

Package ‘windfarmGA’

February 19, 2019

Title Genetic Algorithm for Wind Farm Layout Optimization

Version 2.2.1

Date 2019-01-30

Author Sebastian Gatscha

Maintainer Sebastian Gatscha <sebastian_gatscha@gmx.at>

Description The genetic algorithm is designed to optimize small wind farms with irregular shapes. The algorithm works with a fixed amount of turbines, a fixed rotor radius and a mean wind speed value for every incoming wind direction. If required, it can include a terrain effect model which downloads an 'SRTM' elevation model automatically and loads a Corine Land Cover raster, which has to be downloaded previously. Further information can be found in the description of the function 'windfarmGA'. To start an optimization run, either the function 'windfarmGA' or 'genAlgo' can be used. The function 'windfarmGA' checks the user inputs interactively and then runs the function 'genAlgo'. If the input parameters are already known, an optimization can be run directly via the function 'genAlgo'. Their output is identical.

Depends R (>= 3.2.3)

Imports rgdal (>= 1.0-4), Rcpp (>= 1.0.0), raster (>= 2.8.19), sf (>= 0.7.2), rayshader (>= 0.5.1), plyr (>= 1.8.4), rworldmap (>= 1.3.6), sp (>= 1.2-2), maptools (>= 0.8-36), calibrate (>= 1.7.2), gstat (>= 1.0-26), ggplot2 (>= 2.2.0), RColorBrewer (>= 1.1-2), rgeos (>= 0.3-11), magrittr (>= 1.5), foreach (>= 1.4.3), parallel (>= 3.4.1), doParallel (>= 1.0.11), leaflet (>= 1.1.0), methods (>= 3.2.5), spatstat (>= 1.42-2), grDevices, graphics, stats, utils

LinkingTo Rcpp

LazyData TRUE

License MIT + file LICENSE

URL <https://github.com/YsoSirius/windfarmGA>

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

RoxygenNote 6.0.1

Suggests testthat

Encoding UTF-8

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-02-19 00:10:03 UTC

R topics documented:

BaroHoehe	3
calculateEn	4
cansee	6
crossover1	7
dup_coords	8
fitness	9
genAlgo	10
getDEM	14
getISO3	15
getRects	16
GridFilter	17
heatmapGA	19
HexaTex	20
InfluPoints	21
interpol_view	22
isSpatial	24
leafPlot	25
mutation	26
permutations	27
plotbeorwor	27
plotCloud	28
plotEvolution	29
plotfitnesssevolution	30
plotparkfitness	30
plotResult	31
PlotWindfarmGA	33
plotWindrose	34
plot_farm_3d	35
plot_viewshed	36
RandomSearch	37
RandomSearchPlot	38
RandomSearchTurb	39
rasterprofile	40
readinteger	40
readintegerSel	41
selection1	41
splitAt	43
StartGA	44

<i>BaroHoehe</i>	3
tess2SPdf	45
trimton	46
VekWinkelCalc	47
viewshed	49
viewTo	50
windata_format	50
windfarmGA	51
Index	53

BaroHoehe	<i>Calculates Air Density, Air Pressure and Temperature according to the Barometric Height Formula</i>
-----------	--------------------------------------------------------------------------------------------------------

Description

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

Usage

```
BaroHoehe(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.

Author(s)

Sebastian Gatscha

Examples

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

 calculateEn

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

```
calculateEn(sel, referenceHeight, RotorHeight, SurfaceRoughness, wnk1, distanz,
  polygon1, resol, RotorR, dirSpeed, srtm_crop, topograp, cclRaster, weibull)
```

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
wnk1	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 genAlgo
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

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.

Author(s)

Sebastian Gatscha

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 +units=m +no_defs'
proj4string(Polygon1) <- CRS(Projection)
plot(Polygon1,axes=TRUE)

## Create a uniform and unidirectional wind data.frame and plot the
## resulting wind rose
data.in <- data.frame(ws = 12, wd = 0)
windrosePlot <- plotWindrose(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 <- GridFilter(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.
resStartGA <- StartGA(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(resStartGA[[1]][, 'X'],resStartGA[[1]][, 'Y'], pch=20,cex=2)
plot(resGrid[[2]], add = TRUE)
resCalcEn <- calculateEn(sel=resStartGA[[1]],referenceHeight= 50,
                        RotorHeight= 50, SurfaceRoughness = 0.14,wnk1 = 20,
                        distanz = 100000, resol = 200,dirSpeed = data.in,
                        RotorR = 50, polygon1 = Polygon1, topograp = FALSE,
                        weibull = FALSE)

length(resCalcEn)
str(resCalcEn)
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 <- plotWindrose(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 <- calculateEn(sel=resStartGA[[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)

length(resCalcEn)
str(resCalcEn)

## End(Not run)

```

cansee

cansee

Description

Check if point1 (xy1) visible from point2 (xy2) given a certain DEM (r)

Usage

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

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 of Point 1
h2	A numeric giving the extra height offset of Point 2

Value

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

Author(s)

Sebastian Gatscha

crossover1	<i>Crossover Method</i>
------------	-------------------------

Description

The crossover method of the genetic algorithm, which takes the selected individuals after the `selection1` function and produces new offsprings through permutation.

Usage

```
crossover1(se6, u, uplimit, crossPart, verbose, seed)
```

Arguments

<code>se6</code>	The selected individuals. The output of <code>selection1</code> (list)
<code>u</code>	The crossover point rate. (numeric)
<code>uplimit</code>	The upper limit of allowed permutations. The current algorithm has an upper bound of 300 permutations. (numeric)
<code>crossPart</code>	The crossover method. Either "EQU" or "RAN". (character)
<code>verbose</code>	If TRUE, will print out further information.
<code>seed</code>	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)

Author(s)

Sebastian Gatscha

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))
)
Parents

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

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

```
## Cross their data at equal locations with 2 crossover parts
crossover1(CrossSampl, u = 1.1, uplimit = 300, crossPart = "EQU")

## with 3 crossover parts and equal locations
crossover1(CrossSampl, u = 2.5, uplimit = 300, crossPart = "EQU")

## or with random locations and 5 crossover parts
crossover1(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 [heatmapGA](#)

Usage

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

Arguments

x	Two column numeric matrix or data frame
...	passed to <code>sapply</code> . If <code>simplify = TRUE</code> (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 duplicated locations. Otherwise, the default method returns a list where each component is a vector with the positions or the rownames, if available, of the duplicated 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 duplicated 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>

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 [calculateEn](#). 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 genAlgo
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.

Author(s)

Sebastian Gatscha

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 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)
plot(Polygon1,axes=TRUE)

## 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 <- plotWindrose(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 <- GridFilter(shape = Polygon1,resol = 200,prop = 1);
Grid <- Grid1[[1]]
AmountGrids <- nrow(Grid)

wind <- list(wind, probab = 100)
startsel <- StartGA(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)

head(fit)

```

genAlgo

*Start a Genetic Algorithm for a wind farm layout optimization***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

```
genAlgo(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 low 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 activated or not. Default is FALSE
elitism	Boolean value, which indicates whether elitism should be included or not. Default is TRUE
nelit	If elitism is TRUE, then this input variable 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 'dirSpeed' 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, will print out information for every generation Default is FALSE
plotit	If TRUE, will plot the best windfarm of a 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 paramters, 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 matrix with all tested wind farm layouts.

Author(s)

Sebastian Gatscha

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 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)
plot(Polygon1,axes=TRUE)

## Create a uniform and unidirectional wind data.frame and plot the
## resulting wind rose
data.in <- data.frame(ws = 12, wd = 0)
windrosePlot <- plotWindrose(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 <- genAlgo(Polygon1 = Polygon1,
                  n = 12,
                  vdirspe = data.in,
                  Rotor = 30,
                  RotorHeight = 100)
PlotWindfarmGA(result = result, Polygon1 = Polygon1)

## Runs the same optimization, but with parallel processing and 3 cores.
result_par <- genAlgo(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)
PlotWindfarmGA(result = result_par, GridMethod = "h", Polygon1 = Polygon1)

## Runs the same optimization, this time with hexagonal grids.
result_hex <- genAlgo(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)
PlotWindfarmGA(result = result_hex, GridMethod = "h", Polygon1 = Polygon1)

```

```

## Run an optimization with the Weibull parameters included in the package.
result_weibull <- genAlgo(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)
PlotWindfarmGA(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 <- genAlgo(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)
#PlotWindfarmGA(result = result_weibull, GridMethod= "h", Polygon1 = Polygon1)

```

getDEM

getDEM

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

Author(s)

Sebastian Gatscha

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 rworldmap package

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

Author(s)

Sebastian Gatscha

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)
```

getRects

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

```
getRects(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.

Author(s)

Sebastian Gatscha

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 <- GridFilter(shape = Polygon1,resol = 200,prop = 1);
Grid <- Grid1[[1]]
AmountGrids <- nrow(Grid)

startsel <- StartGA(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 <- selection1(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 <- crossover1(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 <- getRects(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)

```

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

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

Arguments

shape	Shape file of the considered area
resol	The resolution of the grid in meter. Default is 500. (numeric)
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 $\text{Rotor} * \text{fcrR}$. 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>

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)
plot(Polygon1,axes=TRUE)

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

## Exemplary 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)
plot(Polygon1,axes=TRUE)

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

```

heatmapGA

Plot heatmap of fit grid cells

Description

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

Usage

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

Arguments

result	The output matrix of <code>windfarmGA</code> or <code>genAlgo</code> , which has stored all relevant information
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

Author(s)

Sebastian Gatscha

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
result <- resulthex
heatmapGA(result)

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

HexaTex

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

```
HexaTex(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

Author(s)

Sebastian Gatscha

Examples

```

library(spatstat)
library(maptools)
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)
plot(Polygon1,axes=TRUE)
HexGrid <- HexaTex(Polygon1, 100, TRUE)
plot(HexGrid[[2]])
str(HexGrid[[1]])

```

InfluPoints

Find potentially influencing turbines

Description

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

Usage

```
InfluPoints(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.

Author(s)

Sebastian Gatscha

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)))
polyYgon <- Polygons(list(polyGon), 1)
polyYgon <- SpatialPolygons(list(polyYgon))
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(polyYgon) <- CRS(Projection)
plot(polyYgon, axes = TRUE)

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

resInfluPoi <- InfluPoints(t, wnkl, dist, polyYgon, dirct, plotAngles = TRUE)
str(resInfluPoi)
```

interpol_view

interpol_view

Description

Plot an interpolated view of the viewshed analysis

Usage

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

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
...	Arguments passed on to plot .

Value

An interpolated raster

Author(s)

Sebastian Gatscha

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)

sample_POI <- spsample(DEM_meter[[2]], n = ncell(DEM_meter[[1]]),
                       type = "regular")
sample_xy <- coordinates(sample_POI)

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)

## ... 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

Author(s)

Sebastian Gatscha

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 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)
df_fort <- ggplot2::fortify(Polygon1)
isSpatial(df_fort, Projection)
```

leafPlot *Leaflet Plot of a Wind Park*

Description

Plot a resulting wind farm on a leaflet map.

Usage

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

Arguments

result	The resulting matrix of the function 'genAlgo' 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 GridFilter or HexaTex functions.

Value

Returns a leaflet map.

Author(s)

Sebastian Gatscha

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)
leafPlot(result = resulthex, Polygon1 = polygon, which = 1)

## Plot the last wind farm (ordered by chronology).
leafPlot(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 <- GridFilter(polygon, resol = 150, prop = 0.4)
leafPlot(result = resultrect, Polygon1 = polygon, which = 1,
         GridPol = Grid[[2]])
```

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

 mutation

Mutation Method

Description

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

Usage

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

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.

Author(s)

Sebastian Gatscha

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)
## Check which values are not like the originals
a==aMut

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

permutations	<i>Enumerate the Combinations or Permutations of the Elements of a Vector</i>
--------------	-------------------------------------------------------------------------------

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/>

plotbeorwor	<i>Plot if previous population was better or worse</i>
-------------	--------------------------------------------------------

Description

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

Usage

```
plotbeorwor(result)
```

Arguments

result The output matrix of `windfarmGA` or `genAlgo`, which has stored all relevant information.

Author(s)

Sebastian Gatscha

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
result <- resulthex
plotbeorwor(result)
```

plotCloud

Plot outputs of all generations with standard deviations

Description

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

Usage

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

Arguments

result The output matrix of `windfarmGA` or `genAlgo`, which has stored all relevant information.

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.

Author(s)

Sebastian Gatscha

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
result <- resulthex
plcdf <- plotCloud(result, TRUE)
str(plcdf)
```

plotEvolution *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

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

Arguments

result	The output matrix of <code>windfarmGA</code> or <code>genAlgo</code> , which has stored all relevant information.
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

Author(s)

Sebastian Gatscha

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
result <- resultrect
plotEvolution(result, ask = TRUE, spar = 0.1)
```

plotfitnesssevolution *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

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

Arguments

result	An output matrix of the function <code>windfarmGA</code> or <code>genAlgo</code> which has stored all relevant information.
spar	A numeric value determining how exact a spline should be drawn. Default is 0.1

Author(s)

Sebastian Gatscha

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
result <- resulthex
plotfitnesssevolution(result, 0.1)
```

plotparkfitness *Plot the genetic algorithm results*

Description

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

Usage

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

Arguments

result	An output matrix of <code>windfarmGA</code> or <code>genAlgo</code> , which has stored all relevant information.
spar	A numeric value determining how exact a spline should be drawn. Default is 0.1

Author(s)

Sebastian Gatscha

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
result <- resulthex
plotparkfitness(result)
```

plotResult

Plot the best Results

Description

Plot the best resulting 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 that should be plotted.

Usage

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

Arguments

result	An output matrix of the function <code>windfarmGA</code> or <code>genAlgo</code> , which has stored all relevant information.
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 (<code>plotEn==1</code>) plots the best energy solutions and (<code>plotEn==2</code>) 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 <code>GridFilter</code> 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. This variable is only used if <code>weibull==TRUE</code> .

Value

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

Author(s)

Sebastian Gatscha

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
result <- resulthex
Polygon1 <- polygon
Grid <- HexaTex(Polygon1, size = 87.5, FALSE)
plotResult(result, Polygon1, best = 1, plotEn = 1, topographie = FALSE,
           Grid = Grid[[2]])

## Plot the results of a rectangular grid optimization
result <- resultrect
Polygon1 <- polygon
Grid <- GridFilter(Polygon1, resol = 175, 1, FALSE)
plotResult(result, Polygon1, best = 1, plotEn = 1, topographie = FALSE,
           Grid = Grid[[2]])

## Plot the results of with a weibull mean background
result <- resultrect
Polygon1 <- polygon
a_param <- readRDS(system.file("extdata/a_weibull.RDS", package = "windfarmGA"))
```



```

k_param <- readRDS(system.file("extdata/k_weibull.RDS", package = "windfarmGA"))
weibullsrc <- list(k_param, a_param)
plotResult(result, Polygon1, best = 2, plotEn = 2, topographie = FALSE,
           Grid = Grid[[2]], weibullsrc = weibullsrc)

## Plot the hexagonal results ith weibull mean background
result <- resulthex
Grid <- HexaTex(Polygon1, size = 87.5, FALSE)
plotResult(result, Polygon1, best = 2, plotEn = 2, topographie = FALSE,
           Grid = Grid[[2]], weibullsrc = weibullsrc)

```

PlotWindfarmGA

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

```

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

```

Arguments

result	An output matrix of the function <code>windfarmGA</code> or <code>genAlgo</code> , which has stored all relevant information.
Polygon1	The considered area as shapefile. Only required if the shapefile is already loaded.
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 should be 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. This variable is only used if weibull==TRUE.

Author(s)

Sebastian Gatscha

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
result <- resulthex
Polygon1 <- polygon
PlotWindfarmGA(result, GridMethod = "h", Polygon1, whichPl = "all", best = 1, plotEn = 1)

## Plot the results of a rectangular grid optimization
result <- resultrect
Polygon1 <- polygon
PlotWindfarmGA(result, GridMethod = "r", Polygon1, whichPl = "all", best = 1, plotEn = 1)

## Plot the results with hexagonal grid cells and a weibull mean background.
a_param <- readRDS(system.file("extdata/a_weibull.RDS", package = "windfarmGA"))
k_param <- readRDS(system.file("extdata/k_weibull.RDS", package = "windfarmGA"))
weibullsrc <- list(k_param, a_param)
PlotWindfarmGA(result, GridMethod = "h", Polygon1, whichPl = "all",
               best = 1, plotEn = 1, weibullsrc = weibullsrc)
```

plotWindrose

Plot a Windrose

Description

Plot a wind rose of the wind data frame.

Usage

```
plotWindrose(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

Author(s)

Sebastian Gatscha

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 <- plotWindrose(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 <- plotWindrose(data = data.in, spd = data.in$ws,
  dir = data.in$wd)
```

plot_farm_3d

plot_farm_3d

Description

Plot a wind farm with rayshader in 3D. (Still experimental)

Usage

```
plot_farm_3d(DEM, turbines, zscale_sphere = 200, zscale_3d = 200,
  texture = "imhof1", z = 1, sunangle = 315, ...)
```

Arguments

DEM	The ISO3 code of the country
turbines	boolean, indicating if polygon should be cropped. Default is TRUE
zscale_sphere	zscale for rayshader::sphere_shade. Default is 200.
zscale_3d	zscale for rayshader::plot_3d Default is 200.
texture	texture for rayshader::sphere_shade.
z	zoom value. Default is 1
sunangle	Default '315' (NW). The direction of the main highlight color.
...	Arguments passed on to rayshader::plot_3d

Author(s)

Sebastian Gatscha

Examples

```
## Not run:
library(sp)
Polygon1 <- Polygon(rbind(c(4488182, 2663172), c(4488182, 2669343),
                          c(4499991, 2669343), c(4499991, 2663172)))
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");

plot_farm_3d(DEM_meter[[1]], turbloc, texture="imhof4")

## End(Not run)
```

plot_viewshed

plot_viewshed

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

Author(s)

Sebastian Gatscha

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 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)
DEM_meter <- getDEM(Polygon1)

sample_POI <- spsample(DEM_meter[[2]], n = ncell(DEM_meter[[1]]), type = "regular")
sample_xy <- coordinates(sample_POI)

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)

## End(Not run)
```

RandomSearch

RandomSearch Randomise 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

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

Arguments

result	The resulting matrix of the function 'genAlgo' 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 may also be placed outside of their hexagon.
max_dist	A numeric value multiplied by the rotor radius to perform collision checks. Default is 2.2

Value

Returns a list.

Author(s)

Sebastian Gatscha

Examples

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

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

RandomSearchPlot *RandomSearchPlot* Plot the result of a randomized output.

Description

Plotting method for the functions [RandomSearchTurb](#) and [RandomSearch](#).

Usage

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

Arguments

resultRS	The result of the random functions RandomSearchTurb and RandomSearch .
result	The result of the function genAlgo or windfarmGA .
Polygon1	The Polygon for the wind farm area.
best	How many best candidates to plot.

Author(s)

Sebastian Gatscha

Examples

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

Res = RandomSearchTurb(result = resultrect, Polygon1 = polygon, n=10)
RandomSearchPlot(resultRS = Res, result = resultrect, Polygon1 = polygon, best=2)
```

RandomSearchTurb *RandomSearchTurb Randomise 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

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

Arguments

result	The resulting matrix of the function 'genAlgo' 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 input variable. The randomly generated points may also be placed outside of their hexagon.
max_dist	A numeric value multiplied by the rotor radius to perform collision checks. Default is 2.2

Value

Returns a list.

Author(s)

Sebastian Gatscha

Examples

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

Res = RandomSearchTurb(result = resultrect, Polygon1 = polygon, n = 10)
RandomSearchPlot(resultRS = Res, result = resultrect, Polygon1 = polygon, best = 2)
```

rasterprofile *rasterprofile*

Description

Sample a raster along a straight line between 2 points

Usage

```
rasterprofile(r, xy1, xy2, 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
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)

Author(s)

Sebastian Gatscha

readinteger *Check Input Crossover Method*

Description

Checks whether the input for [crossover1](#) 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)

Author(s)

Sebastian Gatscha

Examples

```
readinteger()
```

readintegerSel	<i>Check Input Selection Method</i>
----------------	-------------------------------------

Description

Checks whether the input for `selection1` 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)

Author(s)

Sebastian Gatscha

Examples

```
readintegerSel()
```

selection1	<i>Selection Method</i>
------------	-------------------------

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

```
selection1(fit, Grid, teil, elitism, nelit, selstate, verbose)
```

Arguments

fit	A list of all fitness-evaluated individuals
Grid	Is the indexed grid output from GridFilter
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.

Author(s)

Sebastian Gatscha

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)
plot(Polygon1,axes=TRUE)

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

startsel <- StartGA(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 <- selection1(fit, Grid, 2, T, 6, "VAR", NULL)
selec6best <- selection1(fit, Grid, 2, T, 6, "FIX", NULL)
selec6best <- selection1(fit, Grid, 4, F, 6, "FIX", NULL)
str(selec6best)

## 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 [crossover1](#).

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.

Author(s)

Sebastian Gatscha

Examples

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

StartGA

*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

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

Arguments

Grid	The data.frame output of "GridFilter" 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.

Author(s)

Sebastian Gatscha

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)
plot(Polygon1,axes=TRUE)

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

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

`tess2SPdf`*Create a Tessellation from a Polygon*

Description

Returns a Spatial Polygons object from a Tessellation object.

Usage

```
tess2SPdf(x)
```

Arguments

`x` The Tessellation object (Tessellation)

Value

Returns a SpatialPolygons. (SpatialPolygons)

Author(s)

Sebastian Gatscha

Examples

```
library(spatstat)
library(maptools)
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 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)
plot(Polygon1,axes=TRUE)
## Create a hexagonal Tessellation
HexaGrid <- spatstat::hextess(maptools::as.owin.SpatialPolygons(Polygon1),s = 100)
plot(HexaGrid)
HexaGrid
## Convert the Tessellation to SpatialPolygons
Hex2spdf <- tess2SPdf(HexaGrid)
plot(Hex2spdf)
Hex2spdf
```

 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

Author(s)

Sebastian Gatscha

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)
plot(Polygon1,axes=TRUE)

## 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 <- GridFilter(shape = Polygon1,resol = 200,prop = 1);
Grid <- Grid1[[1]]
AmountGrids <- nrow(Grid)

startsel <- StartGA(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 <- selection1(fit, Grid,2, T, 6, "VAR");
selec6best <- selection1(fit, Grid,2, T, 6, "FIX");
selec6best <- selection1(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 <- crossover1(selec6best, 2, uplimit = 300, crossPart="RAN");
crossOut <- crossover1(selec6best, 7, uplimit = 500, crossPart="RAN");
crossOut <- crossover1(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);
mut==crossOut

## 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(mut1)

```

Description

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

Usage

```
VekWinkelCalc(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

Author(s)

Sebastian Gatscha

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])))
wkl <- 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]] <- VekWinkelCalc(t = t, o = i, wkl = wkl,
    distanz = distanz, polyGon = polyGon, plotAngles = TRUE)
}
potInfTur
```

viewshed	<i>viewshed</i>
----------	-----------------

Description

Calculate visibility for given points in a given area.

Usage

```
viewshed(r, shape, turbine_locs, h1 = 0, h2 = 0, progress = "none")
```

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
progress	Is passed on to plyr::aapply

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

Author(s)

Sebastian Gatscha

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)

sample_POI <- spsample(DEM_meter[[2]], n = ncell(DEM_meter[[1]]), type = "regular")
sample_xy <- coordinates(sample_POI)

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	<i>viewTo</i>
--------	---------------

Description

Check if Point 1 (xy) is visible from multiple points (xy2)

Usage

```
viewTo(r, xy1, xy2, h1 = 0, h2 = 0, progress = "none")
```

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
h1	A numeric giving the extra height offset of Point 1
h2	A numeric giving the extra height offset of Point 2
progress	Is passed on to plyr::aapply

Value

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

Author(s)

Sebastian Gatscha

windata_format	<i>Transform Winddata</i>
----------------	---------------------------

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 pobabilities

Author(s)

Sebastian Gatscha

Examples

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

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

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

windfarmGA

Check user input and start an optimization

Description

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

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 matrix with all tested wind farm layouts.

Author(s)

Sebastian Gatscha

See Also

[genAlgo](#)

Index

BaroHoehe, 3

calculateEn, 4, 9
cansee, 6
crossover1, 7, 40, 43

dup_coords, 8

fitness, 9

genAlgo, 4, 9, 10, 19, 28–31, 33, 38, 52
getDEM, 14
getISO3, 15
getRects, 16
GridFilter, 17, 25, 32, 42

heatmapGA, 8, 19
HexaTex, 20, 25

InfluPoints, 21
interpol_view, 22
isSpatial, 24

leafPlot, 25

mutation, 26

permutations, 27
plot, 23
plot_farm_3d, 35
plot_viewshed, 36
plotbeorwor, 27
plotCloud, 28
plotEvolution, 29
plotfitnessevolution, 30
plotparkfitness, 30
plotResult, 31
PlotWindfarmGA, 33
plotWindrose, 34

quantile, 23

RandomSearch, 37, 38
RandomSearchPlot, 38
RandomSearchTurb, 38, 39
rasterprofile, 40
readinteger, 40
readintegerSel, 41

selection1, 7, 41, 41
splitAt, 43
StartGA, 44

tess2SPdf, 45
trimton, 46

VekWinkelCalc, 47
viewshed, 49
viewTo, 50

windata_format, 50
windfarmGA, 4, 18, 19, 28–31, 33, 38, 51