Package 'lbmech'

March 7, 2023

Type Package

Title A Package to Study the Mechanics of Landscape and Behavior

Version 0.4.0

Author Andres G. Mejia Ramon

Maintainer Andres G. Mejia Ramon <andres.mejiaramon@oist.jp>

Description A series of functions to calculate the cost required to move across the landscape in terms of time, mechanical work, and net metabolic expenditure (energy).

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Imports stringr,
igraph,
quantreg,
reshape2,
stats,
utils,
methods,
sf,
geosphere,
tmaptools,
gtools,
elevatr,
fst,
raster

Depends terra,
data.table

Suggests R.utils

R topics documented:

`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",
  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",
  rivers = FALSE,
  river_speed = "speed",
  overwrite = FALSE,
  dir = tempdir(),
  ...
)
```

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'`
`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.

rivers — A SpatialPolygon* or SpatVector polygon representing the area covered by rivers. Optional.

river_speed	A character representing the column name in rivers that contains the average speed of the river in m/s. Required if rivers is provided.
overwrite	If a directory with a World subdirectory already exists, should the latter be overwritten? Default is overwrite = 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.
...	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)
```

`downsample` *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

```
downsample(
  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 downsampled 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 <- downsample(data = data, t_step = 3, z = 'z')
```

`dtVelocity` *Estimate multiple velocity functions in data.tables*

Description

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`.

Usage

```
dtVelocity(data, ...)
```

Arguments

`data`	A data.frame or something coercible to a data.table containing all observations.
`...`	Additional arguments to pass along to `getVelocity`. Set the `ID` parameter parameter to the column name representing each individual track segment. Otherwise, employ the same singular column name as in `by=` (data.table syntax).

Value

A list with five entries: `$k`, the sample's topographic sensitivity factor and its associated p-value `$k_p`; `$s`, the sample's slope of fastest travel and its associated p-value `$s_p`; and the estimated maximum walking speed `$v_max`.

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),
                   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)]
```

`energyCosts.Rd` *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, rivers = FALSE)

energyCosts(
  DT,
  method = "kuo",
  m = NULL,
  BMR = NULL,
  g = 9.81,
  epsilon = 0.2,
  l_s = NULL,
  L = NULL,
  gamma = NULL,
  time = timeCosts,
  rivers = 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 row over water? Default is `row_speed = NULL`, but required if `rivers = TRUE`.
`rivers`	Logical. If `FALSE` (the default), movement costs are calculated as if over land. If `rivers = TRUE`, movement costs are calculated considering a moving river.
`method`	A character string for the method that energy costs per unit stride should be calculated. one of `method %in% c('kuo','heglund','oscillator')`.

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' or 'oscillator', ignored for 'heglund'
L	The average leg length. Required for method = 'kuo', ignored for 'heglund' and 'oscillator'.
gamma	The fractional maximal deviation from average velocity per stride. Required for method = 'oscillator', ignored for 'kuo' and 'heglund'.
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 can a person row over water? Default is row_work = NULL, but required if rivers = TRUE.
...	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.

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)
```

`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)
```

`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 = 0.995,
  tau_nlrq = 0.95,
  k_init = 3.5,
  s_init = -0.05,
  v_lim = Inf,
  slope_lim = 1,
  tile_id = "TILEID",
  vals = "location",
  dir = tempdir(),
  ...
)
```

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 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 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 = 0.995 (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 = 0.95 (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_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.

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))
```

`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 `downsample`.

Usage

```
importGPX(tracks)
```

Arguments

`tracks`	A character string or vector with filepaths pointing to the location of the GPX files

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",
  dir = tempdir(),
  filt = 0,
  ...
)
```

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.

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.
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.
...	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 = NULL,
  ...
)
```

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)
```

`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")
```

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.

Value

Rasters representing cost corridors. If `length(costs) == 1`, a Raster* If `length(costs) > 1` a list of Raster* with one slot for each `cost`.

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)

#### Example 1:
# 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, 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. 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)
```

---

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","dz","dl","dr"), but c("x_f","y_f","z_i","z_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)
```

`rastToTable` *Convert raster to data.table*

Description

Convert a raster to a data.table keeping the geometry properties. Note that the raster may not have any layers named 'x' or 'y' (lowercase). This does not preserve projection information.

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))
```

---

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, ...)

importRST(filename, 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.
`...`	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.

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"
writeRST(dem, 'DEM.fst')
importRST('DEM.fst')
```

Index