Package 'windfarmGA'

August 28, 2019

Title Genetic Algorithm for Wind Farm Layout Optimization

Version 2.2.2

Date 2019-08-25

Maintainer Sebastian Gatscha <sebastian_gatscha@gmx.at>

Description The genetic algorithm is designed to optimize wind farms of any shape. It requires a predefined amount of turbines, a unified rotor radius and an average wind speed value for each incoming wind direction. A terrain effect model can be included that downloads an 'SRTM' elevation model and loads a Corine Land Cover raster to approximate surface roughness.

Depends R (>= 3.2.3)

Imports rgdal, Rcpp, raster, sf, sp, rworldmap, gstat, ggplot2, RColorBrewer, calibrate, grDevices, graphics, leaflet, magrittr, foreach, parallel, doParallel, methods, spatstat, stats, utils

LinkingTo Rcpp

LazyData TRUE

License MIT + file LICENSE

URL https://ysosirius.github.io/windfarmGA/index.html

BugReports https://github.com/YsoSirius/windfarmGA/issues

RoxygenNote 6.1.1

Suggests testthat, pkgdown, dplyr, rgeos

Encoding UTF-8

X-schema.org-keywords windfarm-layout, optimization, genetic-algorithm, renewable-energy, r, rstats, r-package

NeedsCompilation yes

Author Sebastian Gatscha [aut, cre]

Repository CRAN

Date/Publication 2019-08-27 23:30:02 UTC

R topics documented:

`barometric_height` *Calculates Air Density, Air Pressure and Temperature according to the Barometric Height Formula*

Description

Calculates air density, temperature and air pressure respective to certain heights according to the International standard atmosphere and the barometric height formula.

Usage

```
barometric_height(data, height, po = 101325, ro = 1.225)
```

Arguments

`data`	A data.frame containing the height values
`height`	Column name of the height values
`po`	Standardized air pressure at sea level (101325 Pa)
`ro`	Standardized air density at sea level (1,225 kg per m3)

Value

Returns a data.frame with height values and corresponding air pressures, air densities and temperatures in Kelvin and Celsius.

See Also

Other Wind Energy Calculation Functions: `calculate_energy`, `get_dist_angles`, `turbine_influences`

Examples

```
data <- matrix(seq(0,5000,500));
barometric_height(data)
plot.ts(barometric_height(data))
```

big_shape *A big shapefile*

Description

A big shapefile

Usage

```
big_shape
```

Format

An object of class `SpatialPolygons` of length 1.

calculate_energy *Calculate Energy Outputs of Individuals*

Description

Calculate the energy output and efficiency rates of an individual in the current population under all given wind directions and speeds. If the terrain effect model is activated, the main calculations to model those effects will be done in this function.

Usage

```
calculate_energy(sel, referenceHeight, RotorHeight, SurfaceRoughness, wnkl,
  distanz, polygon1, resol, RotorR, dirSpeed, srtm_crop, topograp,
  cclRaster, weibull, plotit = FALSE)
```

Arguments

`sel`	A data.frame of an individual of the current population (data.frame)
`referenceHeight`	The height at which the incoming wind speeds were measured (numeric)
`RotorHeight`	The desired height of the turbines
`SurfaceRoughness`	A surface roughness length of the considered area in m. If the terrain effect model is activated, a surface roughness will be calculated for every grid cell with the elevation and land cover information
`wnkl`	Indicates the angle at which no wake influences are considered (numeric)
`distanz`	Indicates the distance after which the wake effects are considered to be eliminated
`polygon1`	The considered area as shapefile

resol	The resolution of the grid in meter
RotorR	The desired rotor radius in meter
dirSpeed	The wind speed and direction data.frame
srtm_crop	A list of 3 raster, with 1) the elevation, 2) an orographic and 3) a terrain raster. Calculated in genetic_algorithm
topograp	Logical value that indicates whether the terrain effect model is activated (TRUE) or deactivated (FALSE)
cclRaster	A Corine Land Cover raster that has to be downloaded previously. See also the details at windfarmGA The raster will only be used when the terrain effect model is activated. (raster)
weibull	A raster representing the estimated wind speeds
plotit	Logical value. If TRUE, the process will be plotted. Default is FALSE.

Value

Returns a list of an individual of the current generation with resulting wake effects, energy outputs, efficiency rates for every wind direction. The length of the list will be the amount of incoming wind directions.

See Also

Other Wind Energy Calculation Functions: barometric_height, get_dist_angles, turbine_influences

Examples

```
## Not run:
## Create a random shapefile
library(sp)
Polygon1 <- Polygon(rbind(c(4498482, 2668272), c(4498482, 2669343),
                    c(4499991, 2669343), c(4499991, 2668272)))
Polygon1 <- Polygons(list(Polygon1),1);
Polygon1 <- SpatialPolygons(list(Polygon1))
Projection <- '+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
+ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs'
proj4string(Polygon1) <- CRS(Projection)

## Create a uniform and unidirectional wind data.frame and plot the
## resulting wind rose
data.in <- data.frame(ws = 12, wd = 0)
windrosePlot <- plot_windrose(data = data.in, spd = data.in$ws,
                dir = data.in$wd, dirres=10, spdmax=20)

## Assign the rotor radius and a factor of the radius for grid spacing.
Rotor= 50; fcrR= 3
resGrid <- grid_area(shape = Polygon1, resol = Rotor*fcrR, prop=1,
                      plotGrid = TRUE)
## Assign the indexed data frame to new variable. Element 2 of the list
## is the grid, saved as SpatialPolygon.
resGrid1 <- resGrid[[1]]
```

```
## Create an initial population with the indexed Grid, 15 turbines and
## 100 individuals.
initpop <- init_population(Grid = resGrid1, n = 15, nStart = 100)

## Calculate the expected energy output of the first individual of the
## population.
par(mfrow = c(1,2))
plot(Polygon1); points(initpop[[1]][,'X'],initpop[[1]][,'Y'], pch=20,cex=2)
plot(resGrid[[2]], add = TRUE)
resCalcEn <- calculate_energy(sel=initpop[[1]],referenceHeight= 50,
                   RotorHeight= 50, SurfaceRoughness = 0.14,wnkl = 20,
                   distanz = 100000, resol = 200,dirSpeed = data.in,
                   RotorR = 50, polygon1 = Polygon1, topograp = FALSE,
                   weibull = FALSE)
resCalcEn <- as.data.frame(resCalcEn)
plot(Polygon1, main = resCalcEn[,'Energy_Output_Red'][[1]])
points(x = resCalcEn[,'Bx'], y = resCalcEn[,'By'], pch = 20)


## Create a variable and multidirectional wind data.frame and plot the
## resulting wind rose
data.in10 <- data.frame(ws = runif(10,1,25), wd = runif(10,0,360))
windrosePlot <- plot_windrose(data = data.in10, spd = data.in10$ws,
                dir = data.in10$wd, dirres=10, spdmax=20)

## Calculate the energy outputs for the first individual with more than one
## wind direction.
resCalcEn <- calculate_energy(sel=initpop[[1]],referenceHeight= 50,
                   RotorHeight= 50, SurfaceRoughness = 0.14,wnkl = 20,
                   distanz = 100000, resol = 200,dirSpeed = data.in10,
                   RotorR = 50, polygon1 = Polygon1, topograp = FALSE,
                   weibull = FALSE)

## End(Not run)
```

---

cansee *Calculate Visibility between 2 locations*

Description

Check if point 1 is visible from point 2 given a certain elevation model

Usage

```
cansee(r, xy1, xy2, h1 = 0, h2 = 0, reso)
```

Arguments

r	A DEM raster
xy1	A vector/matrix with X and Y coordinates for Point 1
xy2	A vector/matrix with X and Y coordinates for Point 2
h1	A numeric giving the extra height offset for Point 1
h2	A numeric giving the extra height offset for Point 2
reso	The minimal resolution of the DEM raster. It is calculated in viewshed and passed along.

Value

A boolean value, indicating if the point (xy2) is visible

See Also

Other Viewshed Analysis: interpol_view, plot_viewshed, rasterprofile, viewTo, viewshed

---

crossover *Crossover Method*

Description

The crossover method of the genetic algorithm, which takes the selected individuals after the selection function and produces new offsprings through permutation.

Usage

```
crossover(se6, u, uplimit, crossPart, verbose, seed)
```

Arguments

se6	The selected individuals. The output of selection (list)
u	The crossover point rate. (numeric)
uplimit	The upper limit of allowed permutations. The current algorithm has an upper bound of 300 permutations. (numeric)
crossPart	The crossover method. Either "EQU" or "RAN". (character)
verbose	If TRUE, will print out further information.
seed	Set a seed for comparability. Default is NULL

Value

Returns a binary coded matrix of all permutations and all grid cells, 0 indicates no turbine and 1 indicates a turbine in the grid cell. (matrix)

See Also

Other Genetic Algorithm Functions: fitness, genetic_algorithm, init_population, mutation, selection, trimton, windfarmGA

Examples

```
## Create two random parents with an index and random binary values
Parents <- data.frame(
  ID = 1:20,
  bin = sample(c(0,1),20, replace = TRUE, prob = c(70,30)),
  bin.1 = sample(c(0,1),20, replace=TRUE,prob = c(30,70)))

## Create random Fitness values for both individuals
FitParents <- data.frame(ID = 1, Fitness = 1000, Fitness.1 = 20)

## Assign both values to a list
CrossSampl <- list(Parents,FitParents);

## Cross their data at equal locations with 2 crossover parts
crossover(CrossSampl, u = 1.1, uplimit = 300, crossPart = "EQU")

## with 3 crossover parts and equal locations
crossover(CrossSampl, u = 2.5, uplimit = 300, crossPart = "EQU")

## or with random locations and 5 crossover parts
crossover(CrossSampl, u = 4.9, uplimit = 300, crossPart = "RAN")
```

---

dup_coords *Splits duplicated coords (copy of geoR::dup.coords)*

Description

This function takes an object with 2-D coordinates and returns the positions of the duplicated coordinates. Also sets a method for duplicated. Helper function for plot_heatmap

Usage

```
dup_coords(x, ...)
```

Arguments

x	Two column numeric matrix or data frame
...	passed to sapply. If simplify = TRUE (default) results are returned as an array if possible (when the number of replicates are the same at each replicated location)

Value

Function and methods returns NULL if there are no duplicates locations. Otherwise, the default method returns a list where each component is a vector with the positions or the rownames, if available, of the duplicates coordinates. The method for geodata returns a data-frame with rownames equals to the positions of the duplicated coordinates, the first column is a factor indicating duplicates and the remaining are output of as.data.frame.geodata.

Author(s)

Paulo Justiniano Ribeiro Jr. <paulojus@leg.ufpr.br> Peter J. Diggle <p.diggle@lancaster.ac.uk>

See Also

Other Helper Functions: getDEM, getISO3, get_grids, grid_area, hexa_area, isSpatial, permutations, readintegerSel, readinteger, splitAt, tess2SPdf, windata_format

---

fitness *Evaluate the Individual Fitness values*

Description

The fitness values of the individuals in the current population are calculated after having evaluated their energy outputs in calculate_energy. This function reduces the resulting energy outputs to a single fitness value for every individual.

Usage

```
fitness(selection, referenceHeight, RotorHeight, SurfaceRoughness, Polygon,
  resol1, rot, dirspeed, srtm_crop, topograp, cclRaster, weibull, Parallel,
  numCluster)
```

Arguments

selection	A list containing all individuals of the current population.
referenceHeight	The height at which the incoming wind speeds were measured.
RotorHeight	The desired height of the turbine.
SurfaceRoughness	A surface roughness length of the considered area in m.
Polygon	The considered area as shapefile.
resol1	The resolution of the grid in meter.
rot	The desired rotor radius in meter.
dirspeed	The wind data as list.
srtm_crop	A list of 3 raster, with 1) the elevation, 2) an orographic and 3) a terrain raster. Calculated in genetic_algorithm

topograp	Logical value that indicates whether the terrain effect model is activated (TRUE) or deactivated (FALSE).
cclRaster	A Corine Land Cover raster, that has to be adapted previously by hand with the surface roughness length for every land cover type. Is only used, when the terrain effect model is activated.
weibull	A raster representing the estimated wind speeds
Parallel	Boolean value, indicating whether parallel processing should be used. The parallel and doParallel packages are used for parallel processing.
numCluster	If Parallel is TRUE, this variable defines the number of clusters to be used.

Value

Returns a list with every individual, consisting of X & Y coordinates, rotor radii, the runs and the selected grid cell IDs, and the resulting energy outputs, efficiency rates and fitness values.

See Also

Other Genetic Algorithm Functions: crossover, genetic_algorithm, init_population, mutation, selection, trimton, windfarmGA

Examples

```
## Create a random rectangular shapefile
library(sp)
Polygon1 <- Polygon(rbind(c(4498482, 2668272), c(4498482, 2669343),
                  c(4499991, 2669343), c(4499991, 2668272)))
Polygon1 <- Polygons(list(Polygon1),1);
Polygon1 <- SpatialPolygons(list(Polygon1))
Projection <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
+ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)

## Create a uniform and unidirectional wind data.frame and plots the
## resulting wind rose
## Uniform wind speed and single wind direction
wind <- data.frame(ws = 12, wd = 0)
# windrosePlot <- plot_windrose(data = wind, spd = wind$ws,
#                dir = wind$wd, dirres=10, spdmax=20)

## Calculate a Grid and an indexed data.frame with coordinates and
## grid cell IDs.
Grid1 <- grid_area(shape = Polygon1,resol = 200,prop = 1);
Grid <- Grid1[[1]]
AmountGrids <- nrow(Grid)

wind <- list(wind, probab = 100)
startsel <- init_population(Grid,10,20);
fit <- fitness(selection = startsel, referenceHeight = 100, RotorHeight=100,
               SurfaceRoughness=0.3,Polygon = Polygon1, resol1 = 200,rot=20,
```

```
                dirspeed = wind, srtm_crop="", topograp=FALSE, cclRaster="",
                Parallel = FALSE)
```

---

genetic_algorithm *Run a Genetic Algorithm to optimize a wind farm layout*

Description

Run a Genetic Algorithm to optimize the layout of wind turbines on a given area. The algorithm works with a fixed amount of turbines, a fixed rotor radius and a mean wind speed value for every incoming wind direction.

Usage

```
genetic_algorithm(Polygon1, GridMethod, Rotor, n, fcrR, referenceHeight,
  RotorHeight, SurfaceRoughness, Proportionality, iteration, mutr, vdirspe,
  topograp, elitism, nelit, selstate, crossPart1, trimForce, Projection,
  sourceCCL, sourceCCLRoughness, weibull, weibullsrc, Parallel, numCluster,
  verbose = FALSE, plotit = FALSE)
```

Arguments

Polygon1	The considered area as SpatialPolygon, SimpleFeature Polygon or coordinates as matrix/data.frame
GridMethod	Should the polygon be divided into rectangular or hexagonal grid cells? The default is "Rectangular" grid cells and hexagonal grid cells are computed when assigning "h" or "hexagon" to this input variable.
Rotor	A numeric value that gives the rotor radius in meter
n	A numeric value indicating the required amount of turbines
fcrR	A numeric value that is used for grid spacing. Default is 5
referenceHeight	The height at which the incoming wind speeds were measured. Default is the RotorHeight.
RotorHeight	The desired height of the turbine.
SurfaceRoughness	A surface roughness length of the considered area in m. If the terrain effect model is activated, a surface roughness will be calculated for every grid cell with the elevation and land cover information. Default is 0.3
Proportionality	A numeric value used for grid calculation. Determines the percentage a grid has to overlay. Default is 1
iteration	A numeric value indicating the desired amount of iterations of the algorithm. Default is 20
mutr	A numeric mutation rate with a default value of 0.008

`vdirspe` A data.frame containing the incoming wind speeds, wind directions and probabilities

`topograp` Logical value, which indicates if the terrain effect model should be enabled or not. Default is FALSE

`elitism` Boolean value, which indicates whether elitism should be activated or not. Default is TRUE

`nelit` If `elitism` is TRUE, this input determines the amount of individuals in the elite group. Default is 7

`selstate` Determines which selection method is used, "FIX" selects a constant percentage and "VAR" selects a variable percentage, depending on the development of the fitness values. Default is "FIX"

`crossPart1` Determines which crossover method is used, "EQU" divides the genetic code at equal intervals and "RAN" divides the genetic code at random locations. Default is "EQU"

`trimForce` If activated (`trimForce == TRUE`), the algorithm will take a probabilistic approach to trim the windfarms to the desired amount of turbines. If deactivated (`trimForce == FALSE`) the adjustment will be random. Default is FALSE

`Projection` A desired Projection can be used instead of the default Lambert Azimuthal Equal Area Projection (EPSG:3035).

`sourceCCL` The path to the Corine Land Cover raster (.tif). Only required when the terrain effect model is activated. If nothing is assign, it will try to download a version from the EEA-website.

`sourceCCLRoughness`
The source to the adapted Corine Land Cover legend as .csv file. Only required when terrain effect model is activated. As default a .csv file within this package ('`~/extdata`') is taken that was already adapted manually. To use your own .csv legend this variable has to be assigned.

`weibull` A logical value that specifies whether to take Weibull parameters into account. If 'weibull == TRUE', the wind speed values from the 'vdirspe' data frame are ignored. The algorithm will calculate the mean wind speed for every wind turbine according to the Weibull parameters. Default is FALSE

`weibullsrc` A list of Weibull parameter rasters, where the first list item must be the shape parameter raster 'k' and the second item must be the scale parameter raster 'a' of the Weibull distribution. If no list is given, then rasters included in the package are used instead, which currently only cover Austria. This variable is only used if 'weibull == TRUE'.

`Parallel` Boolean value, indicating whether parallel processing should be used. The parallel and doParallel packages are used for parallel processing. Default is FALSE

`numCluster` If `Parallel` is TRUE, this variable defines the number of clusters to be used

`verbose` If TRUE it will print information for every generation. Default is FALSE

`plotit` If TRUE it will plot the best windfarm of every generation. Default is FALSE

Details

A terrain effect model can be included in the optimization process. Therefore, an SRTM elevation model will be downloaded automatically via the `raster::getData` function. A land cover raster can also be downloaded automatically from the EEA-website, or the path to a raster file can be passed to `sourceCCL`. The algorithm uses an adapted version of the Raster legend ("clc_legend.csv"), which is stored in the package directory '`~/inst/extdata`'. To use other values for the land cover roughness lengths, insert a column named **"Rauhigkeit_z"** to the .csv file, assign a surface roughness length to all land cover types. Be sure that all rows are filled with numeric values and save the file with **";"** separation. Assign the path of the file to the input variable `sourceCCLRoughness` of this function.

Value

The result is a matrix with aggregated values per generation, the best individual regarding energy and efficiency per generation, some fuzzy control variables per generation, a list of all fitness values per generation, the amount of individuals after each process, a matrix of all energy, efficiency and fitness values per generation, the selection and crossover parameters, a matrix with the generational difference in maximum and mean energy output, a matrix with the given inputs, a dataframe with the wind information, the mutation rate per generation and a matrix with all tested wind farm layouts.

See Also

Other Genetic Algorithm Functions: `crossover`, `fitness`, `init_population`, `mutation`, `selection`, `trimton`, `windfarmGA`

Examples

```
## Create a random rectangular shapefile
library(sp)
Polygon1 <- Polygon(rbind(c(4498482, 2668272), c(4498482, 2669343),
                          c(4499991, 2669343), c(4499991, 2668272)))
Polygon1 <- Polygons(list(Polygon1), 1);
Polygon1 <- SpatialPolygons(list(Polygon1))
Projection <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
+ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)

## Create a uniform and unidirectional wind data.frame and plot the
## resulting wind rose
data.in <- data.frame(ws = 12, wd = 0)
windrosePlot <- plot_windrose(data = data.in, spd = data.in$ws,
                dir = data.in$wd, dirres=10, spdmax=20)

## Runs an optimization run for 20 iterations with the
## given shapefile (Polygon1), the wind data.frame (data.in),
## 12 turbines (n) with rotor radii of 30m (Rotor) and rotor height of 100m.
result <- genetic_algorithm(Polygon1 = Polygon1,
                  n = 12,
                  vdirspe = data.in,
                  Rotor = 30,
```

```
                  RotorHeight = 100)
plot_windfarmGA(result = result, Polygon1 = Polygon1)

## Runs the same optimization, but with parallel processing and 3 cores.
result_par <- genetic_algorithm(Polygon1 = Polygon1, GridMethod ="h", n=12, Rotor=30,
                fcrR=5,iteration=10, vdirspe = data.in,crossPart1 = "EQU",
                selstate="FIX",mutr=0.8, Proportionality = 1,
                SurfaceRoughness = 0.3, topograp = FALSE,
                elitism=TRUE, nelit = 7, trimForce = TRUE,
                referenceHeight = 50,RotorHeight = 100,
                Parallel = TRUE, numCluster = 3)
plot_windfarmGA(result = result_par, GridMethod = "h", Polygon1 = Polygon1)

## Runs the same optimization, this time with hexagonal grids.
result_hex <- genetic_algorithm(Polygon1 = Polygon1, GridMethod ="h", n=12, Rotor=30,
                fcrR=5,iteration=10, vdirspe = data.in,crossPart1 = "EQU",
                selstate="FIX",mutr=0.8, Proportionality = 1,
                SurfaceRoughness = 0.3, topograp = FALSE,
                elitism=TRUE, nelit = 7, trimForce = TRUE,
                referenceHeight = 50,RotorHeight = 100)
plot_windfarmGA(result = result_hex, GridMethod = "h", Polygon1 = Polygon1)

## Run an optimization with the Weibull parameters included in the package.
result_weibull <- genetic_algorithm(Polygon1 = Polygon1, GridMethod ="h", n=12,
                fcrR=5,iteration=10, vdirspe = data.in,crossPart1 = "EQU",
                selstate="FIX",mutr=0.8, Proportionality = 1, Rotor=30,
                SurfaceRoughness = 0.3, topograp = FALSE,
                elitism=TRUE, nelit = 7, trimForce = TRUE,
                referenceHeight = 50,RotorHeight = 100,
                weibull = TRUE)
plot_windfarmGA(result = result_weibull, GridMethod= "h", Polygon1 = Polygon1)

## Run an optimization with given Weibull parameter rasters.
#araster <- "/..pathto../a_param_raster.tif"
#kraster <- "/..pathto../k_param_raster.tif"
#weibullrasters <- list(raster(kraster), raster(araster))
#result_weibull <- genetic_algorithm(Polygon1 = Polygon1, GridMethod ="h", n=12,
#                  fcrR=5,iteration=10, vdirspe = data.in,crossPart1 = "EQU",
#                  selstate="FIX",mutr=0.8, Proportionality = 1, Rotor=30,
#                  SurfaceRoughness = 0.3, topograp = FALSE,
#                  elitism=TRUE, nelit = 7, trimForce = TRUE,
#                  referenceHeight = 50,RotorHeight = 100,
#                  weibull = TRUE, weibullsrc = weibullrasters)
#plot_windfarmGA(result = result_weibull, GridMethod= "h", Polygon1 = Polygon1)
```

---

getDEM *Get DEM raster*

Description

Get a DEM raster for a country based on ISO3 code

Usage

```
getDEM(polygon, ISO3 = "AUT", clip = TRUE)
```

Arguments

polygon	A Spatial / SimpleFeature Polygon to crop the DEM
ISO3	The ISO3 code of the country
clip	boolean, indicating if polygon should be cropped. Default is TRUE

Value

A list with the DEM raster, and a SpatialPolygonsDataFrame or NULL if no polygon is given

See Also

Other Helper Functions: dup_coords, getISO3, get_grids, grid_area, hexa_area, isSpatial, permutations, readintegerSel, readinteger, splitAt, tess2SPdf, windata_format

Examples

```
## Not run:
library(sp)
library(raster)
Polygon1 <- Polygon(rbind(c(4488182, 2667172), c(4488182, 2669343),
                          c(4499991, 2669343), c(4499991, 2667172)))
Polygon1 <- Polygons(list(Polygon1), 1);
Polygon1 <- SpatialPolygons(list(Polygon1))
Projection <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
+ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)
DEM_meter <- getDEM(Polygon1)
plot(DEM_meter[[1]])
plot(DEM_meter[[2]], add=T)

## End(Not run)
```

---

getISO3 *getISO3*

Description

Get point values from the getMap

Usage

```
getISO3(pp, crs_pp = 4326, col = "ISO3", resol = "low",
  coords = c("LONG", "LAT"), ask = F)
```

Arguments

pp	SpatialPoints or matrix
crs_pp	The CRS of the points
col	Which column/s should be returned
resol	The search resolution if high accuracy is needed
coords	The column names of the point matrix
ask	A boolean, to ask which columns can be returned

Value

A character vector

See Also

Other Helper Functions: dup_coords, getDEM, get_grids, grid_area, hexa_area, isSpatial, permutations, readintegerSel, readinteger, splitAt, tess2SPdf, windata_format

Examples

```
## Not run:
points = cbind(c(4488182.26267016, 4488852.91748256),
c(2667398.93118627, 2667398.93118627))
getISO3(pp = points, ask = T)
getISO3(pp = points, crs_pp = 3035)

points <- as.data.frame(points)
colnames(points) <- c("x","y")
points <- st_as_sf(points, coords = c("x","y"))
st_crs(points) <- 3035
getISO3(pp = points, crs_pp = 3035)

## End(Not run)
```

---

get_dist_angles *Calculate distances and angles of possibly influencing turbines*

Description

Calculate distances and angles for a turbine and all it's potentially influencing turbines.

Usage

```
get_dist_angles(t, o, wkl, distanz, polYgon, plotAngles)
```

Arguments

t	A matrix of the current individual with x and y coordinates
o	A numeric value indicating the index of the current turbine
wkl	A numeric value indicating the angle, at which no wake influences are considered. Default is 20 degrees.
distanz	A numeric value indicating the distance, after which the wake effects are considered to be eliminated. Default is 100km.
polYgon	A shapefile representing the considered area
plotAngles	A logical variable, which is used to plot the distances and angles. Default is FALSE

Value

Returns a matrix with the distances and angles of potentially influencing turbines

See Also

Other Wind Energy Calculation Functions: barometric_height, calculate_energy, turbine_influences

Examples

```
library(sp)
library(raster)

## Exemplary input Polygon with 2km x 2km:
polYgon <- Polygon(rbind(c(0, 0), c(0, 2000), c(2000, 2000), c(2000, 0)))
polYgon <- Polygons(list(polYgon),1)
polYgon <- SpatialPolygons(list(polYgon))
Projection <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
               +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(polYgon) <- CRS(Projection); plot(polYgon, axes = TRUE)

## Create a random windfarm with 10 turbines
t <- as.matrix(cbind(x = runif(10, 0, raster::extent(polYgon)[2]),
     y = runif(10, 0, raster::extent(polYgon)[4])))
wnkl <- 20
distanz <- 100000

## Evaluate and plot for every turbine all other potentially influencing turbines
potInfTur <- list()
for (i in 1:(length(t[,1]))) {
  potInfTur[[i]] <- get_dist_angles(t = t, o = i, wkl = wnkl,
                    distanz = distanz, polYgon = polYgon, plotAngles = TRUE)
}
potInfTur
```

get_grids *Get the Grid-IDs from binary matrix*

Description

Get the grid IDs from the trimmed binary matrix, where the binary code indicates which grid cells are used in the current wind farm constellation.

Usage

```
get_grids(trimtonOut, Grid)
```

Arguments

trimtonOut	Input matrix with binary values.
Grid	Grid of the considered area

Value

Returns a list of all individuals with X and Y coordinates and the grid cell ID.

See Also

Other Helper Functions: dup_coords, getDEM, getISO3, grid_area, hexa_area, isSpatial, permutations, readintegerSel, readinteger, splitAt, tess2SPdf, windata_format

Examples

```
## Create a random rectangular shapefile
library(sp)
Polygon1 <- Polygon(rbind(c(0, 0), c(0, 2000), c(2000, 2000), c(2000, 0)))
Polygon1 <- Polygons(list(Polygon1),1);
Polygon1 <- SpatialPolygons(list(Polygon1))
Projection <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
+ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)

## Calculate a Grid and an indexed data.frame with coordinates and grid cell Ids.
Grid1 <- grid_area(shape = Polygon1,resol = 200,prop = 1);
Grid <- Grid1[[1]]
AmountGrids <- nrow(Grid)

startsel <- init_population(Grid,10,20);
wind <- data.frame(ws = 12, wd = 0)
wind <- list(wind, probab = 100)
fit <- fitness(selection = startsel,referenceHeight = 100, RotorHeight=100,
               SurfaceRoughness=0.3,Polygon = Polygon1, resol1 = 200,rot=20,
               dirspeed = wind, srtm_crop="",topograp=FALSE,cclRaster="")
```

```
allparks <- do.call("rbind",fit);

## SELECTION
## print the amount of Individuals selected.
## Check if the amount of Turbines is as requested.
selec6best <- selection(fit, Grid,2, TRUE, 6, "VAR");

## CROSSOVER
## u determines the amount of crossover points,
## crossPart determines the method used (Equal/Random),
## uplimit is the maximum allowed permutations
crossOut <- crossover(selec6best, 2, uplimit = 300, crossPart="RAN");

## MUTATION
## Variable Mutation Rate is activated if more than 2 individuals represent the
## current best solution.
mut <- mutation(a = crossOut, p = 0.3);

## TRIMTON
## After Crossover and Mutation, the amount of turbines in a windpark change
## and have to be corrected to the required amount of turbines.
mut1 <- trimton(mut = mut, nturb = 10, allparks = allparks,
               nGrids = AmountGrids, trimForce=FALSE)

## Get the new Grid-Ids and run a new fitness run.
getRectV <- get_grids(mut1, Grid)
fit <- fitness(selection = getRectV,referenceHeight = 100, RotorHeight=100,
               SurfaceRoughness=0.3,Polygon = Polygon1, resol1 = 200,rot=20,
               dirspeed = wind, srtm_crop="",topograp=FALSE,cclRaster="")
head(fit)
```

`grid_area` *Make a grid from a Polygon*

Description

Create a grid from a given polygon and with a certain resolution and proportionality. The center points of each grid cell represent possible locations for wind turbines.

Usage

```
grid_area(shape, resol = 500, prop = 1, plotGrid = FALSE)
```

Arguments

`shape`	Shape file of the considered area
`resol`	The resolution of the grid in meters. Default is 500

prop	A factor used for grid calculation. Determines the percentage a grid has to overlay the considered area to be represented as grid cell. Default is 1.
plotGrid	Logical value indicating whether resulting grid should be plotted or not. Default is FALSE.

Value

Returns a list with 2 elements. List element 1 will have the grid cell IDS, and the X and Y coordinates of the centers of each grid cell. List element 2 is the grid as SpatialPolygons, which is used for plotting purposes.

Note

The grid of the genetic algorithm will have a resolution of `Rotor * fcrR`. See the arguments of `windfarmGA`

Author(s)

Jose Hidasi (original) / Sebastian Gatscha (adapted)

References

http://rfunctions.blogspot.co.at/2014/12/gridfilter-intersect-grid-with-shape.html

See Also

Other Helper Functions: `dup_coords`, `getDEM`, `getISO3`, `get_grids`, `hexa_area`, `isSpatial`, `permutations`, `readintegerSel`, `readinteger`, `splitAt`, `tess2SPdf`, `windata_format`

Examples

```
library(sp)
library(raster)
library(rgeos)

## Exemplary input Polygon with 2km x 2km:
Polygon1 <- Polygon(rbind(c(0, 0), c(0, 2000),
c(2000, 2000), c(2000, 0)))
Polygon1 <- Polygons(list(Polygon1),1);
Polygon1 <- SpatialPolygons(list(Polygon1))
Projection <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
+ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)

## Create a Grid
grid_area(Polygon1,200,1,TRUE)
grid_area(Polygon1,400,1,TRUE)


## Examplary irregular input Polygon
Polygon1 <- Polygon(rbind(c(0, 20), c(0, 200),
```

```
                          c(2000, 2000), c(3000, 0)))
Polygon1 <- Polygons(list(Polygon1),1);
Polygon1 <- SpatialPolygons(list(Polygon1))
Projection <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
+ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)

## Create a Grid
grid_area(Polygon1,200,1,TRUE)
grid_area(Polygon1,200,0.5,TRUE)
grid_area(Polygon1,200,0.1,TRUE)
grid_area(Polygon1,400,1,TRUE)
grid_area(Polygon1,400,0.5,TRUE)
grid_area(Polygon1,400,0.1,TRUE)
```

`hexa_area` *Polygon to Hexagonal Grid Tessellation*

Description

The function takes a Polygon and a sizing argument and creates a list with an indexed matrix with coordinates and a SpatialPolygons object, that consists of hexagonal grids

Usage

```
hexa_area(Polygon1, size, plotTrue = FALSE)
```

Arguments

`Polygon1`	The SpatialPolygons object
`size`	The side length of an hexagon
`plotTrue`	Should the object be plotted

Value

Returns a list with an indexed matrix of the point coordinates and a SpatialPolygons object of the hexagons

See Also

Other Helper Functions: `dup_coords`, `getDEM`, `getISO3`, `get_grids`, `grid_area`, `isSpatial`, `permutations`, `readintegerSel`, `readinteger`, `splitAt`, `tess2SPdf`, `windata_format`

Examples

```
library(spatstat)
library(sp)
library(raster)
Polygon1 <- Polygon(rbind(c(4498482, 2668272), c(4498482, 2669343),
                          c(4499991, 2669343), c(4499991, 2668272)))
Polygon1 <- Polygons(list(Polygon1),1);
Polygon1 <- SpatialPolygons(list(Polygon1))
Projection <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
+ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)
HexGrid <- hexa_area(Polygon1, 100, TRUE)
plot(HexGrid[[2]])
```

---

hole_shape *A shapefile with a hole*

Description

A shapefile with a hole

Usage

```
hole_shape
```

Format

An object of class `SpatialPolygonsDataFrame` with 1 rows and 1 columns.

---

init_population *Create a random initial Population*

Description

Create `nStart` random sub-selections from the indexed grid and assign binary variable 1 to selected grids. This function initiates the genetic algorithm with a first random population and will only be needed in the first iteration.

Usage

```
init_population(Grid, n, nStart = 100)
```

Arguments

Grid	The data.frame output of grid_area" function, with X and Y coordinates and Grid cell IDs.
n	A numeric value indicating the amount of required turbines.
nStart	A numeric indicating the amount of randomly generated initial individuals. Default is 100.

Value

Returns a list of nStart initial individuals, each consisting of n turbines. Resulting list has the x and y coordinates, the grid cell ID and a binary variable of 1, indicating a turbine in the grid cell.

See Also

Other Genetic Algorithm Functions: crossover, fitness, genetic_algorithm, mutation, selection, trimton, windfarmGA

Examples

```
library(sp)
## Exemplary input Polygon with 2km x 2km:
Polygon1 <- Polygon(rbind(c(0, 0), c(0, 2000),
c(2000, 2000), c(2000, 0)))
Polygon1 <- Polygons(list(Polygon1),1);
Polygon1 <- SpatialPolygons(list(Polygon1))
Projection <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
+ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)

Grid <- grid_area(Polygon1,200,1,"TRUE")

## Create 5 individuals with 10 wind turbines each.
firstPop <- init_population(Grid = Grid[[1]], n = 10, nStart = 5)
```

---

interpol_view *Plot an interpolated viewshed*

Description

Plot an interpolated view of the viewshed analysis

Usage

```
interpol_view(res, plot = TRUE, breakseq, breakform = NULL,
  plotDEM = FALSE, fun = mean, pal = NULL, ...)
```

Arguments

res	The result list from viewshed.
plot	Should the result be plotted? Default is TRUE
breakseq	The breaks for value plotting. By default, 5 equal intervals are generated.
breakform	If 'breakseq' is missing, a sampling function to calculate the breaks, like quantile, fivenum, etc.
plotDEM	Plot the DEM? Default is FALSE
fun	Function used for rasterize. Default is mean
pal	A color palette
...	Arguments passed on to plot.

Value

An interpolated raster

See Also

Other Viewshed Analysis: cansee, plot_viewshed, rasterprofile, viewTo, viewshed

Other Plotting Functions: plot_cloud, plot_development, plot_evolution, plot_fitness_evolution, plot_heatmap, plot_parkfitness, plot_result, plot_viewshed, plot_windfarmGA, plot_windrose, random_search_single

Examples

```
## Not run:
library(sp)
library(raster)
Polygon1 <- Polygon(rbind(c(4488182, 2667172), c(4488182, 2669343),
                          c(4499991, 2669343), c(4499991, 2667172)))
Polygon1 <- Polygons(list(Polygon1), 1);
Polygon1 <- SpatialPolygons(list(Polygon1))
Projection <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
+ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)
DEM_meter <- getDEM(Polygon1)

turbloc = spsample(DEM_meter[[2]], 10, type = "random");
res <- viewshed(r = DEM_meter[[1]], shape=DEM_meter[[2]],
                turbine_locs = turbloc,  h1=1.8, h2=50)
interpol_view(res, plotDEM = T)

interpol_view(res, breakseq = seq(0,max(colSums(res$Result)),1))
interpol_view(res, plotDEM = F, breakform = quantile)
interpol_view(res, breakform = factor)

## Different color palettes
interpol_view(res, plotDEM = T, pal=topo.colors)
interpol_view(res, plotDEM = T, pal=colorRampPalette(c("white","purple")))
```

```
## ... Arguments are past on to the raster plot method
interpol_view(res, plotDEM = T, alpha=0.5)
interpol_view(res, plotDEM = F, breakseq = seq(0,10,1), colNA="black")


## End(Not run)
```

`isSpatial` *Transform to SpatialPolygons*

Description

Helper Function, which transforms SimpleFeatures or coordinates in matrix/data.frame/data.table into a SpatialPolygon

Usage

```
isSpatial(shape, proj)
```

Arguments

`shape`	An area as SpatialPolygon, SimpleFeature Polygon or coordinates as matrix/data.frame
`proj`	Which Projection should be assigned to matrix / data.frame coordinates

Details

If the columns are named, it will look for common abbreviation to match x/y or long/lat columns.
If the columns are not named, the first 2 numeric columns are taken.

Value

A SpatialPolygons object

See Also

Other Helper Functions: `dup_coords`, `getDEM`, `getISO3`, `get_grids`, `grid_area`, `hexa_area`, `permutations`, `readintegerSel`, `readinteger`, `splitAt`, `tess2SPdf`, `windata_format`

Examples

```
df <- rbind(c(4498482, 2668272), c(4498482, 2669343),
            c(4499991, 2669343), c(4499991, 2668272))
isSpatial(df)

Polygon1 <- Polygon(rbind(c(4498482, 2668272), c(4498482, 2669343),
                          c(4499991, 2669343), c(4499991, 2668272)))
Polygon1 <- Polygons(list(Polygon1), 1);
```

```
Polygon1 <- SpatialPolygons(list(Polygon1))
Projection <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
+ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)
df_fort <- ggplot2::fortify(Polygon1)
isSpatial(df_fort, Projection)
```

`multi_shape` *A multi-shapefile with 3 Polygons*

Description

A multi-shapefile with 3 Polygons

Usage

```
multi_shape
```

Format

An object of class `SpatialPolygons` of length 1.

`mutation` *Mutation Method*

Description

Mutate the genes of every chromosome or individual with low probability.

Usage

```
mutation(a, p, seed = NULL)
```

Arguments

`a`	The binary matrix of all individuals.
`p`	The mutation rate.
`seed`	Set a seed for comparability. Default is NULL

Value

Returns a binary matrix with mutated genes.

See Also

Other Genetic Algorithm Functions: crossover, fitness, genetic_algorithm, init_population, selection, trimton, windfarmGA

Examples

```
## Create 4 random individuals with binary values
a <- cbind(bin=sample(c(0,1),20,replace=TRUE,prob = c(70,30)),
        bin.1=sample(c(0,1),20,replace=TRUE,prob = c(30,70)),
        bin.2=sample(c(0,1),20,replace=TRUE,prob = c(30,70)),
        bin.3=sample(c(0,1),20,replace=TRUE,prob = c(30,70)))
a

## Mutate the individuals with a low percentage
aMut <- mutation(a,0.1, NULL)
## Check which values are not like the originals
a==aMut

## Mutate the individuals with a high percentage
aMut <- mutation(a,0.4, NULL)
## Check which values are not like the originals
a==aMut
```

---

permutations *Enumerate the Combinations or Permutations of the Elements of a Vector*

Description

permutations enumerates the possible permutations. The function is forked and minified from gtools::permutations

Usage

```
permutations(n, r, v = 1:n)
```

Arguments

n	Size of the source vector
r	Size of the target vectors
v	Source vector. Defaults to 1:n

Value

Returns a matrix where each row contains a vector of length r.

Author(s)

Original versions by Bill Venables <Bill.Venables@cmis.csiro.au.> Extended to handle repeats.allowed by Gregory R. Warnes <greg@warnes.net.>

References

Venables, Bill. "Programmers Note", R-News, Vol 1/1, Jan. 2001. https://cran.r-project.org/doc/Rnews/

See Also

Other Helper Functions: dup_coords, getDEM, getISO3, get_grids, grid_area, hexa_area, isSpatial, readintegerSel, readinteger, splitAt, tess2SPdf, windata_format

---

plot_cloud *Plot outputs of all generations with standard deviations*

Description

Plot the fitness, efficiency and energy outputs of all generations and the corresponding standard deviations.

Usage

```
plot_cloud(result, pl = FALSE)
```

Arguments

result	The output of windfarmGA or genetic_algorithm
pl	Should the results be plotted? Default is FALSE

Value

Returns a data.frame with the values for fitness, efficiency and energy for all evaluated individuals

See Also

Other Plotting Functions: interpol_view, plot_development, plot_evolution, plot_fitness_evolution, plot_heatmap, plot_parkfitness, plot_result, plot_viewshed, plot_windfarmGA, plot_windrose, random_search_single

Examples

```
## Add some data examples from the package
load(file = system.file("extdata/resulthex.rda", package = "windfarmGA"))

## Plot the results of a hexagonal grid optimization
plcdf <- plot_cloud(resulthex, TRUE)
```

---

plot_development *Plot the progress of populations*

Description

Plot the changes in mean and max fitness values to previous generation.

Usage

```
plot_development(result)
```

Arguments

result	The output windfarmGA or genetic_algorithm

See Also

Other Plotting Functions: interpol_view, plot_cloud, plot_evolution, plot_fitness_evolution, plot_heatmap, plot_parkfitness, plot_result, plot_viewshed, plot_windfarmGA, plot_windrose, random_search_single

Examples

```
plot_development(resultrect)
```

---

plot_evolution *Plot the evolution of fitness values*

Description

Plot the evolution of energy outputs and efficiency rates over the whole generations. Plots min, mean and max values.

Usage

```
plot_evolution(result, ask = TRUE, spar = 0.1)
```

Arguments

result	The output of windfarmGA or genetic_algorithm
ask	Should R wait for interaction for subsequent plotting. Default is TRUE
spar	A numeric value determining how exact a spline should be drawn. Default is 0.1

See Also

Other Plotting Functions: interpol_view, plot_cloud, plot_development, plot_fitness_evolution, plot_heatmap, plot_parkfitness, plot_result, plot_viewshed, plot_windfarmGA, plot_windrose, random_search_single

Examples

```
## Add some data examples from the package
load(file = system.file("extdata/resultrect.rda", package = "windfarmGA"))

## Plot the results of a rectangular grid optimization
plot_evolution(resultrect, ask = TRUE, spar = 0.1)
```

---

plot_fitness_evolution

Plot the changes of min/mean/max fitness values

Description

Plot the evolution of fitness values and the change in the min, mean and max fitness values to the former generations.

Usage

```
plot_fitness_evolution(result, spar = 0.1)
```

Arguments

result	The output of function windfarmGA or genetic_algorithm
spar	A numeric value determining how exact a spline should be drawn. Default is 0.1

See Also

Other Plotting Functions: interpol_view, plot_cloud, plot_development, plot_evolution, plot_heatmap, plot_parkfitness, plot_result, plot_viewshed, plot_windfarmGA, plot_windrose, random_search_single

Examples

```
## Add some data examples from the package
load(file = system.file("extdata/resulthex.rda", package = "windfarmGA"))

## Plot the results of a hexagonal grid optimization
plot_fitness_evolution(resulthex, 0.1)
```

`plot_heatmap` *Plot a heatmap of selected grid cells*

Description

Plot a heatmap of selected grid cells. Green grid cells have been selected more often than red grid cells.

Usage

```
plot_heatmap(result, si = 2, idistw)
```

Arguments

`result`	The output of `windfarmGA` or `genetic_algorithm`
`si`	A numeric value that is used for the sizing of the resolution of the heatmap. Default is 2
`idistw`	The inverse distance weighting power. Default is the rotor radius from the 'result' values

Value

Invisibly returns a list with the result of the inverse distance weighting and an aggregated dataframe of all grid cells

See Also

Other Plotting Functions: `interpol_view`, `plot_cloud`, `plot_development`, `plot_evolution`, `plot_fitness_evolution`, `plot_parkfitness`, `plot_result`, `plot_viewshed`, `plot_windfarmGA`, `plot_windrose`, `random_search_single`

Examples

```
## Add some data examples from the package
load(file = system.file("extdata/resulthex.rda", package = "windfarmGA"))

## Plot the results of a hexagonal grid optimization
plot_heatmap(resulthex)
```

```
## Plot the heatmap with different settings
plot_heatmap(resulthex, si = 6, idistw = 2)
plot_heatmap(resulthex, si = 6, idistw = 100)
plot_heatmap(resulthex, si = 20, idistw = 10)
```

---

plot_leaflet *Plot a Wind Farm with leaflet*

Description

Plot a resulting wind farm on a leaflet map.

Usage

```
plot_leaflet(result, Polygon1, which = 1, orderitems = TRUE, GridPol)
```

Arguments

result	The resulting matrix of the function genetic_algorithm or windfarmGA
Polygon1	The Polygon for the wind farm area.
which	A numeric value, indicating which best individual to plot. The default is 1 (the best resulting wind farm).
orderitems	A logical value indicating whether the results should be ordered by energy values (TRUE) or chronologically (FALSE).
GridPol	The output grid polygon of the grid_area or hexa_area functions.

Value

Returns a leaflet map.

Examples

```
load(file = system.file("extdata/resulthex.rda", package = "windfarmGA"))
load(file = system.file("extdata/polygon.rda", package = "windfarmGA"))

## Plot the best wind farm on a leaflet map (ordered by energy values)
plot_leaflet(result = resulthex, Polygon1 = polygon, which = 1)

## Plot the last wind farm (ordered by chronology).
plot_leaflet(result = resulthex, Polygon1 = polygon, orderitems = F,
         which = 1)

load(file = system.file("extdata/resultrect.rda", package = "windfarmGA"))
## Plot the best wind farm on a leaflet map with the rectangular Grid
Grid <- grid_area(polygon, resol = 150, prop = 0.4)
```

```
plot_leaflet(result = resultrect, Polygon1 = polygon, which = 1,
         GridPol = Grid[[2]])

## Plot the last wind farm with hexagonal Grid
Grid <- hexa_area(polygon, size = 75)
plot_leaflet(result = resulthex, Polygon1 = polygon, which = 1,
         GridPol = Grid[[2]])
```

`plot_parkfitness` *Plot the genetic algorithm results*

Description

Plot the evolution of fitness values with the influences of selection, crossover and mutation.

Usage

```
plot_parkfitness(result, spar = 0.1)
```

Arguments

`result`	The output of `windfarmGA` or `genetic_algorithm`
`spar`	A numeric value determining how exact a spline should be drawn. Default is 0.1

See Also

Other Plotting Functions: `interpol_view`, `plot_cloud`, `plot_development`, `plot_evolution`, `plot_fitness_evolution`, `plot_heatmap`, `plot_result`, `plot_viewshed`, `plot_windfarmGA`, `plot_windrose`, `random_search_single`

Examples

```
## Add some data examples from the package
load(file = system.file('extdata/resulthex.rda', package = 'windfarmGA'))

## Plot the results of a hexagonal grid optimization
plot_parkfitness(resulthex)
```

plot_random_search *Plot the result of a randomized output.*

Description

Plotting method for the results of random_search_single and random_search.

Usage

```
plot_random_search(resultRS, result, Polygon1, best)
```

Arguments

resultRS	The result of the random functions random_search_single and random_search.
result	The result of the function genetic_algorithm or windfarmGA
Polygon1	The Polygon for the wind farm area.
best	How many best candidates to plot. Default is 1.

See Also

Other Randomization: random_search_single, random_search

Examples

```
load(file = system.file("extdata/resultrect.rda", package = "windfarmGA"))
load(file = system.file("extdata/polygon.rda", package = "windfarmGA"))

Res = random_search(result = resultrect, Polygon1 = polygon)
plot_random_search(resultRS = Res, result = resultrect, Polygon1 = polygon, best=2)
```

plot_result *Plot the best results*

Description

Plot the best solutions of the genetic algorithm. Depending on plotEn, either the best energy or efficiency solutions can be plotted. best indicates the amount of best solutions to plot.

Usage

```
plot_result(result, Polygon1, best = 3, plotEn = 1,
  topographie = FALSE, Grid, Projection, sourceCCLRoughness, sourceCCL,
  weibullsrc)
```

Arguments

result	The output of windfarmGA or genetic_algorithm
Polygon1	The considered area as shapefile
best	A numeric value indicating how many of the best individuals should be plotted
plotEn	A numeric value that indicates if the best energy or efficiency output should be plotted. If (plotEn==1) plots the best energy solutions and (plotEn==2) plots the best efficiency solutions
topographie	A logical value, indicating whether terrain effects should be considered and plotted or not
Grid	The grid as SpatialPolygons, which is obtained from grid_area and used for plotting
Projection	A desired Projection can be used instead of the default Lambert Azimuthal Equal Area Projection
sourceCCLRoughness	The source to the adapted Corine Land Cover legend as .csv file. Only required when terrain effect model is activated. As default a .csv file within this package ('~/extdata/clc_legend.csv') is taken that was already adapted manually
sourceCCL	The source to the Corine Land Cover raster (.tif). Only required, when the terrain effect model is activated
weibullsrc	A list of Weibull parameter rasters, where the first list item must be the shape parameter raster k and the second item must be the scale parameter raster a of the Weibull distribution. If no list is given, then rasters included in the package are used instead, which currently only cover Austria.

Value

Returns a data.frame of the best (energy/efficiency) individual during all iterations

See Also

Other Plotting Functions: interpol_view, plot_cloud, plot_development, plot_evolution, plot_fitness_evolution, plot_heatmap, plot_parkfitness, plot_viewshed, plot_windfarmGA, plot_windrose, random_search_single

Examples

```
## Add some data examples from the package
load(file = system.file("extdata/resultrect.rda", package = "windfarmGA"))
load(file = system.file("extdata/resulthex.rda", package = "windfarmGA"))
load(file = system.file("extdata/polygon.rda", package = "windfarmGA"))

## Plot the results of a hexagonal grid optimization
Grid <- hexa_area(Polygon1, size = 87.5, FALSE)
plot_result(resulthex, polygon, best = 1, plotEn = 1, topographie = FALSE,
           Grid = Grid[[2]])
```

```
## Plot the results of a rectangular grid optimization
Grid <- grid_area(polygon, resol = 150, 1, FALSE)
plot_result(resultrect, polygon, best = 1, plotEn = 1, topographie = FALSE,
           Grid = Grid[[2]])
```

`plot_viewshed` *Plot viewshed results*

Description

Plot the result of `viewshed`

Usage

```
plot_viewshed(res, legend = FALSE, ...)
```

Arguments

`res`	The resulting list from viewshed
`legend`	Plot a legend? Default is FALSE
`...`	Is passed along to `plot`

See Also

Other Viewshed Analysis: `cansee`, `interpol_view`, `rasterprofile`, `viewTo`, `viewshed`

Other Plotting Functions: `interpol_view`, `plot_cloud`, `plot_development`, `plot_evolution`, `plot_fitness_evolution`, `plot_heatmap`, `plot_parkfitness`, `plot_result`, `plot_windfarmGA`, `plot_windrose`, `random_search_single`

Examples

```
## Not run:
library(sp)
library(raster)
Polygon1 <- Polygon(rbind(c(4488182, 2667172), c(4488182, 2669343),
                          c(4499991, 2669343), c(4499991, 2667172)))
Polygon1 <- Polygons(list(Polygon1), 1);
Polygon1 <- SpatialPolygons(list(Polygon1))
Projection <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
+ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)
DEM_meter <- getDEM(Polygon1)

turbloc = spsample(DEM_meter[[2]], 10, type = "random");
res <- viewshed(r = DEM_meter[[1]], shape = DEM_meter[[2]], turbine_locs = turbloc,
                h1 = 1.8, h2 = 50)
plot_viewshed(res)
```

```
## ... Arguments are past on to raster::plot
plot_viewshed(res, legend = T, interpolate=T, colNA="black",
              col = topo.colors(15))

## End(Not run)
```

---

plot_windfarmGA *Plot the results of an optimization run*

Description

Plot the results of a genetic algorithm run with given inputs. Several plots try to show all relevant effects and outcomes of the algorithm. 6 plot methods are available that can be selected individually.

Usage

```
plot_windfarmGA(result, Polygon1, GridMethod = "r", whichPl = "all",
  best = 1, plotEn = 1, Projection, weibullsrc)
```

Arguments

result	The output of windfarmGA or genetic_algorithm
Polygon1	The area as shapefile.
GridMethod	Which grid spacing method was used. Default is "rectangular". If hexagonal grid cells were used, assign any of the following arguments: "h","hexa", "hexagonal"
whichPl	Which plots should be shown: 1-6 are possible. The default is "all" which shows all available plots
best	A numeric value indicating how many of the best individuals should be plotted
plotEn	A numeric value that indicates if the best energy or efficiency output is plotted. If (plotEn==1) plots the best energy solutions and (plotEn==2) plots the best efficiency solutions
Projection	A desired Projection can be used instead of the default Lambert Azimuthal Equal Area Projection
weibullsrc	A list of Weibull parameter rasters, where the first list item must be the shape parameter raster ‘k‘ and the second item must be the scale parameter raster ‘a‘ of the Weibull distribution. If no list is given, then rasters included in the package are used instead, which currently only cover Austria.

See Also

Other Plotting Functions: interpol_view, plot_cloud, plot_development, plot_evolution, plot_fitness_evolution, plot_heatmap, plot_parkfitness, plot_result, plot_viewshed, plot_windrose, random_search_single

Examples

```
library(sp)
## Add some data examples from the package
load(file = system.file("extdata/resultrect.rda", package = "windfarmGA"))
load(file = system.file("extdata/resulthex.rda", package = "windfarmGA"))
load(file = system.file("extdata/polygon.rda", package = "windfarmGA"))

## Plot the results of a hexagonal grid optimization
plot_windfarmGA(resulthex, GridMethod = "h", polygon, whichPl = "all", best = 1, plotEn = 1)

## Plot the results of a rectangular grid optimization
plot_windfarmGA(resultrect, GridMethod = "r", polygon, whichPl = "all", best = 1, plotEn = 1)
```

`plot_windrose` *Plot a Windrose*

Description

Plot a wind rose of the wind data frame.

Usage

```
plot_windrose(data, spd, dir, spdres = 2, dirres = 10, spdmin = 1,
  spdmax = 30, palette = "YlGnBu", spdseq = NULL, plotit = TRUE)
```

Arguments

`data`	A data.frame containing the wind information
`spd`	The column of the wind speeds in "data"
`dir`	The column of the wind directions in "data"
`spdres`	The increment of the wind speed legend. Default is 2
`dirres`	The size of the wind sectors. Default is 10
`spdmin`	Minimum wind speed. Default is 1
`spdmax`	Maximal wind speed. Default is 30
`palette`	A color palette used for drawing the wind rose
`spdseq`	A wind speed sequence, that is used for plotting
`plotit`	Should the windrose be plotted? Default is TRUE

See Also

Other Plotting Functions: `interpol_view`, `plot_cloud`, `plot_development`, `plot_evolution`, `plot_fitness_evolution`, `plot_heatmap`, `plot_parkfitness`, `plot_result`, `plot_viewshed`, `plot_windfarmGA`, `random_search_single`

Examples

```
## Exemplary Input Wind speed and direction data frame
# Uniform wind speed and single wind direction
data.in <- data.frame(ws = 12, wd = 0)
windrosePlot <- plot_windrose(data = data.in, spd = data.in$ws,
   dir = data.in$wd)

# Random wind speeds and random wind directions
data.in <- data.frame(ws = sample(1:25, 10),
                      wd = sample(1:260, 10))
windrosePlot <- plot_windrose(data = data.in, spd = data.in$ws,
   dir = data.in$wd)
```

`random_search` *Randomize the output of the Genetic Algorithm*

Description

Perform a random search in the grid cells, to further optimize the output of the wind farm layout.

Usage

```
random_search(result, Polygon1, n, best, Plot, GridMethod,
  max_dist = 2.2)
```

Arguments

`result`	The resulting matrix of the function `genetic_algorithm` or `windfarmGA`
`Polygon1`	The Polygon for the wind farm area.
`n`	The number of random searches to be performed. Default is 20.
`best`	Which best individuals should be the starting conditions for a random search. The default is 1.
`Plot`	Should the random search be plotted? Default is FALSE
`GridMethod`	Should the polygon be divided into rectangular or hexagonal grid cells? The default is rectangular grid cells and hexagonal grid cells are computed when assigning "h" or "hexagon" to this input variable. The randomly generated points might be placed outside their hexagons.
`max_dist`	A numeric value multiplied by the rotor radius to perform collision checks. Default is 2.2

Value

Returns a list.

See Also

Other Randomization: plot_random_search, random_search_single

Examples

```
load(file = system.file("extdata/resultrect.rda", package = "windfarmGA"))
load(file = system.file("extdata/polygon.rda", package = "windfarmGA"))

new <- random_search(resultrect, polygon, n = 20, best = 4)
plot_random_search(resultRS = new, result = resultrect, Polygon1 = polygon, best = 2)
```

---

random_search_single *Randomize the location of a single turbine*

Description

Perform a random search for a single turbine, to further optimize the output of the wind farm layout.

Usage

```
random_search_single(result, Polygon1, n, Plot, GridMethod,
  max_dist = 2.2)
```

Arguments

result	The resulting matrix of the function genetic_algorithm or windfarmGA
Polygon1	The Polygon for the wind farm area.
n	The number of random searches to be performed. Default is 20.
Plot	Should the random search be plotted? Default is TRUE
GridMethod	Should the polygon be divided into rectangular or hexagonal grid cells? The default is rectangular grid cells and hexagonal grid cells are computed when assigning "h" or "hexagon" to this variable. The randomly generated points might be placed outside their hexagon.
max_dist	A numeric value multiplied by the rotor radius to perform collision checks. Default is 2.2

Value

Returns a list

See Also

Other Randomization: plot_random_search, random_search

Other Plotting Functions: interpol_view, plot_cloud, plot_development, plot_evolution, plot_fitness_evolution, plot_heatmap, plot_parkfitness, plot_result, plot_viewshed, plot_windfarmGA, plot_windrose

rasterprofile *Sample values from a raster*

Description

Sample a raster along a straight line between 2 points

Usage

```
rasterprofile(r, xy1, xy2, reso, plot = FALSE)
```

Arguments

r	A DEM raster
xy1	A matrix with X and Y coordinates for Point 1
xy2	A matrix with X and Y coordinates for Points 2
reso	The minimal resolution of the DEM raster. It is calculated in viewshed and passed along.
plot	Plot the process? Default is FALSE

Value

A boolean vector, indicating if Point 1 (xy1) is visible from all elements of Points 2 (xy2)

See Also

Other Viewshed Analysis: cansee, interpol_view, plot_viewshed, viewTo, viewshed

readinteger *Check Input Crossover Method*

Description

Checks whether the input for crossover is given correctly. If not, a message is prompted which asks to input one of the 2 available crossover methods. The available inputs are "E" and "R". "E" refers to partitioning at equal intervals and "R" refers to random partitioning.

Usage

```
readinteger()
```

Value

Returns the selected crossover method (character)

See Also

Other Helper Functions: dup_coords, getDEM, getISO3, get_grids, grid_area, hexa_area, isSpatial, permutations, readintegerSel, splitAt, tess2SPdf, windata_format

Examples

```
readinteger()
```

---

readintegerSel *Check Input Selection Method*

Description

Checks whether the input for selection is given correctly. If not, a message is prompted which asks to input one of the 2 available selection methods. The available inputs are "F" and "V". "F" refers to a fixed percentage of 50 percentage, based on the development of the population fitness values.

Usage

```
readintegerSel()
```

Value

Returns the selected selection method (character)

See Also

Other Helper Functions: dup_coords, getDEM, getISO3, get_grids, grid_area, hexa_area, isSpatial, permutations, readinteger, splitAt, tess2SPdf, windata_format

Examples

```
readintegerSel()
```

resultrect — *A resulting matrix of* genetic_algorithm *with 200 iterations and a rectangular shapefile* sp_polygon

Description

A resulting matrix of genetic_algorithm with 200 iterations and a rectangular shapefile sp_polygon

Usage

```
resultrect
```

Format

An object of class matrix with 200 rows and 13 columns.

selection — *Selection Method*

Description

Select a certain amount of individuals and recombine them to parental teams. Add the mean fitness value of both parents to the parental team. Depending on the selected selstate, the algorithm will either take always 50 percent or a variable percentage of the current population. The variable percentage depends on the evolution of the populations fitness values.

Usage

```
selection(fit, Grid, teil, elitism, nelit, selstate, verbose)
```

Arguments

fit	A list of all fitness-evaluated individuals
Grid	Is the indexed grid output from grid_area
teil	A numeric value that determines the selection percentage
elitism	Boolean value which indicates whether elitism should be included or not.
nelit	If elitism is TRUE, then this input variable determines the amount of individuals in the elite group.
selstate	Determines which selection method is used, "FIX" selects a constant percentage and "VAR" selects a variable percentage, depending on the development of the fitness values.
verbose	If TRUE, will print out further information.

Value

Returns list with 2 elements. Element 1 is the binary encoded matrix which shows all selected individuals. Element 2 represent the mean fitness values of each parental team.

See Also

Other Genetic Algorithm Functions: crossover, fitness, genetic_algorithm, init_population, mutation, trimton, windfarmGA

Examples

```
## Not run:
## Create a random rectangular shapefile
library(sp)
Polygon1 <- Polygon(rbind(c(0, 0), c(0, 2000), c(2000, 2000), c(2000, 0)))
Polygon1 <- Polygons(list(Polygon1),1);
Polygon1 <- SpatialPolygons(list(Polygon1))
Projection <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
+ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)

## Calculate a Grid and an indexed data.frame with coordinates and grid cell Ids.
Grid1 <- grid_area(shape = Polygon1,resol = 200,prop = 1);
Grid <- Grid1[[1]]
AmountGrids <- nrow(Grid)

startsel <- init_population(Grid,10,20);
wind <- as.data.frame(cbind(ws=12,wd=0))
wind <- list(wind, probab = 100)
fit <- fitness(selection = startsel, referenceHeight = 100, RotorHeight=100,
               SurfaceRoughness=0.3,Polygon = Polygon1, resol1 = 200,
               rot = 20, dirspeed = wind,
               srtm_crop = "", topograp = FALSE, cclRaster = "")
allparks <- do.call("rbind",fit);

## SELECTION
## print the amount of Individuals selected. Check if the amount
## of Turbines is as requested.
selec6best <- selection(fit, Grid, 2, T, 6, "VAR")
selec6best <- selection(fit, Grid, 2, T, 6, "FIX")
selec6best <- selection(fit, Grid, 4, F, 6, "FIX")

## End(Not run)
```

---

splitAt *Divide matrices or integer at certain locations*

Description

Required function for the crossover method to split a genetic code at random intervals. See also crossover.

Usage

```
splitAt(x, pos)
```

Arguments

x	A numeric variable representing the binary genetic code of an individual (numeric)
pos	A numeric value which shows at which position the genetic code is cut (numeric)

Value

Returns a list of the splitted genetic code.

See Also

Other Helper Functions: dup_coords, getDEM, getISO3, get_grids, grid_area, hexa_area, isSpatial, permutations, readintegerSel, readinteger, tess2SPdf, windata_format

Examples

```
splitAt(1:100,20)
splitAt(as.matrix(1:100),20)
```

---

sp_polygon *The rectangular shapefile used to create* resultrect

---

Description

The rectangular shapefile used to create resultrect

Usage

```
sp_polygon
```

Format

An object of class SpatialPolygons of length 1.

tess2SPdf *Create a Tesselation from a Polygon*

Description

Returns a Spatial Polygons object from a Tesselation object.

Usage

```
tess2SPdf(x)
```

Arguments

x	The Tesselation object (Tessellation)

Value

Returns a SpatialPolygons. (SpatialPolygons)

See Also

Other Helper Functions: dup_coords, getDEM, getISO3, get_grids, grid_area, hexa_area, isSpatial, permutations, readintegerSel, readinteger, splitAt, windata_format

trimton *Adjust the amount of turbines per windfarm*

Description

Adjust the mutated individuals to the required amount of turbines.

Usage

```
trimton(mut, nturb, allparks, nGrids, trimForce, seed)
```

Arguments

mut	A binary matrix with the mutated individuals
nturb	A numeric value indicating the amount of required turbines
allparks	A data.frame consisting of all individuals of the current generation
nGrids	A numeric value indicating the total amount of grid cells
trimForce	A boolean value which determines which adjustment method should be used. TRUE uses a probabilistic approach and FALSE uses a random approach
seed	Set a seed for comparability. Default is NULL

Value

Returns a binary matrix with the correct amount of turbines per individual

See Also

Other Genetic Algorithm Functions: crossover, fitness, genetic_algorithm, init_population, mutation, selection, windfarmGA

Examples

```
## Create a random rectangular shapefile
library(sp)
Polygon1 <- Polygon(rbind(c(0, 0), c(0, 2000), c(2000, 2000), c(2000, 0)))
Polygon1 <- Polygons(list(Polygon1),1);
Polygon1 <- SpatialPolygons(list(Polygon1))
Projection <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
+ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)

## Create a uniform and unidirectional wind data.frame and plots the
## resulting wind rose
## Uniform wind speed and single wind direction
data.in <- as.data.frame(cbind(ws=12,wd=0))

## Calculate a Grid and an indexed data.frame with coordinates and grid cell Ids.
Grid1 <- grid_area(shape = Polygon1,resol = 200,prop = 1);
Grid <- Grid1[[1]]
AmountGrids <- nrow(Grid)

startsel <- init_population(Grid,10,20);
wind <- as.data.frame(cbind(ws=12,wd=0))
wind <- list(wind, probab = 100)
fit <- fitness(selection = startsel,referenceHeight = 100, RotorHeight=100,
           SurfaceRoughness=0.3,Polygon = Polygon1, resol1 = 200,rot=20, dirspeed = wind,
              srtm_crop="",topograp=FALSE,cclRaster="")
allparks <- do.call("rbind",fit);

## SELECTION
## print the amount of Individuals selected.
## Check if the amount of Turbines is as requested.
selec6best <- selection(fit, Grid,2, T, 6, "VAR");
selec6best <- selection(fit, Grid,2, T, 6, "FIX");
selec6best <- selection(fit, Grid,4, F, 6, "FIX");

## CROSSOVER
## u determines the amount of crossover points,
## crossPart determines the method used (Equal/Random),
## uplimit is the maximum allowed permutations
crossOut <- crossover(selec6best, 2, uplimit = 300, crossPart="RAN");
crossOut <- crossover(selec6best, 7, uplimit = 500, crossPart="RAN");
crossOut <- crossover(selec6best, 3, uplimit = 300, crossPart="EQU");
```

```
## MUTATION
## Variable Mutation Rate is activated if more than 2 individuals represent
## the current best solution.
mut <- mutation(a = crossOut, p = 0.3, NULL);

## TRIMTON
## After Crossover and Mutation, the amount of turbines in a windpark change and have to be
## corrected to the required amount of turbines.
mut1 <- trimton(mut = mut, nturb = 10, allparks = allparks, nGrids = AmountGrids,
                trimForce=FALSE)
colSums(mut)
colSums(mut1)
```

`turbine_influences` *Find potentially influencing turbines*

Description

Find all turbines that could potentially influence another turbine and save them to a list.

Usage

```
turbine_influences(t, wnkl, dist, polYgon, dirct, plotAngles = FALSE)
```

Arguments

`t`	A data.frame of the current individual with X and Y coordinates
`wnkl`	A numeric value indicating the angle, at which no wake influences are considered. Default is 20 degrees.
`dist`	A numeric value indicating the distance, after which the wake effects are considered to be eliminated. Default is 100km.
`polYgon`	A shapefile representing the considered area
`dirct`	A numeric value indicating the current wind direction
`plotAngles`	A logical variable, which is used to plot the distances and angles. Default is FALSE.

Value

Returns a list of all individuals of the current generation which could potentially influence other turbines. List includes the relevant coordinates, the distances and angles in between and assigns the Point ID.

See Also

Other Wind Energy Calculation Functions: `barometric_height`, `calculate_energy`, `get_dist_angles`

Examples

```
library(sp)
library(raster)
## Exemplary input Polygon with 2km x 2km:
polYgon <- Polygon(rbind(c(0, 0), c(0, 2000),
c(2000, 2000), c(2000, 0)))
polYgon <- Polygons(list(polYgon), 1)
polYgon <- SpatialPolygons(list(polYgon))
Projection <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
+ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(polYgon) <- CRS(Projection)

t <- as.matrix(cbind(x = runif(10, 0, extent(polYgon)[2]),
     y = runif(10, 0, extent(polYgon)[4])))
wnkl <- 20
dist <- 100000
dirct <- 0

res <- turbine_influences(t, wnkl, dist, polYgon, dirct, plotAngles = TRUE)
```

`viewshed` *Calculate visibility*

Description

Calculate visibility for given points in a given area.

Usage

```
viewshed(r, shape, turbine_locs, h1 = 0, h2 = 0)
```

Arguments

`r`	A DEM raster
`shape`	A SpatialPolygon of the windfarm area.
`turbine_locs`	Coordinates or SpatialPoint representing the wind turbines
`h1`	A numeric giving the extra height offset of Point 1
`h2`	A numeric giving the extra height offset of Point 2

Value

A list of 5, containing the boolean result for every cell, the raster cell points, a SimpleFeature Polygon of the given area and the DEM raster

See Also

Other Viewshed Analysis: `cansee`, `interpol_view`, `plot_viewshed`, `rasterprofile`, `viewTo`

Examples

```
## Not run:
library(sp)
Polygon1 <- Polygon(rbind(c(4488182, 2667172), c(4488182, 2669343),
                          c(4499991, 2669343), c(4499991, 2667172)))
Polygon1 <- Polygons(list(Polygon1), 1);
Polygon1 <- SpatialPolygons(list(Polygon1))
Projection <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
+ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)
DEM_meter <- getDEM(Polygon1)

turbloc = spsample(DEM_meter[[2]], 10, type = "random");
res <- viewshed(r = DEM_meter[[1]], shape=DEM_meter[[2]], turbine_locs = turbloc, h1=1.8, h2=50)

## End(Not run)
```

---

viewTo *Calculate Visibility between multiple locations*

Description

Check if a location is visible from multiple locations

Usage

```
viewTo(r, xy1, xy2, h1 = 0, h2 = 0, reso)
```

Arguments

r	A DEM raster
xy1	A vector/matrix with X and Y coordinates for Point 1
xy2	A vector/matrix with X and Y coordinates for Point 2
h1	A numeric giving the extra height offset for Point 1
h2	A numeric giving the extra height offset for Point 2
reso	The minimal resolution of the DEM raster. It is calculated in viewshed and passed along.

Value

A boolean vector, indicating if xy1 is visible from all elements of xy2

See Also

Other Viewshed Analysis: cansee, interpol_view, plot_viewshed, rasterprofile, viewshed

windata_format *Transform Winddata*

Description

Helper Function, which transforms winddata to an acceptable format

Usage

```
windata_format(df)
```

Arguments

df	The wind data with speeds, direction and optionally a probability column. If not assigned, it will be calculated

Value

A list of windspeed and probabilities

See Also

Other Helper Functions: dup_coords, getDEM, getISO3, get_grids, grid_area, hexa_area, isSpatial, permutations, readintegerSel, readinteger, splitAt, tess2SPdf

Examples

```
wind_df <- data.frame(ws = c(12, 30, 45),
                      wd = c(0, 90, 150),
                      probab = 30:32)
windata_format(wind_df)

wind_df <- data.frame(speed = c(12, 30, 45),
                      direction = c(90, 90, 150)
                      ,probab = c(10, 20, 60)
)
windata_format(wind_df)

wind_df <- data.frame(speed = c(12, 30, 45),
                      direction = c(400, 90, 150)
)
windata_format(wind_df)
```

windfarmGA *Run a Genetic Algorithm to optimize a wind farm layout*

Description

The initiating function of an optimization run which will interactively check user-inputs. If all inputs are correct, an optimization will be started.

Usage

```
windfarmGA(dns, layer, Polygon1, GridMethod, Projection, sourceCCL,
  sourceCCLRoughness, vdirspe, Rotor = 30, fcrR = 3, n = 10,
  topograp = FALSE, iteration = 20, referenceHeight = 50,
  RotorHeight = 50, SurfaceRoughness = 0.14, Proportionality = 1,
  mutr = 0.008, elitism = TRUE, nelit = 7, selstate = "FIX",
  crossPart1 = "EQU", trimForce = TRUE, weibull, weibullsrc, Parallel,
  numCluster, verbose = FALSE, plotit = FALSE)
```

Arguments

dns	The data source name (interpretation varies by driver — for some drivers, dsn is a file name, but may also be a folder)
layer	The layer name
Polygon1	The considered area as SpatialPolygon, SimpleFeature Polygon or coordinates as matrix/data.frame
GridMethod	Should the polygon be divided into rectangular or hexagonal grid cells? The default is "Rectangular" grid cells and hexagonal grid cells are computed when assigning "h" or "hexagon" to this input variable.
Projection	A desired Projection can be used instead of the default Lambert Azimuthal Equal Area Projection (EPSG:3035).
sourceCCL	The path to the Corine Land Cover raster (.tif). Only required when the terrain effect model is activated. If nothing is assign, it will try to download a version from the EEA-website.
sourceCCLRoughness	The source to the adapted Corine Land Cover legend as .csv file. Only required when terrain effect model is activated. As default a .csv file within this package ('~/extdata') is taken that was already adapted manually. To use your own .csv legend this variable has to be assigned.
vdirspe	A data.frame containing the incoming wind speeds, wind directions and probabilities
Rotor	A numeric value that gives the rotor radius in meter
fcrR	A numeric value that is used for grid spacing. Default is 5
n	A numeric value indicating the required amount of turbines

`topograp` Logical value, which indicates if the terrain effect model should be enabled or not. Default is FALSE

`iteration` A numeric value indicating the desired amount of iterations of the algorithm. Default is 20

`referenceHeight`
The height at which the incoming wind speeds were measured. Default is the RotorHeight.

`RotorHeight` The desired height of the turbine.

`SurfaceRoughness`
A surface roughness length of the considered area in m. If the terrain effect model is activated, a surface roughness will be calculated for every grid cell with the elevation and land cover information. Default is 0.3

`Proportionality`
A numeric value used for grid calculation. Determines the percentage a grid has to overlay. Default is 1

`mutr` A numeric mutation rate with a default value of 0.008

`elitism` Boolean value, which indicates whether elitism should be activated or not. Default is TRUE

`nelit` If `elitism` is TRUE, this input determines the amount of individuals in the elite group. Default is 7

`selstate` Determines which selection method is used, "FIX" selects a constant percentage and "VAR" selects a variable percentage, depending on the development of the fitness values. Default is "FIX"

`crossPart1` Determines which crossover method is used, "EQU" divides the genetic code at equal intervals and "RAN" divides the genetic code at random locations. Default is "EQU"

`trimForce` If activated (`trimForce == TRUE`), the algorithm will take a probabilistic approach to trim the windfarms to the desired amount of turbines. If deactivated (`trimForce == FALSE`) the adjustment will be random. Default is FALSE

`weibull` A logical value that specifies whether to take Weibull parameters into account. If ‘weibull == TRUE‘, the wind speed values from the ‘vdirspe‘ data frame are ignored. The algorithm will calculate the mean wind speed for every wind turbine according to the Weibull parameters. Default is FALSE

`weibullsrc` A list of Weibull parameter rasters, where the first list item must be the shape parameter raster ‘k‘ and the second item must be the scale parameter raster ‘a‘ of the Weibull distribution. If no list is given, then rasters included in the package are used instead, which currently only cover Austria. This variable is only used if ‘weibull == TRUE‘.

`Parallel` Boolean value, indicating whether parallel processing should be used. The parallel and doParallel packages are used for parallel processing. Default is FALSE

`numCluster` If `Parallel` is TRUE, this variable defines the number of clusters to be used

`verbose` If TRUE it will print information for every generation. Default is FALSE

`plotit` If TRUE it will plot the best windfarm of every generation. Default is FALSE

Details

A terrain effect model can be included in the optimization process. Therefore, an SRTM elevation model will be downloaded automatically via the `raster::getData` function. A land cover raster can also be downloaded automatically from the EEA-website, or the path to a raster file can be passed to `sourceCCL`. The algorithm uses an adapted version of the Raster legend ("clc_legend.csv"), which is stored in the package directory '`~/inst/extdata`'. To use other values for the land cover roughness lengths, insert a column named **"Rauhigkeit_z"** to the .csv file, assign a surface roughness length to all land cover types. Be sure that all rows are filled with numeric values and save the file with **";"** separation. Assign the path of the file to the input variable `sourceCCLRoughness` of this function.

Value

The result is a matrix with aggregated values per generation, the best individual regarding energy and efficiency per generation, some fuzzy control variables per generation, a list of all fitness values per generation, the amount of individuals after each process, a matrix of all energy, efficiency and fitness values per generation, the selection and crossover parameters, a matrix with the generational difference in maximum and mean energy output, a matrix with the given inputs, a dataframe with the wind information, the mutation rate per generation and a matrix with all tested wind farm layouts.

See Also

`genetic_algorithm`

Other Genetic Algorithm Functions: `crossover`, `fitness`, `genetic_algorithm`, `init_population`, `mutation`, `selection`, `trimton`

`windfarmGA_` *windfarmGA*

Description


A package to optimize small wind farms with irregular shapes using a genetic algorithm. It requires a fixed amount of turbines, a fixed rotor radius and an average wind speed value for each incoming wind direction. A terrain effect model can be included that downloads an 'SRTM' elevation model and loads a Corine Land Cover raster to approximate surface roughness. Further information can be found in the description of the function `windfarmGA`.

Author(s)

Maintainer: Sebastian Gatscha <sebastian_gatscha@gmx.at>

See Also

Useful links:

- Documentation Github.io
- Documentation
- Master Thesis
- Shiny App
- Report Issues

Index