Calculate the path from 1 -> 2 -> 5 -> 1 -> 4
pathOrder <- c(1,2,5,1,4)
```

```
# Make a world but limit it to the DEM grid size
defineWorld(source = grid, cut_slope = 0.5,
            res = res(dem), dir = dir, overwrite=TRUE)

paths <- getPaths(region = region, nodes = points, order = pathOrder,
                  costs = 'all', costFUN = energyCosts,
                       m = 70, v_max = 1.5, BMR = 76, k = 3.5, s = 0.05, l_s = 1,
                       L = 0.8)

## Plot against corridors (not run)
#getCosts(region = region, from = points, proj = crs(dem), res = res(dem),
#                         destination = 'all', costs = 'all',
#                         output = 'file', dir = dir)
#corridors <- makeCorridor(rasters = dir, order = pathOrder)
#plot(corridors$time)
#plot(paths$time,add=TRUE)
```

getSuitability *Calculate the suitability of a set of observations*

Description

This function measures how suitable an observation is given set of metrics that tend to increase the favorability of a location. Z-scores are first calculated for each metric (with an optional threshold). These are then standardized between [0,1], and a weighted mean between all metrics is performed to find the global suitability. Values of zero indicate that all of the least-favorable values are coincident at that observation; values of one indicate that the location sees all of the most favorable conditions for each metric.

Usage

```
getSuitability(
  x,
  group.var = NULL,
  weights = NULL,
  suit.name = "Suit",
  stdev = 2,
  keep.vars = FALSE
)
```

Arguments

`x`	a numeric vector, data.frame, or data.table with metrics representing values that tend to increase (or decrease, if negative `weights` are provided) the favorability of an observation
`group.var`	(Optional) A character string representing the name of the column with group IDs, such that z-scores and suitabilities are calculated only within groups.
`weights`	(Optional) A named list of numerics or vector of class numeric by which to weigh favorability observations in the final suitability estimate. If `NULL`, all weights are given equal (positive) importance. If a negative weight is provided, the assumption is that higher values *decrease* favorability. If vector, it must

	be be of the same length as the number of variable columns in `x`. If named list, the names should correspond with the favorability metrics to consider, and the values to the strength of the weights. Missing values are assumed to have a zero weight.
`suit.name`	The name of the output net suitability column. Default is `suit.name = 'Suit'`. If `NULL`, only thresheld z-scores are calculated
`stdev`	The threshold value for z-scores (see `rescaleSD`). Default is `stdev = 2`
`keep.vars`	Should the thresheld z-scores for each variable be kept? Default is `keep.vars = FALSE`

Examples

```
# Creata dummy data
x <- data.table(A = c(1:10),
                B = c(11:20),
                C = c(21:30),
                D = c(31:40),
                ID = rep(c("a","b"),5))

# Create weights, don't consider 'C'
weights <- list(A = 1,
                B = 2,
                D = -1)

# Get suitabilities
getSuitability(x, group.var = 'ID', weights = weights, keep.vars = TRUE)
```

`getVelocity` *Calculate a velocity function from data*

Description

Calculate the velocity function for an animal from *(x,y,z,t)* data such as from GPS collars, assuming a function of form Tobler (see ####).

Usage

```
getVelocity(
  data,
  x = "x",
  y = "y",
  degs = FALSE,
  dl = NULL,
  z = "z",
  dt = "dt",
  ID = "ID",
  tau = NULL,
  tau_vmax = 0.95,
  tau_nlrq = 0.5,
  k_init = 3.5,
  s_init = -0.05,
  v_min = 0,
  v_lim = Inf,
  slope_lim = 1,
  tile_id = "TILEID",
  vals = "location",
  dir = tempdir(),
  ...
)

dtVelocity(data, ...)
```

Arguments

`data`	A data.frame or something coercible to a data.table containing all observations
`x`	A character string representing the data column containing the 'x' coordinates in meters or degrees. Ignored if data is of class SpatVector or Spatial, and ignored for distance calculations if `dl` is provided but required to extract elevations if `z` is of class Raster* or SpatialPolygonsDataFrame.
`y`	A character string representing the data column containing the 'y' coordinates in meters or degrees. Ignored if data is of class SpatVector or Spatial, and ignored for distance calculations if `dl` is provided but required to extract elevations if `z` is of class SpatRaster, Raster*, or SpatVector, SpatialPolygonsDataFrame.
`degs`	Logical. If FALSE, the `getVelocity` proceeds as if the input coordinates are meters in a projected coordinate system. If TRUE, assumes the input coordinates are decimal degrees in lat/long, with `x` giving the longitude column name and `y` the latitude. See `distGeo`.
`dl`	A character string representing the data column containing the net displacement between this observation and the previous in meters. Optional.
`z`	Either a character string, a SpatRaster or Raster*, or a SpatVector object. If character string, it represents the data column containing the 'z' coordinates/elevations in meters. If a SpatRaster or Raster*, a DEM containing the elevation in meters. If SpatVector, it must represent the sectors and filepaths/URLs corresponding to the region's elevations (see the output of `makeGrid`).
`dt`	A character string representing the data column containing the time difference from the previous observation in seconds.
`ID`	A character string representing the data column containing the unique ID for each observed trajectory. In other words, each set of points for each continuous observation for each observed individual would merit a unique id.
`tau`	A number between 0 and 1, representing a global cutoff value for `tau_vmax` and `tau_nlrq`. Ignored if the latter two are provided.
`tau_vmax`	A number between 0 and 1, representing the percentile at which the maximum velocity is calculated. In other words, if `tau_vmax = 0.95` (the default), the maximum velocity will be at the 99.5th percentile of all observations.
`tau_nlrq`	A number between 0 and 1, representing the percentile at which the nonlinear regression is calculated. In other words, if `tau_nlrq = 0.50` (the default), the total curve will attempt to have at each interval 5% of the observations above the regression and 95% below.
`k_init`	A number representing the value for the topographic sensitivity at which to initiate the nonlinear regression. Default is `k_init = 3.5`.

s_init	A number representing the value for dimensionless slope of maximum velocity at which to initiate the nonlinear regression. Default is s_init = -0.05.
v_min	The maximum velocity that will be considered. Any value equal to or below this will be excluded from the regression. Default is v_lim = 0.
v_lim	The maximum velocity that will be considered. Any value above this will be excluded from the regression. Default is v_lim = Inf, but it should be set to an animal's maximum possible velocity.
slope_lim	the maximum angle that will be considered. Any value above this will be excluded from the regression. Default is slope_lim = 1.
tile_id	a character string representing the name of the column in the z polygon containing the unique Tile IDs. Ignored if elevations are provided as a column or Raster*. Otherwise default is tile_id = 'TILEID'.
vals	A character string or a Raster* object. Required only if the z parameter is a polygon NOT the output of the makeGrid function as the default is the character string 'location'. If not, the vals parameter should be set to the column name containing the URL or file path to the DEM for that sector.
dir	A filepath to the directory being used as the workspace. Default is tempdir() but unless the analyses will only be performed a few times it is highly recommended to define a permanent workspace.
...	Additional parameters to pass to importMap. Ignored unless z is a polygon pointing to source dem files.

Details

dtVelocity is a wrapper for getVelocity for use inside of a data.table with observations from multiple sources (be it different individual animals and/or different instances from the same animal). In a data.table, use this function in the j slot, passing it along .SD.

Value

A list, containing the following entries:

(1) $model, containing an object of class nlrq containing the output model from the nonlinear quantitative regression.

(2) $vmax, containing the identified maximum velocity.

(3) $s, containing the identified angle of maximum velocity.

(4) $k, containing the identified topographic sensitivity factor.

(5) $tau_max, containing the employed tau_max.

(6) $tau_nlrq, containing the employed tau_nlrq.

(7) $data, containing a data.table with the original data in a standardized format

Examples

```
# Note that the output results should be senseless since they
# are computed on random data

#### Example 1:
# If the data contains an 'elevation' or 'z' column
data <- data.table(x = runif(10000,10000,20000),
                   y = runif(10000,30000,40000),
                   elevation = runif(10000,0,200),
                   dt = 120,
                   ID = rep(1:10,each=1000))
velocity <- getVelocity(data = data, z = 'elevation')


#### Example 2:
# If the data do not contain elevation, and 'z' is a raster
suppressWarnings( data[, z := NULL])

# Generate a DEM
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"

velocity <- getVelocity(data = data, z = dem)


### Example 3:
# If the data do not contain elevation, and 'z' is a sector grid pointing
# to file locations

# Export the DEM so it's not just stored on the memory
dir <- tempdir()
writeRaster(dem, paste0(dir,"/DEM.tif"),overwrite=TRUE)

# Import raster, get the grid
dem <- rast(paste0(dir,"/DEM.tif"))
grid <- makeGrid(dem = dem, nx = n, ny = n, sources = TRUE)

velocity <- getVelocity(data = data, z = grid, dir = dir, res = res(dem))

#' # Example 4:
# If you want to get the values by group within a data.table

# Generate fake data with x,y coordinates, z elevation, and a t
# column representing the number of seconds into the observation
data <- data.table(x = runif(10000,10000,20000),
                   y = runif(10000,30000,40000),
                   z = runif(10000,0,200),
                   dt = 15,
                   ID = rep(1:10,each=1000))

# To get the velocity function for all observations together
v1 <- getVelocity(data)

# This is the same as above, but it only returns a list with the
# coefficients and p-values
v2 <- dtVelocity(data)

# Instead this function is best to get the coefficients for
# each individual track un a data.table using .SD
v3 <- data[, dtVelocity(.SD), by = 'ID', .SDcols = names(data)]
```

`importGPX` *Import GPX tracks*

Description

Import `GPX` tracks from commercial GPS equipment as a data.table ready for velocity function estimation. Note that the output coordinates must still be converted to a conformal projection in meters if they are to be used in functions other than `getVelocity`, `dtVelocity`, or `downsampleXYZ`.

Usage

```
importGPX(tracks, verbose = FALSE)
```

Arguments

`tracks`	A character string or vector with filepaths pointing to the location of the GPX files
`verbose`	Should a progress bar be printed? Default is `verbose = FALSE`, recommended particularly when used inside `lapply`

Value

A data.table with five or six columns:

(1) `$TrackID` The name of the input GPX track

(2) `$PID` The order that point appeared in the GPX file

(3) `$t` The timestamp

(4) `$long` The longitude in decimal degrees

(5) `$lat` The latitude in decimal degrees

(6) `$z` The elevation (if present)

Examples

```
# Get a list of GPX tracks in a directory
gpx <-  list.files(pattern = ".gpx$")

# Convert to data.table
gpx <- importGPX(gpx)
```

`importMap` *Import a contiguous raster map*

Description

Import a raster for a specific region from a multisource environment, such as the outputs of the getMap function.

Usage

```
importMap(
  region,
  polys,
  tile_id = "TILEID",
  z_fix = NULL,
  neighbor_distance = 5,
  FUN = NULL,
  mask = FALSE,
  vals = "location",
  filt = 0,
  z_min = NULL,
  dir = tempdir(),
  ...
)
```

Arguments

`region`	A SpatRaster, Raster*, SpatVector, Spatial* or character string representing the area of interest
`polys`	A polygon of class SpatVector representing the partitioning grid for the maximum possible area, in the same format as the output of the `makeGrid` function.
`tile_id`	A character string representing the name of the column in the `polys` polygon pontaining the unique Tile IDs. Default is `tile_id = 'TILEID'`
`z_fix`	A SpatRaster or Raster* object with the desired output projection, resolution, and origin. Required if `tiles` is of classes SpatVector, Spatial*, or character, unless `res` is provided.
`neighbor_distance`	An integer representing the number of cells that tiles are buffered. In other words, to ensure that there are no gaps between tiles, neighboring tiles within `neighborhood_distance` cells are also considered as potential sources. Default is 5 cells.
`FUN`	Function to deal with overlapping values for overlapping tiles. Default is `NA`, which uses `merge`. To use `mosaic`, provide a compatible function
`mask`	If `FALSE` (the default), the output map will contain all cells falling within the extent of `region`. If `TRUE`, places with `NA` (if `region` is SpatRaster or Raster*) or no coverage (if `region` is SpatVector or Spatial*) will be assigned a value of NA.

`vals`	A character string or a Raster* object. Required only if the `z` parameter is a polygon NOT the output of the `makeGrid` function as the default is the character string `'location'`. If not, the `vals` parameter should be set to the column name containing the URL or file path to the DEM for that sector.
`filt`	Numeric. Size of moving window to apply a low-pass filter. Default is `filt = 0`. Ignored unless the tiles need to be generated from the raw source files.
`z_min`	The minimum allowable elevation. Useful if DEM source includes ocean bathymetry as does the SRTM data from AWS. Default is `z_min = NULL`, but set to `0` for SRTM data. Ignored unless the tiles need to be generated from the raw source files.
`dir`	A filepath to the directory being used as the workspace. Default is `tempdir()`, but unless the analyses will only be performed a few times it is highly recommended to define a permanent workspace.
`...`	Additional agruments to pass to `fix_z`

Examples

```
# Generate a DEM, export it
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"

dir <- tempdir()
writeRaster(dem, paste0(dir,"/DEM.tif"),overwrite=TRUE)


# Import raster, get the grid
dem <- rast(paste0(dir,"/DEM.tif"))
grid <- makeGrid(dem = dem, nx = n, ny = n, sources = TRUE)

# Import the map for the center tile resampled to a different resolution
dem2 <- importMap('SECTOR_13', grid, res = 20)
```

`importWorld` *Import a world where movement is possible*

Description

A function that for a given region imports all cells from the transition `.fst` files. If such files have not yet been generated, they can be created by passing along the necessary parameters to this function as with `calculateCosts`.

Usage

```
importWorld(
  region,
  banned = NULL,
  dir = tempdir(),
  vars = NULL,
  costFUN = energyCosts,
  ...
)
```

Arguments

`region`	An object of class SpatRraster, Raster* or SpatialPolygons* representing the total area where movement is possible.
`banned`	An object of class Raster* or SpatialPolygons* representing the total area where movement is *prohibited*. Must lie within the area defined by `polys`
`dir`	A filepath to the directory being used as the workspace, the same one instantiated with `defineWorld`. Default is `tempdir()` but unless the analyses will only be performed a few times it is highly recommended to define a permanent workspace.
`vars`	The variable names to import.
`costFUN`	A cost function such as (`timeCosts` or `energyCosts`). The input to such a function should be a `data.table` object with column names present in the makeWorld file (initially `c('x_i','x_f','y_i','y_f','z_i','z_f','dz','dl','dr')`), and the output should be an data.table object with the same number of rows and the desired output variables as the only column names. Constants can be passed in the `...` slot. Default is `costFUN` = `energyCosts`.
`...`	Additional arguments to pass to `calculateCosts` and `costFUN`.

Details

The default parameters are sufficient for a workflow involving calculating costs with the `energyCosts` function. However, if non-energetic analyses are desired, the user must define their own.

Value

An object of class data.table containing at least three columns:

(1) `$from`, a character string of all possible origin cells in format "x,y",

(2) `$to`, a character string of all possible destination cells in format "x,y"

(3+) a numeric representing the imported costs(s)

Examples

```
# Generate a DEM
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
```

```
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"

# Export it so it doesn't just exist on the memory
dir <- tempdir()
writeRaster(dem, paste0(dir,"/DEM.tif"),overwrite=TRUE)


# Import raster, get the grid
dem <- rast(paste0(dir,"/DEM.tif"))
grid <- makeGrid(dem = dem, nx = n, ny = n, sources = TRUE)

# Select all tiles that exist between x = (12000,16000) and y = (32000,36000)
tiles <- ext(c(12000,16000,32000,36000))
tiles <- as.polygons(tiles)
crs(tiles) <- crs(grid)
tiles <- whichTiles(region = tiles, polys = grid)

#' # Make a world but limit it to the DEM grid size
defineWorld(source = grid, cut_slope = 0.5,
            res = res(dem), dir = dir, overwrite=TRUE)

# Make a world but limit it to the DEM grid size
world <- importWorld(grid[8,], dir = dir, vars = 'dE_l', costFUN = energyCosts,
m = 70, v_max = 1.5, BMR = 76, k = 3.5, s = 0.05, l_s = 1,
L = 0.8)
```

incrementalLID *Incremental Local Indicators of Dispersion*

Description

Determine the bandwidth that maximizes the non-group component of inequality.

Usage

```
incrementalLID(
  x,
  dist,
  bws,
  n = rep(1, length(x)),
  ntrials = 50,
  alpha = 0.05,
  standard = NULL,
  expect = NULL,
  mode = "adaptive",
  weighting = "membership",
  FUN = NULL,
  inf.val = NULL,
  row.stand = "fuzzy",
  minval = 50,
  var.stand = FALSE,
  var.exp = FALSE,
  ng.invert = TRUE,
  max.cross = .Machine$integer.max,
  pb = TRUE,
  ...
)
```

Arguments

`x`	A vector of weights with the same length as `x`
`dist`	A matrix or distance object representing pairwise distances. The distances need not be symmetrical.
`bws`	A vector containing the representing the bandwidth within neighbors are considered. If `mode = 'adaptive'`, `bw` is the number of nearest neighbors. If `mode = 'fixed'`, `bw` is the radius of the window in the map units.
`n`	A vector representing population weights. How much of an impact does a given observation have on any other observation regardless of its influence as provided for in `w`. Default is `1` for all.
`ntrials`	The number of permutations to perform. Default is 50.
`alpha`	Threshold for significance. Default is `alpha = 0.05`.
`standard`	The standards matrix with dimensions `length(x) x length(x)` used when calculating `lid`. Ignored if none had been originally provided, otherwise required.
`expect`	The expectations matrix with dimensions `length(x) x length(x)` used when calculating `lid`. Ignored if none had been originally provided, otherwise required.
`mode`	One of `'adaptive'`, which considers a `bw` number of nearest neighbors; or `'fixed'`, which considers a fixed bandwidth window of radius `bw`.
`weighting`	One of `'membership'`, which considers binary membership such that neighbors are weighted 1 and non-neighbors 0; `'distance'` which weighs neighbors according to `FUN` with the raw distance matrix providing the distance; or `'rank'` which uses the rank-distance (i.e. 1 for nearest neighbor, 2 for second nearest...) as the distance variable.
`FUN`	The distance function. Default is `NULL` for `membership`, and `function(x) 1/x` otherwise.
`inf.val`	When singularities arise (e.g. whenever the value is 1/0 or `Inf`, what is the value by which they are replaced? Default `NULL` uses the value of the smallest neighbor pair from the entire dataset.
`row.stand`	Logical or `'fuzzy'`. If `TRUE` (the default), rows are standardized such that they sum to one. If `'fuzzy'`, rows are standardized as a proportion of the largest value.
`minval`	When distances are raw, what is the minimum allowable distance? Default is 50.
`var.stand`	Logical. Should the standards be permuted if a matrix was provided? Default is `FALSE`.
`var.exp`	Logical. Should the expectations be permuted if a matrix was provided? Default is `FALSE`.
`ng.invert`	Does a higher non-group value imply higher between group inequality? Default is `TRUE`. This is ignored if matrixes were not originally provided, as it is automatically performed.

`max.cross`	When processing, what is the maximum number of rows that an internal data.table can have? This is generally not a concern unless the number of observations approaches `sqrt(.Machine$integer.max)`–usually about 2^31 for most systems. Lower values result in a greater number of chunks thus allowing larger data.sets to be calculated.
`pb`	Logical. Should a progress bar be displayed? Default is `FALSE`, although if a large dataset is processed that requires adjusting `max.cross` this can be useful
`...`	Additional parameters to pass on to `LID`.

Value

A list with three entries:

(1) `index` A named character with the code of the index named by its name

(2) `$bws` The bandwidths that significantly optimize the non-group inequality. Generally, a neighborhood is the first significant peak.

(3) `$stats` A data.table with the global group, non-group, and total values for each bandwidth, as well as a column indicating whether or not it's significant.

Examples

```
# Generate dummy observations
x <- runif(10, 1, 100)

# Get distance matrix
dists <- dist(x)

# Bandwidth sizes from 3 to 5
bws <- 3:5

inc <- incrementalLID(x, dist = dists, bws = bws, index = 'gini', type = 'local',
                      weighting = 'distance', FUN = function(x) 1/x^2, minval = 1)
```

`incrementalPlot` *Incremental Local Indicators of Dispersion plot*

Description

Plot the difference in the non-group component of inequality at increasing bandwidth sizes

Usage

```
incrementalPlot(inc)
```

Arguments

`inc`	The list output of the `incrementalLID` function

Value

A ggplot object

Examples

```
# Generate dummy observations
x <- runif(10, 1, 100)

# Get distance matrix
dists <- dist(x)

# Bandwidth sizes from 3 to 5
bws <- 3:5

inc <- incrementalLID(x, dist = dists, bws = bws, index = 'gini', type = 'local',
                      weighting = 'distance', FUN = function(x) 1/x^2, minval = 1)

# Plot the results
incrementalPlot(inc)
```

`inferLID` *Infer if dispersion is significant*

Description

Infer whether there exists more or less within- and between-group local and global inequality than would be expected versus if for all observations the values of all other observations were permuted. This tests if local values are significantly above or below what is expected given the global dataset, and if global values are significantly above or below what is expected given an otherwise random distribution.

Usage

```
inferLID(
  lid,
  w,
  ntrials = 999,
  alpha = 0.05,
  standard = NULL,
  expect = NULL,
  var.stand = FALSE,
  var.exp = FALSE,
  ng.invert = TRUE,
  max.cross = .Machine$integer.max,
  pb = TRUE,
  clear.mem = FALSE
)
```

Arguments

`lid`	The list output from the `LID` function.
`w`	The same spatial weights matrix used in calculating the `lid` input.
`ntrials`	The number of permutations to perform. Default is 999.
`alpha`	Threshold for significance. Default is `alpha = 0.05`.

`standard`	The standards matrix with dimensions `length(x)` x `length(x)` used when calculating `lid`. Ignored if none had been originally provided, otherwise required.
`expect`	The expectations matrix with dimensions `length(x)` x `length(x)` used when calculating `lid`. Ignored if none had been originally provided, otherwise required.
`var.stand`	Logical. Should the standards be permuted if a matrix was provided? Default is `FALSE`.
`var.exp`	Logical. Should the expectations be permuted if a matrix was provided? Default is `FALSE`.
`ng.invert`	Does a higher non-group value imply higher between group inequality? Default is `TRUE`. This is ignored if matrixes were not originally provided, as it is automatically performed.
`max.cross`	When processing, what is the maximum number of rows that an internal data.table can have? This is generally not a concern unless the number of observations approaches `sqrt(.Machine$integer.max)`–usually about 2^31 for most systems. Lower values result in a greater number of chunks thus allowing larger data.sets to be calculated.
`pb`	Logical. Should a progress bar be displayed? Default is `FALSE`, although if a large dataset is processed that requires adjusting `max.cross` this can be useful
`clear.mem`	Logical. Should `gc` be run in the middle of the calculation? Default is `clear.mem` but set as `TRUE` if memory limits are a concern.

Details

The output list can be passed to `scatterLID` to plot the group and non-group components of local inequality based on the significance classes.

Value

A list with the following entries:

(1) `$local` A data.table with one column, indicating whether an observation is falls in one of nine categories: Global High, Average, or Low for between-group inequality, and Local High, Average, or Low for within-group inequality based on the significance according to the delta-G statistic in the `$stats` data.table.

(2) `$global` A list with three entries, `$G_G` for the group component of the global inequality, `$G_NG` for the nongroup, and `$G` for the total. Each entry itself contains two entries, `$delta`, representing the delta-G statistic, and `$p`, representing its p-value.

(3) `$stats` A data.table containing the number of permutations a randomly-calculated `$G_Gi`, `$G_NGi`, or `$G_i` was above or below the real value

Examples

```
# Generate dummy observations
x <- runif(10, 1, 100)

# Get distance matrix
dists <- dist(x)

# Get fuzzy weights considering 5 nearest neighbors based on
# inverse square distance
weights <- makeWeights(dists, bw = 5,
                                   mode = 'adaptive', weighting = 'distance',
                                   FUN = function(x) 1/x^2, minval = 0.1,
                                   row.stand = 'fuzzy')

# Obtain the 'local gini' value
lid <- LID(x, w = weights, index = 'gini', type = 'local')

# Infer whether values are significant relative to the spatial distribution
# of the neighbots
inference <- inferLID(lid, w = weights, ntrials = 100)
```

LID *Local Indicators of Dispersion*

Description

Calculate dispersion indexes according to a given set of standards and expectations (Mejia Ramon and Munson 2023), obtaining group, non-group, and total values for local observations and the global dataset.

Usage

```
LID(
  x,
  w,
  index = "gini",
  expect = "self",
  standard = "global",
  n = rep(1, length(x)),
  mle = "mean",
  fun.name = paste0(index, "q"),
  type = "spatial",
  max.cross = .Machine$integer.max,
  canonical = FALSE,
  pb = FALSE,
  clear.mem = TRUE
)
```

Arguments

`x`	A vector values
`w`	A weights matrix of dimensions `length(x)` x `length(x)` representing that a given observation `j` (along the columns) is a part of `i` (along the rows)'s group. This can be the output of the `makeWeights` function.
`index`	A character string, either 'gini' (the default), or 'inoua', representing whether distances are calculated in L1 or L2 space, respectively. Alternatively, a numeric representing to what value distances and means are raised to when the index is calculated. `index = 1` for Gini, and `index = 2` for Inoua.
`expect`	Either a character string or a matrix with dimensions `length(x)` x `length(x)`, representing the expectation value from which errors are calculated for each

	observation pair between `i` and `j`. If `expect = 'self'`, the expectation is calculated as `i`; if `expect = 'local'`, the expectation is the neighborhood weighted mean; if if `expect = 'global'`, the expectation is the global mean. Expectations that depend on other metrics (including hypothesis-driven that do not depend on the observed dataset) can be provided by using an appropriate matrix.
`standard`	Either a character string or a matrix with dimensions `length(x) x length(x)`, representing the standard by which errors are judged by each observation pair between `i` and `j`. If `standard = 'self'`, the standard is calculated as `i`; if `standard = 'other'`, the standard is calculated as `j`; if `standard = 'local'`, the standard is the neighborhood weighted mean; if if `standard = 'global'`, the standard is the global mean. Standards that depend on other metrics (including hypothesis-driven that do not depend on the observed dataset) can be provided by using an appropriate matrix.
`n`	A vector representing population weights. How much of an impact does a given observation have on any other observation regardless of its influence as provided for in `w`. Default is `1` for all.
`mle`	Character string identifying the maximum likelihood estimator to be used. Default is `mle = 'mean'` for the traditional Gini, although since it uses mean absolute error and implies Laplace distribution, `mle = 'median'` is recommended. Alternatively, `index = 'inoua'` and `mle = 'mean'` for Gaussian processes.
`fun.name`	If `index != c('gini','inoua',1,2)`, how should the function be named? Default is `fun.name = paste0(index,'q')`.
`type`	A character string, either the name or corresponding code of a particular standard-expectation pair, as defined in #Link to Mejia Ramon and Munson 2023#
`max.cross`	When processing, what is the maximum number of rows that an internal data.table can have? This is generally not a concern unless the number of observations approaches `sqrt(.Machine$integer.max)`–usually about 2^31 for most systems. Lower values result in a greater number of chunks thus allowing larger data.sets to be calculated.
`canonical`	Should the canonical Gini or Inoua value also be calculated? Default is `FALSE`, and is ignored if `index > 2`.
`pb`	Logical. Should a progress bar be displayed? Default is `FALSE`, although if a large dataset is processed that requires adjusting `max.cross` this can be useful
`clear.mem`	Logical. Should `gc` be run in the middle of the calculation? Default is `clear.mem` but set as `TRUE` if memory limits are a concern.

Details

The output list can be passed to `inferLID` to determine whether the values are locally and globally higher or lower than would be expected if other values were randomly distributed among remaining observations.

Value

A list with the following entries:

(1) `$index` A named character string with the code of the index, named with its name

(2) `$local` A data.table, with three columns: `G_Gi`, the local group dispersion index; `G_NGi`, the local non-group dispersion index; and `G_i`, the local total dispersion index. Rows are in the same order as the input vector. This data.table also contains the chosen expectations and standards as hidden attributes to be used by `inferLID`.

(3) `$global` A list with three entries: `$G_G`, the global group dispersion index; `$G_NG`, the global nongroup dispersion index; and `$G`, the global total dispersion index.

(4) `$canonical` The canonical Gini or Inoua index, if `canonical = TRUE` and `index < 3`.

Examples

```
# Generate dummy observations
x <- runif(10, 1, 100)

# Get distance matrix
dists <- dist(x)

# Get fuzzy weights considering 5 nearest neighbors based on
# inverse square distance
weights <- makeWeights(dists, bw = 5,
                       mode = 'adaptive', weighting = 'distance',
                       FUN = function(x) 1/x^2, minval = 0.1,
                       row.stand = 'fuzzy')

# Obtain the 'local gini' value
lid <- LID(x, w = weights, index = 'gini', type = 'local')
```

---

makeCorridor *Calculate cost corridors for a path*

Description

A function to automatically perform the raster arithmetic necessary to calculate the cost-of-travel for paths with multiple waypoints, and the predicted cost of taking a detour to any arbitrary point in the landscape (a 'corridor'). `getCosts` must have been run before this tool can be used.

Usage

```
makeCorridor(rasters = tempdir(), order, costs = "all", name = NULL)
```

Arguments

`rasters`	One of either a character string or multilayer SpatRaster. If character string, it represents the filepath to the workspace used as `dir` for the previous functions. Default is `tempdir()` but unless you are not following best practices you will have to change it to your output directory. If multilayer SpatRaster, it should be the output (or identical in form) to the `getCosts` function with `"object" %in% output`. Note that if files have been generated for two different `from` objects in the `getCosts` sharing an attribute with the same ID name the function may throw an error.
`order`	A character vector containing the desired path in order of visited nodes. For example, to visit "A" then "B" then "C" then "A" the vector would be `c("A","B","C","A")`. Note that these MUST correspond to the ID names for the `from` features used in the `getCosts` function and must have previously been calculated

`costs`	A character vector containing any combination of pre-calculated cost names (e.g. dt for time, dW_l for work using `energyCosts`) if the input world data.table is the output of of the `calculateCosts` function. This selects which types of costs will be calculated. `costs = 'all'` is shorthand for `costs = c("dt","dW_l","dE_l")` while `costs = 'energetics'` is shorthand for `c("dW_l","dE_l")`. Default is `'all'`. Note that these must have previously been calculated.
`name`	A character vector representing the `outname` in `getCosts`. If none is provided, it will be the name of the variable passed to the function in the `from` slot. If `length(costs) == 1`, a Raster* If `length(costs) > 1` a list of Raster* with one slot for each `cost`.

Value

Rasters representing cost corridors.

Examples

```
# Generate a DEM
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"

# Export it so it doesn't just exist on the memory
dir <- tempdir()
writeRaster(dem, paste0(dir,"/DEM.tif"),overwrite=TRUE)


# Import raster, get the grid
dem <- rast(paste0(dir,"/DEM.tif"))
grid <- makeGrid(dem = dem, nx = n, ny = n, sources = TRUE)

# Import the data lying between x = (12000,16000) and y = (32000,36000)
region <- ext(c(12000,16000,32000,36000))
region <- as.polygons(region)
crs(region) <- crs(grid)

# Generate five random points that fall within the region
points <- data.table(ID = 1:5,
                     x = runif(5, ext(region)[1], ext(region)[2]),
                     y = runif(5, ext(region)[3], ext(region)[4]))

# Make a world but limit it to the DEM grid size
defineWorld(source = grid, cut_slope = 0.5,
            res = res(dem), dir = dir, overwrite=TRUE)

# Calculate cost rasters
costRasters  <- getCosts(region, from = points, dir = dir,
                         destination = 'all',
                         polygons = 'center',
                         costs = 'all', costFUN = energyCosts,
                         output = c('object','file'),
                         m = 70, v_max = 1.5, BMR = 76, k = 3.5, s = 0.05, l_s = 1,
                         L = 0.8)

# Calculating the corridors from a list of RasterStacks,
# with path 1 -> 2 -> 4 -> 1 -> 5
corridors <- makeCorridor(rasters = costRasters, order = c(1,2,5,1,4),)

#### Example 2:
# Calculating the corridors from a workspace directory
# with path 1 -> 2 -> 4 -> 1 -> 5
corridors <- makeCorridor(rasters = dir, name = 'points',
                          order = c(1,2,5,1,4))
```

makeGrid *Make partitioning grid*

Description

Generate a partitioning grid for a single raster source representing regional elevations. Smaller partitioning grids (i.e. a greater value of `nx * ny`) results in a greater number of saved files and a greater number of read-write operations in future operations, but reduces the amount of memory employed.

Usage

```
makeGrid(
  dem,
  nx,
  ny,
  path = NA,
  sources = FALSE,
  extension = NULL,
  proj = NULL,
  prefix = "SECTOR_",
  crop = TRUE,
  zoom = 13,
  var = "z",
  overlap = 0.005
)
```

Arguments

`dem`	One of either a single character string, SpatVector (polygon), SpatialPolygons*, SpatRaster, or Raster If character, SpatRaster, Raster, such an object or (filepath to one) containing the elevations for the maximum possible extent imaginable for a study. Note that SpatRaster and Raster only work rasters that have been read in, not those that exist exclusively in the memory. If you have just generated the raster and it is in memory, export it first with `writeRaster` then use the filepath string as `dem` or re-import it with `rast` before using the SoatRaster object. If SpatVector or SpatialPolygons*, the extent of possible movement.

nx	The integer-number of columns in the output grid
ny	The integer-number of rows in the output grid
path	(Optional) The filepath or URL of the source DEM. Ignored if dem is of class raster or SpatRaster. If a SpatVector or SpatialPolygons is provided but no path, getMap will use get_elev_raster to download topographic data.
sources	Logical. Should source information be saved as attributes to the grid for use in getMap and defineWorld? Default is sources = FALSE.
extension	A character vector representing the extension of the source path. Required only if sources = TRUE and the extension is not apparent from the URL stored in the var column.
proj	A crs or something coercible to it representing the desired output projection. Default is the input raster's projection.
prefix	A character string containing the prefix to name individual sectors. Default is prefix = "SECTOR_"
crop	Logical. If TRUE (the default), the output polygons will be cropped by the original dem (if SpatVector or SpatialPolygons*), or by the area covered by non-NA cells (if raster or SpatRaster).
zoom	Considered only if var = 'z' and no data source is set. The zoom level to be downloaded. See documentation for the z parameter in get_elev_raster for further information. Default is 13, but see documentation for the z parameter in get_elev_raster.
var	If the polygons point to a data source, what will be the variable name in the internal GIS? Default is 'z' for elevation.
overlap	How much should adjacent polygons overlap to ensure there's contiguity between different tiles? Default is overlap = 0.005.

Value

Polygons of class SpatVector representing the individual sectors ('tiles'), with a dataframe containing three columns: the "TILEID", the raster's filepath, and a dummy column indicating that the grid was made using the makeGrid function. This will be necessary for future functions. The object MUST be stored on the disk, it should not be stored in the memory

Examples

```
# Generate a DEM, export it
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"

dir <- tempdir()
writeRaster(dem, paste0(dir,"/DEM.tif"),overwrite=TRUE)


# Import raster, get the grid
```

```
dem <- rast(paste0(dir,"/DEM.tif"))
grid <- makeGrid(dem = dem, nx = n, ny = n, sources = TRUE)
```

makeWeights *Calculate spatial weights matrices.*

Description

Given a distance matrix (symmetrical or not) and weighting function, calculate a spatial weights matrix with various ways of handling row-sums

Usage

```
makeWeights(
  x,
  bw,
  mode = "adaptive",
  weighting = "membership",
  FUN = NULL,
  offset = 0,
  inf.val = NA,
  minval = -Inf,
  row.stand = FALSE,
  clear.mem = FALSE
)
```

Arguments

`x`	A matrix or distance object representing pairwise distances. The distances need not be symmetrical.
`bw`	A number representing the bandwidth within neighbors are considered. If `mode = 'adaptive'`, `bw` is the number of nearest neighbors. If `mode = 'fixed'`, `bw` is the radius of the window in the map units.
`mode`	One of `'adaptive'`, which considers a `bw` number of nearest neighbors; or `'fixed'`, which considers a fixed bandwidth window of radius `bw`.
`weighting`	One of `'membership'`, which considers binary membership such that neighbors are weighted 1 and non-neighbors 0; `'distance'` which weighs neighbors according to `FUN` with the raw distance matrix providing the distance; or `'rank'` which uses the rank-distance (i.e. 1 for nearest neighbor, 2 for second nearest...) as the distance variable.
`FUN`	The distance function. Default is `NULL` for `'membership'`, and `function(x) 1/(offset + x)` otherwise.
`offset`	What value is added to the denominator to prevent singularities from arising (e.g. whenever the value is 1/0)? Larger values imply smaller distance-decay. Default is `offset = 0`.
`inf.val`	When singularities arise, (i.e. whenever the value is 1/0), by what value are they replaced? Default is the `FUN` of the lowest non-`minval` value.
`minval`	When distances are raw, what is the minimum allowable distance? Default is `-Inf`.

row.stand	Logical or 'fuzzy'. If TRUE (the default), rows are standardized such that they sum to one. If 'fuzzy', rows are standardized as a proportion of the largest value.
clear.mem	Logical. Should gc be run in the middle of the calculation? Default is clear.mem but set as TRUE if memory limits are a concern.

Value

A matrix of dimensions length(x) x length(x).

Examples

```
# Generate dummy observations
x <- runif(10, 0, 100)

# Get distance matrix
dists <- dist(x)

# Get fuzzy weights considering 5 nearest neighbors based on
# inverse square distance
weights <- makeWeights(dists, bw = 5,
                       mode = 'adaptive', weighting = 'distance',
                       FUN = function(x) 1/x^2, minval = 0.1,
                       row.stand = 'fuzzy')
```

---

makeWorld *Build the world from a defined environment*

Description

Function that defines the grid that can be traversed–the "world"–as well as the cells that can be accessed from each individual cell. This is the most time-intensive function.

Usage

```
makeWorld(
  tiles = NULL,
  dir = tempdir(),
  cols = c("x_i", "y_i", "dz", "dl", "dr"),
  output = "file"
)
```

Arguments

tiles	A character vector–such as the output to whichTiles—containing the unique tile IDs for sectors that should be in the workspace. Default is NULL.
dir	A filepath to the directory being used as the workspace, the same one instantiated with defineWorld. Default is tempdir() but unless the analyses will only be performed a few times it is highly recommended to define a permanent workspace.

cols	A character vector containing the name of the important spatial variables to be retained. Default is cols = c("x_i","y_i","z_i","z_f","dz","dl","dr"), but c("x_f","y_f") are also available.
output	A character string or vector, consisting of one or both of c('file','object'), representing whether a file should be written and/or whether an object should be returned. Default is output = file.

Details

It first checks to see if the required elevation models have been downloaded for each source file for the requested tiles grid, and then if transition .gz then if they have been converted to a sector's local files have already been created in the dir workspace. If not, it generates each at each step

Value

An .fst file for each sector named after its sector id stored in the /World/Diff directory, and/or a data.table object (depending on the output parameter) containing a data.table with five columns

(1) $from, a character string of all possible origin cells in format "x,y", rounded to the next-lowest integer

(2) $to, a character string of all possible destination cells in format "x,y" rounded to the next-lowest integer

(3) $dz, a numeric representing the change in elevation for each origin-destination pair

(4) $dl, a numeric representing the change in planimetric distance (x,y)

(5) $dr, a numeric representing the change in displacement (x,y,z)

Likewise, in the /World/Loc directory the local z elevation values projected to the locally defined grid as a writeRST file.

Examples

```
# Generate a DEM
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"

# Export it so it doesn't just exist on the memory
dir <- tempdir()
writeRaster(dem, paste0(dir,"/DEM.tif"),overwrite=TRUE)


# Import raster, get the grid
dem <- rast(paste0(dir,"/DEM.tif"))
grid <- makeGrid(dem = dem, nx = n, ny = n, sources = TRUE)

# Select all tiles that exist between x = (12000,16000) and y = (32000,36000)
tiles <- ext(c(12000,16000,32000,36000))
tiles <- as.polygons(tiles)
crs(tiles) <- crs(grid)
```

```
tiles <- whichTiles(region = tiles, polys = grid)

# Make a world but limit it to the DEM grid size
defineWorld(source = grid, cut_slope = 0.5,
            res = res(dem), dir = dir, overwrite=TRUE)

makeWorld(tiles = tiles, dir = dir)
```

plotVelocity *Plot GPS Velocities*

Description

Plot the log-probability of the observed velocity data points versus the regressed nonlinear quantile regression

Usage

```
plotVelocity(
  velocity,
  v_lim = 3,
  v_min = 0,
  slope_lim = 1,
  bins = 100,
  x.bins = bins,
  y.bins = bins
)
```

Arguments

`velocity`	The list output to `getVelocity`.
`v_lim`	The maximum velocities to plot (y-axis limit). Default is 3 m/s.
`v_min`	The minimum velocities to plot (y-axis limit). Default is 0.
`slope_lim`	The maximum slopes to plot (x-axis limits). Default is 1.
`bins`	Into how many bins are the axes divided?
`x.bins`	Into how many bins are the axes divided?
`y.bins`	Into how many bins are the axes divided?

Value

A ggplot object

Examples

```
# Note that the output results should be senseless since they
# are computed on random data

# If the data contains an 'elevation' or 'z' column
data <- data.table(x = runif(10000,10000,20000),
                   y = runif(10000,30000,40000),
                   elevation = runif(10000,0,200),
                   dt = 120,
                   ID = rep(1:10,each=1000))
velocity <- getVelocity(data = data, z = 'elevation')
plotVelocity(velocity)
```

---

rangeSample *Get a conditional sample from a raster*

Description

Get a sample of XY locations from a single raster meeting a certain condition

Usage

```
rangeSample(z, n, minval = NULL, maxval = NULL, replace = FALSE)
```

Arguments

z	A SpatRaster or RasterLayer object
n	Numeric. How many samples to draw
minval	Numeric. Select all cells larger than this value
maxval	Numeric. Select all cells lesser than thsi value
replace	Logical. Should samples be taken with replacement?

Examples

```
# Generate a DEM
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"

# Take a sample of 10 values between 150 and 200
points <- rangeSample(dem, 10, minval = 150, maxval = 200)
```

rastToTable *Append x,y data to a raster*

Description

Append x,y data to a raster (wrapper for init).

Usage

```
rastToTable(z)
```

Arguments

z	An object of class Raster* or SpatRaster

Value

A data.table containing the raster values in column names after the raster layers, and 'x' and 'y' columns containing the cell locations

Examples

```
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"
dem <- rastToTable(dem)
```

regionMask *Convert SpatRaster, Raster*, SpatVector, or SpatialPolygon* to "x,y"*

Description

Function that converts raster and SpatialPolygon* objects to a list of cells that fall within such a region.

Usage

```
regionMask(
  region,
  proj = crs(region),
  id = NULL,
  z_fix = NULL,
  precision = 2,
  ...
)
```

Arguments

`region`	An object of class SpatRaster, Raster*, SpatVector, or SpatialPolygon* to convert to "x,y"
`proj`	A crs object or character string representing the output projection. Default is `proj = crs(region)` unless `proj` or `z_fix` are provided in which case the latter takes precedence.
`id`	A character string indicating which column in a Spatial* or Spat* contains each feature's unique ID. Otherwise ignored
`z_fix`	A SpatRaster with the same origin and resolution as the `z_fix` used to generate the 'world' with `makeWorld`. Do not modify this parameter if you didn't modify it when you ran `makeWorld`.
`precision`	An integer representing the number of decimals to retain in the x and y directions. For grid sizes with nice, round numbers precisions can be low. This factor is controled by `rast` and must be the same as the one used to generate the 'world' with `makeWorld`. Default is 2.
`...`	Additional arguments to pass to `fix_z`.

Value

A character vector containing all cells that fall in the same location as the input 'region'. If `id` is provided, a data.table.

Examples

```
# Generate a DEM
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"

# Generate a polygon that falls within the DEM's extent
region <- ext(c(12500,12600,32500,32700))
region <- as.polygons(region)
crs(region) <- crs(dem)

maskedCells <- regionMask(region = region, res = res(dem))
```

`rescaleSD` *Standard deviational stretch*

Description

Perform a standard deviational stretch on any numeric coercible to a vector object. Z-scores are first calculated, and values that exceed the threshold are reassigned the theshold value

Usage

```
rescaleSD(x, stdev = 2)
```

Arguments

`x`	The input value, vector, list, or anything coercible to a vector with `unlist`. Values must be numeric
`stdev`	The threshold number of standard deviations. Input values `n` whith a z score whose absolute value is beyond this threshold will be reassigned `x[n] <- stdev * z/abs(z)`

Value

A numeric vector with threshold-limited z-scores

Examples

```
# Create a vector
x <- rnorm(10, mean = 10, sd = 10)

y <- rescaleSD(x, stdev = 1)
```

`scatterLID` *Local Indicators of Dispersion Scatterplot*

Description

Plot the local group and non-group components of a local indicator of dispersion, colored by their inference-based class.

Usage

```
scatterLID(lid, inference, log.scale = FALSE, x.lim = NULL, y.lim = NULL)

colorLID(x = NULL, table = FALSE)
```

Arguments

`lid`	The list output of the `LID` function.
`inference`	The list output of the `inferLID` function.
`log.scale`	Logical. Should the axes be log-transformed? Default is `FALSE`. If `TRUE`, log transformation is `log(1+x,10)`.
`x.lim`	One of `NULL` to determine the x-range automatically (the default), a numeric vector of length two providing the x boundaries, or a function that accepts the automatic boundaries and returns new limits (see `scale_x_continuous`).
`y.lim`	One of `NULL` to determine the y-range automatically (the default), a numeric vector of length two providing the y boundaries, or a function that accepts the automatic boundaries and returns new limits (see `scale_y_continuous`).
`x`	A character string or vector containing a LID significance class. Ignored if `table = TRUE`.

`table`	Logical. Should the function convert character strings of classes to hex codes of colors (`table = FALSE`, the default), or should it return the conversion table itself?

Details

`colorLID()` acts as a function converting class names to the hex codes corresponding to the colors used by scatterLID when `table = FALSE` (the default), and returns the color table itself when `table = FALSE`.

Value

A ggplot object with two elements—the LID Scatter plot and its scale.

Examples

```
# Generate dummy observations
x <- runif(10, 1, 100)

# Get distance matrix
dists <- dist(x)

# Get fuzzy weights considering 5 nearest neighbors based on
# inverse square distance
weights <- makeWeights(dists, bw = 5,
                       mode = 'adaptive', weighting = 'distance',
                       FUN = function(x) 1/x^2, minval = 0.1,
                       row.stand = 'fuzzy')

# Obtain the 'local gini' value
lid <- LID(x, w = weights, index = 'gini', type = 'local')

# Infer whether values are significant relative to the spatial distribution
# of the neighbots
inference <- inferLID(lid, w = weights, ntrials = 100, pb = FALSE)

# Plot the inferences
scatterLID(lid, inference)
```

`whichTiles` *Identify necessary tiles/sectors*

Description

Get the names of tiles that would be needed to perform an analysis over a given region-of-interest within the maximum possible extent defined by a source grid.

Usage

```
whichTiles(region, polys, tile_id = "TILEID", x = "x", y = "y")
```

Arguments

`region`	An object of class SpatRaster, SpatVector Raster*, Spatial*, data.frame, or data.table indicating the region-of-interest. If input is of class SpatialPoints*, data.table, or data.frame the `region` represents sectors containing the individual points.
`polys`	A polygon of class SpatVector representing the partitioning grid for the maximum possible area, in the same format as the output of the `makeGrid` function.
`tile_id`	a character string representing the name of the column in the `polys` polygon containing the unique Tile IDs. Default is `tile_id = 'TILEID'`
`x`	a character string representing column name containing the "x" coordinates. Required for SpatialPoints*, data.frame, and data.table `region`, otherwise ignored.
`y`	a character string representing column name containing the "y" coordinates. Required for SpatialPoints*, data.frame, and data.table `region`, otherwise ignored.

Value

A character vector containing the TILEIDs overlapping with the `region`

Examples

```
#### Example 1:
# If the grid is the product if the makeGrid function
# Make the grid
n <- 6
dem <- rast(ncol = n * 600, nrow = n * 600, vals = 1)
ext(dem) <- c(1000, 2000, 3000, 4000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"
# Export it so it doesn't just exist on the memory
dir <- tempdir()
writeRaster(dem, paste0(dir,"/DEM.tif"),overwrite=TRUE)

# Import raster, get the grid
dem <- rast(paste0(dir,"/DEM.tif"))
grid <- makeGrid(dem = dem, nx = n, ny = n, sources = TRUE)

# Select five random points that fall within the grid
points <- data.table(x = runif(5, ext(dem)[1], ext(dem)[2]),
                     y = runif(5, ext(dem)[3], ext(dem)[4]))

tile_list <- whichTiles(region = points, polys = grid)

#### Example 2 (Do not execute):
## If it is a custom polygon "polys", where Tile IDs are stored in
## a "NAME" column, and coordinates in "Easting" and "Northing"
# tile_list <- whichTiles(region = points, grid = polys,
#                         tile_id = "NAME", x = "Easting", y = "Northing")
```

`writeRST` *Fast read and write rasters*

Description

Read and write rasters using the fst library

Usage

```
writeRST(x, filename = NULL, object = FALSE, ...)

importRST(x, layers = NULL, ...)
```

Arguments

`x`	An object of class SpatRaster or Raster. It may not contain layers named 'x' or 'y'
`filename`	Character. Output filename. Do not use extensions. If null, (the defaul), no file is saved
`object`	Logical. If true, function returns a data.table passable to `importRST` that will produce a valid raster (i.e. fixes rounding errors when using `init`. Default is `FALSE`.
`...`	Additional parameters to pass on to `read_fst` or `write_fst`
`layers`	Character vector containing the names of the layers to import. Default is `layers = NULL` which imports all layers.

Value

If `object = TRUE`, a data.table with an added `$crs` attribute containing the projection information. Special values are stored in the first three rows: (1) The x and y resolution (2) The x and y coordinates of the corner, and (3) The x and y coordinates of the rounding offset

Examples

```
n <- 5
dem <- expand.grid(list(x = 1:(n * 100),
                        y = 1:(n * 100))) / 100
dem <- as.data.table(dem)
dem[, z := 250 * exp(-(x - n/2)^2) +
      250 * exp(-(y - n/2)^2)]
dem <- rast(dem)
ext(dem) <- c(10000, 20000, 30000, 40000)
crs(dem) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +datum=WGS84"

dir <- tempdir()

writeRST(dem, paste0(dir,'/DEM.fst'))
importRST(paste0(dir,'/DEM.fst'))
```

Index