Package 'windfarmGA'

July 28, 2017

Title Genetic Algorithm for Wind Farm Layout Optimization

Version 1.1.1

Date 2017-07-25

Author Sebastian Gatscha

Maintainer Sebastian Gatscha <sebastian_gatscha@gmx.at>

Description The genetic algorithm is designed to optimize small wind farms consisting of up to 50 turbines. 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), raster (>= 2.5-2), sp (>= 1.2-2), dplyr (>= 0.5.0), data.table (>= 1.9.6), gtools (>= 3.5.0), maptools (>= 0.8-36), calibrate (>= 1.7.2), geoR (>= 1.7-5.1), gstat (>= 1.0-26), ggplot2 (>= 2.2.0), RColorBrewer (>= 1.1-2), rgeos (>= 0.3-11), RgoogleMaps (>= 1.2.0.7), googleVis (>= 0.5.9), rgl (>= 0.96.0), leaflet (>= 1.1.0), methods (>= 3.2.5), spatstat (>= 1.42-2), RandomFields (>= 3.1.50), grDevices, graphics, stats, utils

Suggests magrittr

LazyData TRUE

License MIT + file LICENSE

RoxygenNote 6.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2017-07-27 22:38:42 UTC

R topics documented:

`BaroHoehe` *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

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

Arguments

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

Value

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

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, windraster,
  wnkl, distanz, polygon1, resol, RotorR, dirSpeed, srtm_crop, topograp,
  cclRaster, weibull, weibullsrc)
```

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 (numeric)
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 (numeric)
windraster	Dummy windraster for the considered area with value 1 (raster)
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 (numeric)
polygon1	The considered area as shapefile (SpatialPolygons)
resol	The resolution of the grid in meter (numeric)
RotorR	The desired rotor radius in meter (numeric)
dirSpeed	The wind speed and direction data.frame (data.frame)
srtm_crop	An SRTM raster for the considered area. It is only used when the terrain effect model is activated (raster)
topograp	Logical value that indicates whether the terrain effect model is activated (TRUE) or deactivated (FALSE) (logical)
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 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. (logical)
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. (list)

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=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)
plot(Polygon1,axes=TRUE)

## Initialize a dummy wind speed raster with value 1
windraster <-raster::rasterize(Polygon1, raster::raster(
                              raster::extent(Polygon1),
                              ncol=180, nrow=180),field=1)

## Create a uniform and unidirectional wind data.frame and plot the
## resulting wind rose
data.in <- as.data.frame(cbind(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,wnkl = 20,
                   distanz = 100000, resol = 200,dirSpeed = data.in,
                   RotorR = 50, polygon1 = Polygon1, topograp = FALSE,
                   windraster = windraster)
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 <- as.data.frame(cbind(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,
                  windraster = windraster)
length(resCalcEn)
str(resCalcEn)

## Take Weibull Paramter Raster from the package. (Only for Austria)
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,wnkl = 20,
                      distanz = 100000, resol = 200,dirSpeed = data.in,
                      RotorR = 50, polygon1 = Polygon1, topograp = FALSE,
                      windraster = windraster, weibull = TRUE)
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)

## Use your own rasters for the Weibull parameters.
araster <- "/..pathto../a_param_raster.tif"
kraster <- "/..pathto../k_param_raster.tif"
weibullrasters <- list(raster(kraster), raster(araster))
plot(Polygon1); points(resStartGA[[1]]$X,resStartGA[[1]]$Y, pch=20,cex=2)
plot(resGrid[[2]],add=TRUE)
resCalcEn1 <-calculateEn(sel=resStartGA[[1]], referenceHeight= 50,
                       RotorHeight= 50, SurfaceRoughness = 0.14,wnkl = 20,
                       distanz = 100000, resol = 200,dirSpeed = data.in,
                       RotorR = 50, polygon1 = Polygon1, topograp = FALSE,
                       windraster = windraster, weibull = TRUE,
                       weibullsrc = weibullrasters)
length(resCalcEn1)
str(resCalcEn1)
resCalcEn1 <- as.data.frame(resCalcEn1)
plot(Polygon1, main = resCalcEn1$Energy_Output_Red[[1]])
points(x = resCalcEn1$Bx, y = resCalcEn1$By, pch = 20)

## End(Not run)
```

crossover1 *Crossover Method*

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)
```

Arguments

se6	The selected individuals. The output of selection1 (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)

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(cbind(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(cbind(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")
```

`euc.dist` *Euclidian Distance between two Points*

Description

Calculates the euclidian distance between two points.

Usage

```
euc.dist(x, y)
```

Arguments

x	A numeric value with X and Y coordinates for Point 1 (numeric)
y	A numeric value with X and Y coordinates for Point 2 (numeric)

Value

Returns a numeric value indicating the euclidian distance between two Points. (numeric)

Author(s)

Sebastian Gatscha

Examples

```
x=c(200,100)
y=c(1000,2000)
euc.dist(x,y)
```

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, weibullsrc)
```

Arguments

selection	A list containing all individuals of the current population. (list)
referenceHeight	The height at which the incoming wind speeds were measured. (numeric)
RotorHeight	The desired height of the turbine. (numeric)
SurfaceRoughness	A surface roughness length of the considered area in m. (numeric)
Polygon	The considered area as shapefile. (SpatialPolygons)
resol1	The resolution of the grid in meter. (numeric)
rot	The desired rotor radius in meter. (numeric)
dirspeed	The wind speed and direction data.frame. (data.frame)
srtm_crop	Logical value that indicates whether the terrain effect model is activated (TRUE) or deactivated (FALSE). (logical)
topograp	Logical value that indicates whether the terrain effect model is activated (TRUE) or deactivated (FALSE). (logical)
cclRaster	A Corine Land Cover raster, that has to be adapted previously by hand with the surface roughness lenght for every land cover type. Is only used, when the terrain effect model is activated. (raster)
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. (logical)
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. (list)

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. (list)

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=0,0,0,0,0,0,0 +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))
# windrosePlot <- plotWindrose(data = data.in, spd = data.in$ws,
#                dir = data.in$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)

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

## Calculate fitness values with the Weibull parameters included in the
## package. Only available for Austria.
fit <- fitness(selection = startsel,referenceHeight = 100, RotorHeight=100,
              SurfaceRoughness=0.3,Polygon = Polygon1, resol1 = 200,rot=20,
              dirspeed = wind, srtm_crop="",topograp=FALSE,cclRaster="",
              weibull=T)
head(fit)

## Calculate fitness values with given Weibull rasters.
library(raster)
```

```
araster <-  "...path.to.../scale_param_weibull.tif"
kraster <- "...path.to.../shape_param_weibull.tif"
weibullrasters <- list(raster(kraster), raster(araster))

fit <- fitness(selection = startsel,referenceHeight = 100, RotorHeight=100,
               SurfaceRoughness=0.3,Polygon = Polygon1, resol1 = 200,rot=20,
               dirspeed = wind, srtm_crop="",topograp=FALSE,cclRaster="",
               weibull=T, weibullsrc = weibullrasters)
head(fit)
```

---

genAlgo *Start The Genetic Algorithm for a wind Farm Layout*

Description

This function coordinates all other elements of the genetic algorithm. To initiate an optimization run, this method has to be called with the desired inputs. To be able to include the terrain effect model, the source of the Corine Land cover raster has to be given. This function will not control user inputs before an optimization process. It is therefore recommended to start an optimization run with the windfarmGA function.

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)
```

Arguments

Polygon1	The considered area as shapefile (SpatialPolygons)
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. (character)
Rotor	A numeric value that gives the rotor radius in meter (numeric)
n	A numeric value indicating the required amount of turbines (numeric)
fcrR	A numeric value that is used for grid spacing (numeric)
referenceHeight	The height at which the incoming wind speeds were measured. (numeric)
RotorHeight	The desired height of the turbine. (numeric)
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. (numeric)

`Proportionality`
: A numeric factor used for grid calculation. Determines the percentage a grid has to overlay (numeric)

`iteration`
: A numeric value indicating the desired amount of iterations of the algorithm (numeric)

`mutr`
: A numeric mutation rate with low default value of 0.008 (numeric)

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

`topograp`
: Logical value, which indicates if the terrain effect model should be activated or not. (logical)

`elitism`
: Boolean value, which indicates whether elitism should be included or not. (logical)

`nelit`
: If `elitism` is TRUE, then this input variable determines the amount of individuals in the elite group. (numeric)

`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. (character)

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

`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. (logical)

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

`sourceCCL`
: The source to the Corine Land Cover raster (.tif). Only required when the terrain effect model is activated. (character)

`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. See Details. (character)

`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. (logical)

`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. (list)

Value

The result of this run is a matrix of all relevant output parameters. This output can be used for several plotting functions. (matrix)

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=0,0,0,0,0,0,0 +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))
# windrosePlot <- plotWindrose(data = data.in, spd = data.in$ws,
#                dir = data.in$wd, dirres=10, spdmax=20)

## Runs an optimization run for 10 iterations (iteration) with the
## given shapefile (Polygon1), the wind data.frame (data.in),
## 12 turbines (n) with rotor radii of 30m (Rotor) and a grid spacing
## factor of 5 (fcrR) and other required inputs.
result <- genAlgo(Polygon1 = Polygon1, 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, Polygon1 = Polygon1)

## Runs the same optimization run, only 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)
```

`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. (matrix)
`Grid`	Grid of the considered area (data.frame)

Value

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

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)

## 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))
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");
selec6best <- selection1(fit, Grid,2, TRUE, 6, "FIX");
selec6best <- selection1(fit, Grid,4, FALSE, 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)

## 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)
```

GoogleChromePlot *Plot the Best Results in Google Chrome*

Description

Plot the best energy or efficiency solutions in Google Chrome with a satellite background image of Google maps. Input Polygon is not plotted

Usage

```
GoogleChromePlot(result, Polygon1, best = 1, plotEn = 1, Projection)
```

Arguments

result	The output matrix of windfarmGA or genAlgo, which has stored all relevant information. (matrix)
Polygon1	The considered area as shapefile (SpatialPolygons)
best	A numeric value indicating the best individual to be plotted. 1 will indicate the solution with highest value. (numeric)
plotEn	A numeric value that indicates if the best energy or efficiency output should be plotted. (plotEn==1) will plot the best energy solution and (plotEn==2) will plot the best efficiency solution. (numeric)
Projection	A desired Projection can be used instead of the default Lambert Azimuthal Equal Area Projection. (character)

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/polygon.rda", package = "windfarmGA"))

## Plot the results of a wind farm optimization
result <- resultrect
Polygon1 <- polygon
GoogleChromePlot(result, Polygon1, 1, 1)
```

GooglePlot *Plot the* best *Results with Google background map*

Description

Plot the best energy or efficiency solutions with a background image of Google maps.

Usage

```
GooglePlot(result, Polygon1, best = 1, plotEn = 1, Projection, ...)
```

Arguments

result	The output matrix of windfarmGA or genAlgo, which has stored all relevant information. (matrix)
Polygon1	The considered area as shapefile (SpatialPolygons)
best	A numeric value indicating which best individuals to be plotted. 1 will indicate the solution with highest value. (numeric)
plotEn	A numeric value that indicates if the best energy or efficiency output should be plotted. (plotEn==1) will plot the best energy solution and (plotEn==2) will plot the best efficiency solution. (numeric)
Projection	A desired Projection can be used instead of the default Lambert Azimuthal Equal Area Projection. (character)
...	Some arguments can be passed to GetMap and PlotOnStaticMap, of the 'RgoogleMaps' package including maptype, col, cex, pch, size and zoom. If the plot is not required, it can be disabled with the option plT = FALSE. Check the examples for some action.

Value

Returns a data.frame with the coordinates in LON/LAT and plots the desired best result with a Google background map. (data.frame)

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 wind farm optimization
result <- resultrect
```

```
Polygon1 <- polygon

GooglePlot(result, Polygon1, 1, 1)
GooglePlot(result, Polygon1, 2, 1, zoom=14, maptype="hybrid",
           col="darkblue", pch=18)
GooglePlot(result, Polygon1, 3, 1, zoom=14, maptype="terrain",
           col="black", pch=20)
GooglePlot(result, Polygon1, 3, 2, zoom=15, maptype="satellite",
           col="red", pch=10)
GooglePlot(result, Polygon1, 1, 1, zoom=14, maptype="mobile",
           col="darkblue", pch=17)
GooglePlot(result, Polygon1, 1, 1, zoom=14, maptype="hybrid",
           col="darkblue", pch=20)
GooglePlot(result, Polygon1, 1, 1, zoom=14, maptype="hybrid",
           col="blue", pch=18, cex= 2)
GooglePlot(result, Polygon1, 1, 1, zoom=14, maptype="hybrid",
           col="blue", pch=18, cex= 2, size=c(320,320))
```

`GridFilter` *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

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

Arguments

`shape`	Shape file of the considered area (SpatialPolygons)
`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. (numeric)
`plotGrid`	Logical value indicating whether resulting grid should be plotted or not. Default is FALSE. (logical)

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. (list)

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

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)

## Create a Grid
GridFilter(Polygon1,200,1,TRUE)
GridFilter(Polygon1,200,0.5,TRUE)
GridFilter(Polygon1,400,1,TRUE)
GridFilter(Polygon1,400,0.5,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)
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 windfarmGA or genAlgo, which has stored all relevant information (matrix)
si	A numeric value that is used for the sizing of the resolution of the heatmap. Default is 2 (numeric)
idistw	The inverse distance weighting power. Default is the rotor radius from the 'result' values (numeric)

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 data frame with coordinates and a SpatialPolygons object, that consists of hexagonal grid cells.

Usage

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

Arguments

Polygon1	The SpatialPolygons object (SpatialPolygons)
size	The side length of an hexagon (numeric)
plotTrue	Should the object be plotted (logical)

Value

Returns a list with an indexed data frame of the point coordinates and a SpatialPolygons object of the hexagons (list)

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

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. (list)

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)

t <- as.matrix(cbind(x=runif(10,0,raster::extent(polYgon)[2]),
     y=runif(10,0,raster::extent(polYgon)[4])))
```

```
wnkl=20
dist=100000
dirct=0

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

---

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)
```

Arguments

result	The resulting matrix of the function 'genAlgo' or 'windfarmGA'. (matrix)
Polygon1	The Polygon for the wind farm area. (SpatialPolygons)
which	A numeric value, indicating which best indidvual to plot. The default is 1 (the best resulting wind farm). (numeric)
orderitems	A logical value indicating whether the results should be ordered by energy values (TRUE) or chronologically (FALSE). (logical)

Value

Returns a leaflet map. (leaflet)

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)
```

mutation *Mutation Method*

Description

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

Usage

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

Arguments

a	The binary matrix of all individuals. (matrix)
p	The mutation rate. (numeric)

Value

Returns a binary matrix with mutated genes. (matrix)

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
```

plotbeorwor *Plot if previous population was better or worse*

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. (matrix)

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. (matrix)
pl	Should the results be plotted? Default is FALSE (logical)

Value

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

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 windfarmGA or genAlgo, which has stored all relevant information. (matrix)
ask	Should R wait for interaction for subsequent plotting. Default is TRUE (logical)
spar	A numeric value determining how exact a spline should be drawn. Default is 0.1 (numeric)

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)
```

`plotfitnessevolution` *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

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

Arguments

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

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
plotfitnessevolution(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 windfarmGA or genAlgo, which has stored all relevant information. (matrix)
spar	A numeric value determining how exact a spline should be drawn. Default is 0.1 (numeric)

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 windfarmGA or genAlgo, which has stored all relevant information. (matrix)
Polygon1	The considered area as shapefile. (SpatialPolygons)
best	A numeric value indicating how many of the best individuals should be plotted. (numeric)
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. (numeric)
topographie	A logical value, indicating whether terrain effects should be considered and plotted or not. (logical)
Grid	The grid as SpatialPolygons, which is obtained from GridFilter and used for plotting.
Projection	A desired Projection can be used instead of the default Lambert Azimuthal Equal Area Projection. (character)
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
sourceCCL	The source to the Corine Land Cover raster (.tif). Only required, when the terrain effect model is activated. (character)
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. (list)

Value

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

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
load(file = system.file("extdata/a_weibull.rda", package = "windfarmGA"))
load(file = system.file("extdata/k_weibull.rda", 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. 8 plot methods are available that can be selected individually.

Usage

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

Arguments

`result`	An output matrix of the function `windfarmGA` or `genAlgo`, which has stored all relevant information. (matrix)
`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". (character)
`Polygon1`	The considered area as shapefile. Only required if the shapefile is already loaded.(SpatialPolygons)
`whichPl`	Which plots should be shown: 1-8 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. (numeric)
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. (numeric)
Projection	A desired Projection can be used instead of the default Lambert Azimuthal Equal Area Projection. (character)
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. (list)

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.
load(file = system.file("extdata/a_weibull.rda", package = "windfarmGA"))
load(file = system.file("extdata/k_weibull.rda", 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", debug = 0, spdseq = NULL)
```

Arguments

data	A data.frame containing the wind information (data.frame)
spd	The column of the wind speeds in the "data"-data.frame (numeric)
dir	The column of the wind directions in the "data"-data.frame (numeric)
spdres	The increment of the wind speed legend. Default is 2 (numeric)
dirres	The size of the wind sectors. Default is 10 (numeric)
spdmin	Minimum wind speed. Default is 1 (numeric)
spdmax	Maximal wind speed. Default is 30 (numeric)
palette	A color palette used for drawing the wind rose (character)
debug	For running a debug. Default is 0 (numeric)
spdseq	A wind speed sequence, that is used for plotting (numeric)

Author(s)

Sebastian Gatscha

Examples

```
## Exemplary Input Wind speed and direction data frame
# Uniform wind speed and single wind direction
data.in <- as.data.frame(cbind(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 <- as.data.frame(cbind(ws=sample(1:25,10),wd=sample(1:260,10)))
windrosePlot <- plotWindrose(data = data.in, spd = data.in$ws,
   dir = data.in$wd)
```

PointToLine2 *Distances between right triangle points*

Description

Takes two input coordinates from the current turbine and from the potentially influencing turbine, which stands in front of the current turbine. The algorithm draws an imaginary right triangle between the two input points and a point C, which is calculated by the algorithm itself. Wind will always seem to come from north as the input area will be rotated accordingly if the incoming wind direction does not come from north. For further calculations, only the distance perpendicular to the wind direction and the sidewise distance to the potentially influencing turbine is required.

Usage

```
PointToLine2(x, y, plotAngles)
```

Arguments

x	Coordinates of the current turbine. (numeric)
y	Coordinates of the turbine that could potentially influence the current turbine. (numeric)
plotAngles	A logical variable, which is used to plot the distances and angles. Default is FALSE. (logical)

Details

Assume two points located at P1=c(x=1,y=1) and P2=c(x=10,y=15) and wind coming from north. If P1 is the current turbine and P2 the potentially influencing turbine, then C will be located at C=c(x=10,y=1) where it will build a right triangle. plot(rbind(P1,P2,C,P1),type="l") See Examples.

Value

Returns a matrix with the resulting distances and the coordinates of 3 points. (matrix)

Author(s)

Sebastian Gatscha

Examples

```
## For further calculations only the distances between B-C and A-C
## and the angle at A will be needed. B represents the current turbine
## and A represents a turbine, that could potentially influence turbine B.
x <- c(100,100); y <- c(500,500);
plot(rbind(x,y),col=c("red","blue"),cex=2,pch=20);
PointToLine2(x,y,TRUE)
```

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 *Check Input Selection Method*

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` *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

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

Arguments

`fit`	A list of all fitness-evaluated individuals (list)
`Grid`	Is the indexed grid output from `GridFilter` (data.frame)
`teil`	A numeric value that determines the selection percentage (numeric)
`elitism`	Boolean value which indicates whether elitism should be included or not. (logical)
`nelit`	If `elitism` is TRUE, then this input variable determines the amount of individuals in the elite group. (numeric)
`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. (character)

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. (list)

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))
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");
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. (data.frame)
n	A numeric value indicating the amount of required turbines. (numeric)
nStart	A numeric indicating the amount of randomly generated initial individuals. Default is 100. (numeric)

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. (list)

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 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)

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=0,0,0,0,0,0,0 +units=m +no_defs"
proj4string(Polygon1) <- CRS(Projection)
plot(Polygon1,axes=TRUE)
## Create a hexagonal Tesselation
HexaGrid <- spatstat::hextess(maptools::as.owin.SpatialPolygons(Polygon1),s = 100)
plot(HexaGrid)
HexaGrid
## Convert the Tesselation 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)
```

Arguments

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

Value

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

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))
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)
```

VekWinkelCalc *Calculate distances and angles of possibly influencing turbines*

Description

Calculate the relevant distances PointToLine2 and angles WinkelCalc for a given turbine location and all potentially influencing turbines.

Usage

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

Arguments

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

Value

Returns a data.frame with the distances and angles of potentially influencing turbines (data.frame)

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

`windfarmGA` *Controls the given inputs and initiates an Optimization run*

Description

The initiating function of an optimization process which will interactively check user-inputs first. If all inputs are correct, an optimization run will be started. This process can take a long time depending on the size of the problem and the number of desired iterations.

Arguments

`dns`	The source to the desired Shapefile. Only required if the shapefile is not already loaded. (character)
`layer`	The name of the desired Shapefile. Only required if the shapefile is not already loaded. (character)
`Polygon1`	The considered area as shapefile. Only required if the shapefile is already loaded.(SpatialPolygons)
`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. (character)
`sourceCCL`	The source to the Corine Land Cover raster (.tif). Only required, when the terrain effect model is activated. (character)

`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. See Details. (character)

`vdirspe`
: A data.frame containing the incoming wind speeds, wind directions and probabilities. To plot a wind rose from this data frame, see: `plotWindrose`. (data.frame)

`n`
: A numeric value indicating the required amount of turbines. (numeric)

`Rotor`
: A numeric value that gives the rotor radius in meter. (numeric)

`fcrR`
: A numeric value that is used for grid spacing. The resolution of a grid cell will be the rotor radius `Rotor` multiplied with this value. (numeric)

`iteration`
: A numeric value indicating the desired amount of iterations of the algorithm. (numeric)

`topograp`
: Logical value that indicates whether the terrain effect model is activated (TRUE) or deactivated (FALSE). (logical)

`referenceHeight`
: The height at which the incoming wind speeds were measured. Default is 50m (numeric)

`RotorHeight`
: The desired height of the turbine. Default is 100m (numeric)

`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. (numeric)

`Proportionality`
: A numeric factor used for grid calculation. Determines the percentage a grid has to overlay. See also: `GridFilter` (numeric)

`mutr`
: A numeric mutation rate with low default value of 0.008 (numeric)

`elitism`
: Boolean value which indicates whether elitism should be included or not. (logical)

`nelit`
: If `elitism` is TRUE, then this input variable determines the amount of individuals in the elite group. (numeric)

`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. (character)

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

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

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

`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. (logical)

`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. (list)

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. Another raster (.tif) is **required** which has to be downloaded previously at the following page: http://www.eea.europa.eu/data-and-maps/data/clc-2006-raster-1. Download the .zip package with **100 meter** resolution. Unzip the downloaded package and assign the source of the Raster Image '"g100_06.tif"' in the package to the input variable `sourceCCL`. The algorithm will use an adapted version of the Raster legend ("clc_legend.csv"), which is stored in the package subdirectory '~/extdata'. To use own values for the land cover roughness lengths, insert a column named **"Rauhigkeit_z"** to the .csv file in the Corine Land Cover package, assign a surface roughness length to all land cover types. Be sure that all rows are filled with numeric values and save the .csv file then with **";"** separation. Assign the source of the resulting .csv file to the input variable `sourceCCLRoughness` of this function. For further information, see the examples.

Value

Assigns the needed input values and starts an optimization run. The result of this run is a matrix of all relevant output parameters. This output is used for several plotting functions.

Author(s)

Sebastian Gatscha

Examples

```
################ REQUIRED INPUT POLYGON AND CCL SOURCE
library(rgdal);library(sp);
Polygon1 <- Polygon(rbind(c(4498482, 2668272), c(4498482, 2669343),
                          c(4499991, 2669343), c(4499991, 2668272)))
Polygon1 <- sp::Polygons(list(Polygon1),1);
Polygon1 <- sp::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)
## Or Polygon is given by source:
dsn="Source of the Polygon File (SHP)"
layer="Name of Polygon"
```

```
Polygon1 <- rgdal::readOGR(dsn=dsn,layer=layer)
plot(Polygon1)


## The following two sources are required, if terrain effects should
## be considered
sourceCCL <- "Source of the CCL raster (TIF)"
sourceCCLRoughness <- "Source of the Adaped CCL legend (CSV)"

################ REQUIRED INPUT WIND SPEED DATA FRAME
## Exemplary Input Wind speed and direction data frame
## Uniform wind speed and single wind direction
vdirspe <- structure(list(ws = c(12,12), wd = c(0,0), probab = c(25,25)),
                    .Names = c("ws", "wd","probab"),
                    row.names = c(NA, 2L),class ="data.frame")
windrosePlot <- plotWindrose(data = vdirspe, spd = vdirspe$ws,
                             dir = vdirspe$wd, dirres = 10, spdmax = 20)
## First check if the grid size is convenient
Rotor <- 50;
fcrR <- 3.5
## with rectangular Grid Cells
GridFilter(shape = Polygon1,resol = (Rotor*fcrR), prop = 1, plotGrid = TRUE)
## or with hexagonal Grid Cells
HexaTex(Polygon1 = Polygon1, size = ((Rotor*fcrR)/2), plotTrue = TRUE)

################ STARTING AN OPTIMIZATION RUN
result <- windfarmGA(Polygon1 = Polygon1, n=12, Rotor=Rotor,fcrR=fcrR,
             vdirspe = vdirspe, crossPart1 = "EQU",selstate="FIX",mutr=0.8,
             Proportionality = 1, SurfaceRoughness = 0.3, topograp = FALSE,
             elitism=TRUE, nelit = 7, trimForce = TRUE,iteration=10,
             referenceHeight = 50,RotorHeight = 100)
PlotWindfarmGA(result = result, Polygon1 = Polygon1)

## Start the same optimization run, only with hexagonal grid cells.
result <- windfarmGA(Polygon1 = Polygon1, GridMethod="h", n=12, Rotor=Rotor,
             fcrR=fcrR,iteration=10, vdirspe = vdirspe, 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, GridMethod = "h", Polygon1 = Polygon1)

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

## Use the resulting matrix with the different plotting methods of
## this package, to explore the behaviour of the genetic algorithm.
```

`WinkelCalc` *Calculates Angles between 3 Points*

Description

Calculates all three angles for an imaginary right triangle between the actual turbine, the possible influencing turbine and a right angle. The function works as well with non-right triangles, although it is not needed for the genetic algorithm, as only the distance perpendicular to the wind direction and the sidewise distance to the potentially influencing turbine is required for further calculations. Point C, where the right angle is located will therefore be calculated by the algorithm itself. See also `PointToLine2`.

Usage

```
WinkelCalc(Aa, Bb, Cc)
```

Arguments

`Aa`	A numeric value with the x and y coordinates of a potentially influencing turbine (numeric)
`Bb`	A numeric value with the x and y coordinates of the current turbine (numeric)
`Cc`	A numeric value with the x and y coordinates of the imaginary right angle (numeric)

Value

Returns a matrix with the alpha, betha and gamma angles of the imaginary right triangle (matrix)

Author(s)

Sebastian Gatscha

Examples

```
Aa= as.numeric(cbind(1,1))
Bb= as.numeric(cbind(10,3))
Cc= as.numeric(cbind(10,1))
plot(rbind(Aa,Bb,Cc,Aa), type="b", xlab="x",
     ylab="y", ylim=c(0,4), xlim=c(0,11));
points(x=Aa[1],y=Aa[2],col="green",pch=20);
points(x=Bb[1],y=Bb[2],col="red",pch=20);
points(x=Cc[1],y=Cc[2],col="blue",pch=20)
Angles <- WinkelCalc(Aa,Bb,Cc); Angles
text(rbind(Aa,Bb,Cc),labels=round(Angles,2),pos=1)
```

Index