# Package ‘wflo’

April 22, 2020

**Title** Data Set and Helper Functions for Wind Farm Layout Optimization Problems

**Version** 1.5

**Date** 2020-04-12

**Description** Provides a convenient data set, a set of helper functions, and a benchmark function for economically (profit) driven wind farm layout optimization. This enables researchers in the field of the NP-hard (non-deterministic polynomial-time hard) problem of wind farm layout optimization to focus on their optimization methodology contribution and also provides a realistic benchmark setting for comparability among contributions.

**Depends** R (>= 3.5.0)

**Suggests** MASS, nloptr, pso, rgenoud

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**NeedsCompilation** no

**Author** Carsten Croonenbroeck [aut, cre], David Hennecke [ctb]

**Maintainer** Carsten Croonenbroeck `<carsten.croonenbroeck@uni-rostock.de>`

**Repository** CRAN

**Date/Publication** 2020-04-22 10:00:03 UTC

## R topics documented:

## `AcquireData` *Downloads the larger data set for entire Germany.*

### Description

Convenience function that downloads the larger data set covering the entire area of Germany and saves it to the specified directory. This data set can replace the smaller sub sample data set that is part of this package.

### Usage

```
AcquireData(Folder)
```

### Arguments

| | |
|---|---|
| `Folder` | must be a character string containing the folder location to where the file will be saved. For instance, use `getwd` to save to the current working directory. |

### Details

This function will interactively lead the user through the downloading process and also gives advice on how to use the larger data set. Make sure that R can write to the specified directory.

### Value

`AcquireData` returns nothing.

### Author(s)

Carsten Croonenbroeck

## See Also

Use `FarmVars` for the settings object and `FarmData` for the smaller, built-in data set that is part of this package.

## Examples

```
## Not run:
AcquireData(tempdir())
# Will download the data file to the specified directory.

## End(Not run)
```

---

`Cost` *Stub for a turbine's cost function.*

---

## Description

A function that returns the yearly installation costs for a given set of turbines (provide `x` and `y` for the turbines' locations). In its present form it only returns the 'UnitCost' value from the `FarmVars` settings object per turbine.

## Usage

```
Cost(x, y)
```

## Arguments

| | |
|---|---|
| `x` | can be a single value or a numeric vector of values, contains the 'x' location(s) of turbines. |
| `y` | can be a single value or a numeric vector of values, contains the 'y' location(s) of turbines. |

## Details

Note that `x` and `y` should both be of length `n`, i.e. the numbers of values they contain should match the number of turbines in the current wind farm layout problem.
This function is a stub and can and should be replaced by something reasonable in an actual wind farm layout problem.

## Value

`Cost` returns a vector of values. The number of elements matches the length of `x` and `y`.

## Author(s)

Carsten Croonenbroeck

## See Also

Profit to see where to use Cost, Yield for a similar function for yearly yield.

## Examples

```
## Returns a vector of two, c(100000, 100000).
Cost(c(0.5, 0.7), c(0.2, 0.3))

## Replace the function by another function
## also called 'Cost', embedded in environment e.
## Also, see the vignette.

if(requireNamespace("pso")){
library(pso)
e$Cost <- function(x, y) #x, y \in R^n
{
retVal <- rep(e$FarmVars$UnitCost, min(length(x), length(y)))
retVal[x > 0.5] <- retVal[x > 0.5] * 2
return(retVal)
}
set.seed(1357)
Result <- psoptim(par = runif(NumTurbines * 2), fn = Profit,
  lower = rep(0, NumTurbines * 2), upper = rep(1, NumTurbines * 2))
Result
rm(Cost, envir = e)
}
```

---

e *Environment for data and variables.*

---

## Description

Environment that contains variables object FarmVars and, if downloaded, full data set FarmData.

## Usage

```
e
```

## Details

e contains data and variables, see FarmVars and FarmData.

## Value

e returns the environment for FarmVars and FarmData.

## Author(s)

Carsten Croonenbroeck

## See Also

`AcquireData` for downloading the full data set.

## Examples

```
e
## Gives the environment object.
```

---

`FarmData` *Data set for wind farm layout optimization.*

---

## Description

This list object contains four matrices covering adjusted yield, wind speed, wind direction and standard deviations of wind directions in Germany.

## Usage

```
FarmData
```

## Format

A `list` object containing seven matrices, each containing 25 x 25 values at a raster resolution of 200 x 200 m (note that for the larger data set downloadable using `AcquireData`, each matrix contains 4400 x 3250 values):

`$AdjustedYield`: Yield is average annual energy production (AEP). According to FGW technical guidelines, AEP is adjusted due to different location qualities to obtain a better guess at the marketable energy output. Interpret these values directly as 'megawatt hours per year'.

`$WindSpeed`: Average wind speeds in meters per second.

`$WindDirection`: Average wind directions in degrees (azimuth system).

`$SDDirection`: Standard deviations of wind directions in degrees (azimuth system).

`$Elevation`: Terrain relief (elevation in meters).

`$Slope`: Terrain slope in degrees.

`$SlopeDirection`: Direction of hillside in degrees.

## Note

If the full mode data set is present, it is loaded into the environment `e`. The built-in data set `FarmData` contains matrices of dimension 25 x 25, while the full data set `e$FarmData` consists of 4400 x 3250 matrices.

## Author(s)

Carsten Croonenbroeck
David Hennecke

## Source

DWD Climate Data Center (CDC): ftp://opendata.dwd.de/climate_environment/CDC/grids_germany/multi_annual/wind_parameters/resol_200x200/
FGW Technical Guidelines: https://wind-fgw.de/shop/technical-guidelines/?lang=en

## Examples

```
# 'Profit' uses this data set internally:
NumTurbines <- 4
set.seed(1357)
Result <- optim(par = runif(NumTurbines * 2), fn = Profit, method = "L-BFGS-B",
  lower = rep(0, NumTurbines * 2), upper = rep(1, NumTurbines * 2))
Result
PlotResult(Result)
```

---

`FarmVars` *Variables object for wind farm specifications.*

---

## Description

This list object collects all necessary variables for wind farm layout optimizations within this package. The object is loaded upon loading the package, embedded into a user writeable environment e and provides a set of default values.

## Usage

```
FarmVars
```

## Format

A `list` object containing:

`$UnitCost`: This contains yearly costs per turbine in actual currency. Mostly, this will be installation costs, maintenance cost and others. If, for example, total turbine installation costs are 2 mio. EUR and the projected life span of the turbine is 20 years, then the yearly cost is roughly 100,000 EUR, which is also the default value for this item. Refine the number at will to encompass annuity (taking interest effects into consideration), salvage, or the mentioned maintenance cost.

`$Price`: This denotes the average sale price of electricity produced at the wind farm per megawatt hour. Defaults to 100, since 100 EUR per megawatt hour is a good (still rough) estimate in the European electricity market.

`$StartPoint`: Denotes the point at which to start to 'cut out' a sample field from the data set.

Since the full data set covers the entire region of Germany at a 200 x 200 meters resolution, the entire area is too big for a wind farm. For an actual wind farm layout optimization problem, a smaller region is indicated. This package will conveniently provide a square area of 5 km x 5 km in size. Note: Concerning the larger data set covering the entire area of Germany (conveniently downloadable using `AcquireData`), this area con be found at point 2000:2000 (full data covering a range of 4400 x 3250 points) and from there, spans a five kilometers square, i.e. covers the raster at 2000:2024, 2000:2024. Change the default value of 1 to 2000 to find that area or to something else for cross checking your algorithm with data covering a different area. However, for comparability, evaluate your algorithm result at the default of 2000 or using the built-in data set.

`$Width`: Denotes the width (and, since the test area is a square, also the height) of the test area. Defaults to 25, which leads to a 5 km x 5 km square as a sample wind field, as the raster resolution is 200 x 200 m.

`$MeterMinDist`: Denotes the minimum distance from one turbine to another in meters. Defaults to 500 meters, which is a rather conservative value.

`$EndPoint`: The complement to '$StartPoint'. Is computed based on '$StartPoint' and '$Width' as follows: FarmVars$StartPoint + FarmVars$Width - 1. By default, $StartPoint = 1 and $Width = 25, so $EndPoint = 1 + 25 - 1 = 25.

`$MeterWidth`: Contains the width (and, since the test area is a square, also the height) of the test area in meters. Since by default, FarmVars$Width = 25 and the raster resolution is 200 x 200 m, this defaults to $MeterWidth = 200 * 25 = 5000 meters (5 km).

`$MinDist`: Contains the minimum distance in problem space. Since the problem space in a convenient unit square, this defaults to $MinDist = MeterMinDist / MeterWidth = 500 / 5000 = 0.1.

`$z`: Contains the hub height of the turbine type under investigation. Default value is 100 meters.

`$z0`: Contains the terrain's roughness length required for Jensen's wake model. In most studies, a default value of 0.1 is agreed to be a good guess for onshore wind farms, so this is the default value here.

`$r0`: Contains the rotor radius for the turbine type under investigation. For instance, a Vestas V90 turbine has a rotor diameter of 90 meters (hence the name), so the radius is 45 mesters, which is also the default value here.

`$BenchmarkSolution`: Contains the best setup for the standard benchmark field at a task of placing 20 turbines. This result has been generated by optimizer `genoud()` from package `rgenoud`. It should serve as a benchmark or starting point for improved solutions.

## Source

Carsten Croonenbroeck

## Examples

```
# Inspect the benchmark solution by
```

```
Result <- list(par = e$FarmVars$BenchmarkSolution)
Profit(Result$par)
PlotResult(Result)
ShowWakePenalizers(Result)
```

---

`Geo2Ari` *Converts degrees between the arithmetic system and the azimuth system (and vice versa).*

---

## Description

Use this function to convert degrees from the arithmetic system (0° being east, ascending counter-clockwise) into the azimuth system (0° being north, ascending clockwise) and vice versa.

## Usage

```
Geo2Ari(g)
```

## Arguments

| | |
|---|---|
| `g` | contains a single value degree (usually between 0° and 360°, decimal fractions allowed). |

## Details

`g` may contain degrees from both systems, the function turns the data into the respective other system.

## Value

`Geo2Ari` returns a single value of degrees.

## Author(s)

Carsten Croonenbroeck

## See Also

`GetDirInfo` for further degree information.

## Examples

```
Geo2Ari(0)
## In an arithmetic system, 0° means 'east', while 'east' in
## azimuth notation is 90°. This call returns 90.
Geo2Ari(90)
## In an azimuth system, 90° means 'east', while 'east' in
## arithmetic notation is 0°. This call returns 0.
Geo2Ari(Geo2Ari(123))
## Returns 123.
```

## `GetAngle` *Returns the angle between two turbines.*

### Description

As seen from a point in the wind farm, computes the angle to another point in that farm.

### Usage

```
GetAngle(x1, y1, x2, y2)
```

### Arguments

| | |
|---|---|
| `x1` | must be a single value. Provide the `x` location of the first turbine. |
| `y1` | must be a single value. Provide the `y` location of the first turbine. |
| `x2` | must be a single value. Provide the `x` location of the second turbine. |
| `y2` | must be a single value. Provide the `y` location of the second turbine. |

### Details

From point 2's point of view, computes the angle to point 1.

### Value

`GetAngle` returns a single number between 0 and 360. If both points are identical, the return value is 0.

### Note

Note that this function returns an angle in arithmetic degrees notation. To convert to azimuth notation, use `Geo2Ari`.

### Author(s)

Carsten Croonenbroeck

### See Also

Use `JensenAngle` to compute the wake cone and with it, use `JensenTrapezoid` to see if another turbine B is in turbine A's wake.

### Examples

```
GetAngle(0.2, 0.2, 0.1, 0.1)
## Looking from point (0.1, 0.1) at point (0.2, 0.2), the angle is 45° (arithmetic).
```

GetArrow *Simple helper function for* PlotResult.

## Description

Given a point, an angle information (in arithmetic degrees), and a length information, computes start and end points of an arrow usable via the arrows function.

## Usage

```
GetArrow(BaseX, BaseY, Degrees, Frac = 25)
```

## Arguments

| | |
|---|---|
| BaseX | must be a single value containing the x value of a point which later will be the center of the arrow. |
| BaseY | must be a single value containing the y value of a point which later will be the center of the arrow. |
| Degrees | must be a single value containing the desired rotation degree of the arrow. |
| Frac | must be a single value containing the length of the arrow. Default is 25 and for convenience, this parameter should in most cases be identical to FarmVars$Width. |

## Details

This function will be used internally by PlotResult.

## Value

GetArrow returns a vector of four values representing x and y for the start point and x and y for the end point of an arrow (in that order).

## Author(s)

Carsten Croonenbroeck

## See Also

Use PlotResult to visualize the optimization result. See FarmVars for the data object.

## Examples

```
GetArrow(0.5, 0.5, 45)
#At c(0.5, 0.5), generates an arrow pointing in north-eastern direction.
```

## GetDirInfo — *Returns average wind direction and direction standard deviation for a turbine's location.*

### Description

For a turbine's location represented by x and y, looks up the wind direction from the matrix Dirs and the corresponding standard deviation from matrix SDs. Internally transforms coordinates of x and y from problem space (usually unit square) to the matrix space of Dirs and SDs, respectively.

### Usage

```
GetDirInfo(x, y, Dirs, SDs)
```

### Arguments

| | |
|---|---|
| x | must be a single value containing the 'x' location of a turbine in problem space. |
| y | must be a single value containing the 'y' location of a turbine in problem space. |
| Dirs | a matrix containing average yearly wind directions. Usually, the third element of the list object FarmData will be used as this matrix. |
| SDs | a matrix containing average yearly wind direction standard deviations. Usually, the fourth element of the list object FarmData will be used as this matrix. |

### Value

GetDirInfo returns a vector of two elements, the first being the average wind direction in degrees and the second being the corresponding standard deviation. Note that degrees are meant in the arithmetic degrees system (0° being east, ascending counterclockwise). To transform into an azimuth system (0° being north, ascending clockwise), use function Geo2Ari. Also note that wind directions are meant to denote 'where the wind is going to' rather than 'where the wind is coming from'.

### Author(s)

Carsten Croonenbroeck

### See Also

Profit to see where to use GetDirInfo, Yield for a similar function for adjusted yield. See FarmVars for the data object.

### Examples

```
Dirs <- FarmData[[3]][e$FarmVars$StartPoint:e$FarmVars$EndPoint,
e$FarmVars$StartPoint:e$FarmVars$EndPoint]
SDs <- FarmData[[4]][e$FarmVars$StartPoint:e$FarmVars$EndPoint,
e$FarmVars$StartPoint:e$FarmVars$EndPoint]
GetDirInfo(0.5, 0.7, Dirs, SDs)
## Returns wind direction and standard deviation for the given location
## and provided the matrices given.
```

## ImposeVectorField *Simple helper function for* PlotResult.

### Description

Draws a set of arrows over an existing plot. Will be used internally by PlotResult to visualize the vector field of wind directions over a plot of the wind farm.

### Usage

```
ImposeVectorField(xNum = 5, yNum = 5, ColMain = "black",
  ColBand = "white", Frac = 25, DoSDs = TRUE)
```

### Arguments

| | |
|---|---|
| xNum | must be a single value containing the desired number of arrows horizontally. |
| yNum | must be a single value containing the desired number of arrows vertically. |
| ColMain | must be a single value containing the desired color for the main wind direction arrow. |
| ColBand | must be a single value containing the desired color for the secondary arrows representing the standard deviation arrows. Only used if `DoSDs = TRUE`. |
| Frac | must be a single value containing the desired length information for the arrows. Will be passed to GetArrow internally. |
| DoSDs | must be `TRUE` or `FALSE`. If `TRUE`, will not only draw one arrow for the main wind direction per point, but also two additional arrows (using color `ColBand`) representing the main wind direction plus/minus half the standard deviation. |

### Details

This function will be used internally by PlotResult.
`ImposeVectorField` requires FarmData to be present and an existing plot to impose the vector field over.

### Value

`ImposeVectorField` returns nothing.

### Author(s)

Carsten Croonenbroeck

### See Also

Use PlotResult to visualize the optimization result.

## Examples

```
plot(c(0, 1), c(0, 1))
ImposeVectorField(ColBand = "red")
```

---

`JensenAngle` *For a given distance* `x`*, computes the wake cone generated by a turbine.*

---

## Description

In Jensen's wake model, a wake cone is generated by a turbine based on the turbine's hub height z, its rotor radius r_0, the terrain's roughness length z_0 and the distance x between the turbine under investigation and a certain second point. z, r_0 and z_0 are taken from the `FarmVars` settings object, while x is to be provided.

## Usage

```
JensenAngle(x)
```

## Arguments

| | |
|---|---|
| `x` | must be a single value. Provide distance in meters. |

## Details

If a second turbine (B) is in a first turbine (A)'s wake, turbine B's power output must be penalized due to turbulence. This can be performed via Jensen's wake model, but first, it should be checked whether B is actually in A's wake. This function computes the angle of the wake cone generated by turbine A. This can be used together with function `JensenTrapezoid` `PointInPolygon` and to see if B is in the wake of A.

## Value

`JensenAngle` returns the (onesided) angle of the wake cone generated by a turbine for distance `x`.

## Author(s)

Carsten Croonenbroeck

## References

Jensen, N. O. (1983). A note on wind generator interaction. Roskilde: Risø National Laboratory. Risø-M, No. 2411

## See Also

`JensenTrapezoid` to check whether there are wake effects present. `FarmVars` for the data object.

## Examples

```
JensenAngle(500)
## For a distance of 500 m, the function returns a wake cone angle of 9.2233 degrees
## (given standard values for z, z_0 and r_0 in the FarmVars settings object).
```

---

## JensenFactor — *For a given distance x, computes the penalty factor for a turbine's wake.*

---

### Description

In Jensen's wake model, a wake cone is generated by a turbine based on the turbine's hub height z, its rotor radius r_0, the terrain's roughness length z_0 and the distance x between the turbine under investigation and a certain second point. z, r_0 and z_0 are taken from the FarmVars settings object, while x is to be provided.

### Usage

```
JensenFactor(x)
```

### Arguments

| | |
|---|---|
| x | must be a single value. Provide distance in meters. |

### Details

If a second turbine (B) is in a first turbine (A)'s wake, turbine B's power output must be penalized due to turbulence. This function computes the penalty factor for that wake.

### Value

`JensenFactor` returns the a single number between 0 and 1, which can immediately be used a a penalty factor.

### Note

Note that for Jensen's multiple wake model, wake penalties from several potentially influencing upwind turbines must be computed. As the factors are between 0 and 1, they can conveniently multiplied by each other afterward. If there are four turbines (A, B, C, D) inside a wind farm, you are investigating turbine D and it turns out that D is in the wake of B and C, but not A, then compute penalty for B (assume 0.7) and C (assume 0.8). Now, the penalty factors for A, B and C are 1, 0.7 and 0.8, so 1 * 0.7 * 0.8 = 0.56 is the penalty for turbine D.

### Author(s)

Carsten Croonenbroeck

### References

Jensen, N. O. (1983). A note on wind generator interaction. Roskilde: Risø National Laboratory. Risø-M, No. 2411

### See Also

Use `JensenAngle` to compute the wake cone and with it, use `JensenTrapezoid` to see if another turbine B is in turbine A's wake. Apply `JensenFactor` only if this is the case.

### Examples

```
JensenFactor(500)
## Provided that turbine B is in turbine A's wake and that the two turbines are 500 meters apart,
## turbine B must be penalized by a factor of 0.7952.
```

---

## `JensenTrapezoid` *Computes the four corner points of a Jensen trapezoid (or cone).*

---

### Description

Provided a wind direction, a point and a downwind length, computes the corner points of a Jensen trapezoid (in the literature oftentimes called a 'cone').

### Usage

```
JensenTrapezoid(WindDir, Point, x)
```

### Arguments

| | |
|---|---|
| `WindDir` | unique value, a wind direction in degrees. |
| `Point` | a vector of two containing x and y coordinates of a point. Usually, both will be between 0 and 1. |
| `x` | downwind distance of the cone in meters. |

### Value

`JensenTrapezoid` returns matrix of four columns (the four corner points) and two rows (x and y coordinates).

### Author(s)

Carsten Croonenbroeck

### References

Jensen, N. O. (1983). A note on wind generator interaction. Roskilde: Risø National Laboratory. Risø-M, No. 2411

## See Also

`JensenAngle` to compute 'cone' angle information, `PointInPolygon` for a test whether a point is inside a polygon.

## Examples

```
MyTrapezoid <- JensenTrapezoid(45, c(0.5, 0.5), 500)
```

---

## PairPenalty *Returns the Jensen wake penalty factor for a pair of turbines.*

---

## Description

As seen from a turbine in the wind farm, computes the wake penalty factor for another turbine in that farm.

## Usage

```
PairPenalty(x1, y1, x2, y2, Dirs, SDs)
```

## Arguments

| | |
|---|---|
| x1 | must be a single value. Provide the x location of the first turbine. |
| y1 | must be a single value. Provide the y location of the first turbine. |
| x2 | must be a single value. Provide the x location of the second turbine. |
| y2 | must be a single value. Provide the y location of the second turbine. |
| Dirs | a matrix containing average yearly wind directions. Usually, the third element of the list object `FarmData` will be used as this matrix. |
| SDs | a matrix containing average yearly wind direction standard deviations. Usually, the fourth element of the list object `FarmData` will be used as this matrix. |

## Details

First, this function uses `GetAngle` to compute the angle between the two points provided, as seen from point 2's point of view. It then obtains the wind direction at point 2 using `GetDirInfo`. After that, the distance between the two points is computed. With it, the wake cone is computed using `JensenAngle` to check whether point 2 is in point 1's wake using `JensenTrapezoid`. If that is the case, `JensenFactor` is used to compute the penalty factor.

## Value

`PairPenalty` returns a single number between 0 and 1. If point 2 is not in the wake of point 1, the function returns 1.

## Author(s)

Carsten Croonenbroeck

## See Also

Use `JensenFactor` to see how this function operates. See `FarmVars` for the data object.

## Examples

```
Dirs <- FarmData[[3]][e$FarmVars$StartPoint:e$FarmVars$EndPoint,
e$FarmVars$StartPoint:e$FarmVars$EndPoint]
SDs <- FarmData[[4]][e$FarmVars$StartPoint:e$FarmVars$EndPoint,
e$FarmVars$StartPoint:e$FarmVars$EndPoint]
PairPenalty(0.9, 0.8, 0.6, 0.9, Dirs, SDs)
## Weak wake penalty
PairPenalty(0.1, 0.1, 0.6, 0.9, Dirs, SDs)
## No wake penalty
```

---

`PlotResult` *Visualizes the wind farm layout optimization result.*

---

## Description

This function draws the adjusted yields of the wind farm under investigation using `image`, superimposes a contour plot using `contour` and arrows for the wind directions using `ImposeVectorField` and then draws the points for the turbines' locations (aligned to their respective raster grid centers).

## Usage

```
PlotResult(Result, ImageData = NULL, DoLabels = FALSE, Labels = "IDs")
```

## Arguments

| | |
|---|---|
| `Result` | must be an optimization result as returned by an optimizer such as `optim`. Usually, this will be a `list` at least containing '$par'. Most optimizers comply with this R standard. If your optimizer does not, wrap its result into a list containing an object '$par' containing the optimization result (i.e., a vector of points). |
| `ImageData` | a matrix containing the data for the background, processed via `image`. If not provided (or NULL), uses the data in `FarmData[[1]]` or, if present, `e$FarmData[[1]]`. Defaults to NULL. |
| `DoLabels` | a boolean that indicates whether labels should be plotted next to the points. Defaults to FALSE. |
| `Labels` | a vector of length n = N / 2, can be numeric values or strings. Defines the labels to be shown if `DoLabels = TRUE`. Defaults to "IDs", in which case the points are sequently numbered from 1:n. `Labels` is ignored if `DoLabels = FALSE`. |

## Details

For maximum convenience and compatibility with numeric optimizers of most kinds, this function expects nothing but the usual optimization result `list`. This, however, requires that the `FarmData` dataset as well as the `FarmVars` object is present defining additional settings.

### Value

`PlotResult` returns nothing.

### Author(s)

Carsten Croonenbroeck

### See Also

Use `Profit` to obtain an optimization result.

### Examples

```
#Will not provide a very good result
NumTurbines <- 4
set.seed(1357)
Result <- optim(par = runif(NumTurbines * 2), fn = Profit,
  method = "L-BFGS-B", lower = rep(0, NumTurbines * 2),
  upper = rep(1, NumTurbines * 2))
Result
PlotResult(Result)
```

---

## `PointInPolygon` *Checks whether a point is inside a polygon.*

---

### Description

Used the famous Jensen test to check whether a provided point is inside a polygon, the latter defined by a set of points.

### Usage

```
PointInPolygon(PointMat, TestPoint)
```

### Arguments

| | |
|---|---|
| `PointMat` | a 2 x 4 matrix, four corner points of each x and y coordinates. |
| `TestPoint` | a vector of two containing x and y coordinates of a point to test. |

### Value

`PointInPolygon` returns `TRUE` if the point is inside the polygon, or `FALSE`, else.

### Author(s)

Carsten Croonenbroeck

## References

Jensen, N. O. (1983). A note on wind generator interaction. Roskilde: Risø National Laboratory. Risø-M, No. 2411

## See Also

`JensenTrapezoid` to compute a Jensen trapezoid.

## Examples

```
set.seed(1357)
Angle <- runif(n = 1, min = 0, max = 360)
Point <- runif(n = 2, min = 0, max = 1)
Dist <- rnorm(n = 1, mean = 200, sd = 200)

MyTrapezoid = JensenTrapezoid(Angle, Point, Dist)

plot(x = MyTrapezoid[1, ], y = MyTrapezoid[2, ], xlab = "", ylab = "")
lines(x = c(MyTrapezoid[1, 1], MyTrapezoid[1, 2]), y = c(MyTrapezoid[2, 1], MyTrapezoid[2, 2]))
lines(x = c(MyTrapezoid[1, 2], MyTrapezoid[1, 3]), y = c(MyTrapezoid[2, 2], MyTrapezoid[2, 3]))
lines(x = c(MyTrapezoid[1, 3], MyTrapezoid[1, 4]), y = c(MyTrapezoid[2, 3], MyTrapezoid[2, 4]))
lines(x = c(MyTrapezoid[1, 4], MyTrapezoid[1, 1]), y = c(MyTrapezoid[2, 4], MyTrapezoid[2, 1]))

NumTest <- 50

xTest <- runif(n = NumTest, min = min(MyTrapezoid[1, ]), max = max(MyTrapezoid[1, ]))
yTest <- runif(n = NumTest, min = min(MyTrapezoid[2, ]), max = max(MyTrapezoid[2, ]))

for (i in 1:NumTest)
{
ThisPoint <- c(xTest[i], yTest[i])
if (PointInPolygon(MyTrapezoid, ThisPoint))
{
points(ThisPoint[1], ThisPoint[2], pch = 16, col = "green")
} else
{
points(ThisPoint[1], ThisPoint[2], pch = 16, col = "red")
}
}
```

---

# `Profit` *Computes the economic profit for a given wind farm layout configuration*

---

## Description

This is the bread-and-butter function to this package. It takes a set of points (turbine locations) and based on the adjusted yields in the field specified via the `FarmVars` settings object, checks whether the layout is valid, computes the Jensen penalties, generates the total marketable power production of this layout, takes cost and sale price into consideration and finally computes the farm's economic profit. Note, however, that since most numeric optimizers by default operate as a minimizer, `Profit` returns the negative profit, i.e. a negative number for positive profit values and a positive number if cost is greater than revenue.

## Usage

```
Profit(X)
```

## Arguments

| | |
|---|---|
| `X` | must be a numeric vector containing an even number of values, at least two. If the length of the vector is N, then n = N / 2 is the number of points. The first values 1,...,n are interpreted as x coordinates and the subsequent n + 1,...,N values are the y coordinates. |

## Details

For maximum convenience and compatibility with numeric optimizers of most kinds, this function expects nothing but the problem vector (the set of points) as a parameter. This, however, requires that the `FarmData` dataset as well as the `FarmVars` object is present defining additional settings.
As `Profit` returns the negative profit for compatibility with minimizers, reverse the sign for actual profit values.
`Profit` requires the optimizer to accept box constraints, as `Profit` can not compute values outside the wind farm boundary (the function's domain). If your optimizer does not accept box constraints, embed `Profit` into a wrapper function that returns the sum of costs if it least one point is outside the boundary.

## Value

`Profit` returns a single number. The result is the negative profit for any valid setting of points and consequently, the sum of all costs for invalid settings.

## Author(s)

Carsten Croonenbroeck

## See Also

Use `PlotResult` to visualize the optimization result.

## Examples

```
#Will not provide a very good result
NumTurbines <- 4
set.seed(1357)
Result <- optim(par = runif(NumTurbines * 2), fn = Profit,
  method = "L-BFGS-B", lower = rep(0, NumTurbines * 2),
  upper = rep(1, NumTurbines * 2))
Result
PlotResult(Result)
```

```
##########################################
#Will provide a somewhat better result
#Necessary to install pso

if(requireNamespace("pso")){
library(pso)
NumTurbines <- 4
Result <- psoptim(par = runif(NumTurbines * 2), fn = Profit,
  lower = rep(0, NumTurbines * 2), upper = rep(1, NumTurbines * 2))
Result
PlotResult(Result)}
##########################################
#Simple wrapper function for optimizers not accepting box constraints
NumTurbines <- 4
lower <- rep(0, NumTurbines * 2)
upper <- rep(1, NumTurbines * 2)
Wrapper <- function(X)
{
xSel <- seq(from = 1, to = length(X) - 1, by = 2)
x <- X[xSel]
y <- X[xSel + 1]

if (any(x < lower) | any(x > upper) | any(y < lower) | any(y > upper))
{
return(sum(rep(FarmVars$UnitCost, length(x))))
}

return(Profit(X))
}
## Not run:
set.seed(1357)
Result <- optim(par = runif(NumTurbines * 2), fn = Wrapper, method = "SANN")
Result
PlotResult(Result)
## End(Not run)
```

---

`ProfitContributors` *Computes profit contributions for all points in a setup solution.*

---

## Description

This function takes an optimizer result (or simple vector object) and computes the profit contribution for each point.

## Usage

```
ProfitContributors(Result)
```

## Arguments

| | |
|---|---|
| `Result` | must be an optimization result as returned by an optimizer such as `optim` or a vector. Usually, this will be a `list` at least containing '$par'. |

## Details

Neglecting a possibly invalid setup (caused by at least one point), computes the profit contributions. If the setup provided is valid, the sum of contributions is identical to the (absulute value of the) returned value of `Profit`.

## Value

`ProfitContributors` returns a matrix of two columns and n rows, whith n being the number of turbines. The first column is a sequence of 1:n, representing the turbine IDs, while the second column contains the actual profit contributions.

## Author(s)

Carsten Croonenbroeck

## See Also

See `ValidSetup` to see how a setup is categorized as valid or not. Use `PlotResult` to visualize the optimization result.

## Examples

```
#Prints a result and uses the profit contributions as labels.
NumTurbines <- 4
set.seed(1235)
Result <- optim(par = runif(NumTurbines * 2), fn = Profit,
  method = "L-BFGS-B", lower = rep(0, NumTurbines * 2),
  upper = rep(1, NumTurbines * 2))
MyLabels <- ProfitContributors(Result)
MyLabels
#PlotResult(Result, DoLabels = TRUE, Labels = MyLabels[, 2])

# Given a valid setup, this should be TRUE.
#identical(abs(Profit(Result$par)), sum(MyLabels[, 2]))
```

---

## `ShowWakePenalizers` *Visualizes the points causing/'suffering' from wake effects.*

---

## Description

After optimization, this function draws a reduced 'field' and shows which points cause wake penalties on which points. 'Causers' are displayed in grey, 'sufferers', i.e. points that are in (possibly multiple) wakes of (several) causers are displayed in gold.

## Usage

```
ShowWakePenalizers(Result, Cones = TRUE, VectorField = TRUE)
```

## Arguments

| | |
|---|---|
| `Result` | must be an optimization result as returned by an optimizer such as `optim`. Usually, this will be a `list` at least containing '$par'. Most optimizers comply with this R standard. If your optimizer does not, wrap its result into a list containing an object '$par' containing the optimization result (i.e., a vector of points). |
| `Cones` | triggers whether the Jensen cones are to be displayed as well. Defaults to `TRUE`. |
| `VectorField` | controls whether or not to impose the wind directions vector field (arrows). Defaults to `TRUE`. |

## Details

Enables the researcher to inspect the optimization result in an a-posteriori analysis of Jensen's wake model.

## Value

`ShowWakePenalizers` invisibly returns a square matrix (dimension: 'number of turbines') of wake penalty pairs: Columns are sufferers, rows are causers. For each row/column pair unequal to one, the turbine in row i sheds wake on the turbine in column j. The actual numbers are those returned by `JensenFactor`.

## Author(s)

Carsten Croonenbroeck

## See Also

Use `PlotResult` to visualize the optimization result.

## Examples

```
## Not run:
#Will show that turbine 1 sheds wake on turbine 5.
NumTurbines <- 8
set.seed(1357)
Result <- optim(par = runif(NumTurbines * 2), fn = Profit,
  method = "L-BFGS-B", lower = rep(0, NumTurbines * 2),
  upper = rep(1, NumTurbines * 2))
ShowWakePenalizers(Result)

## End(Not run)
```

## ValidSetup — *Checks whether all turbine locations provided satisfy the minimum distance criterion.*

### Description

For a set of turbine locations represented by `x` and `y`, checks whether all possible pairs satisfy the minimum distance criterion.

### Usage

```
ValidSetup(x, y)
```

### Arguments

| | |
|---|---|
| `x` | must be a numeric vector of at least two values, contains the 'x' location(s) of turbines. |
| `y` | must be a numeric vector of at least two values, contains the 'y' location(s) of turbines. |

### Value

`ValidSetup` returns `TRUE` if all pairs of turbines are at least as far away from each other as 'MinDist' from the `FarmVars` settings object requests, or `FALSE`, else.

### Author(s)

Carsten Croonenbroeck

### Examples

```
ValidSetup(c(0.5, 0.7), c(0.7, 0.9))
## Returns TRUE if FarmVars$MinDist is at its standard value (0.1).
```

## wflo — *Data set and functions for wind farm layout optimization.*

### Description

This package makes two contributions to the Wind Farm Layout Optimization (WFLO) research branch: First, it provides a convenient and realistic data set of high resolution and accuracy encompassing wind speeds, wind directions, standard deviations of wind directions, and (adjusted) yields for the entire area of Germany. Second, it supplies a set of helper functions and a benchmark function for economically (profit) driven wind farm layout optimization. This enables researchers in the field of the np-hard problem of wflo to focus on their optimization methodology contribution and also provides a realistic benchmark setting for comparability among contributions.

### Author(s)

Carsten Croonenbroeck
David Hennecke

---

`Yield` *Returns yearly yield for a turbine's location.*

---

### Description

For a turbine's location represented by `x` and `y`, looks up the (adjusted) yield from the matrix `Adj`. Internally transforms coordinates of `x` and `y` from problem space (usually unit square) to the matrix space of `Adj`.

### Usage

```
Yield(x, y, Adj)
```

### Arguments

| | |
|---|---|
| `x` | must be a single value containing the 'x' location of a turbine in problem space. |
| `y` | must be a single value containing the 'y' location of a turbine in problem space. |
| `Adj` | a matrix containing adjusted yields. Usually, the first element of the list object `FarmData` will be used as this matrix. |

### Details

Adjusted yields are the projected yearly average yields dependent on wind speed, hub height and other settings at each point in the raster data. Annual Energy Production (AEP) at a specific location, weighted by a location quality correction factor, produces adjusted yields. This adjustment returns a better guess on the marketable yield at a specific point. For details on the data, see the data set description to this package.

### Value

`Yield` returns a single value.

### Author(s)

Carsten Croonenbroeck

### See Also

`Profit` to see where to use `Yield`, `Cost` for a similar function for yearly cost. `FarmData` for the data set.

## Examples

```
## Returns adjusted yield for the given location.
Adj <- FarmData[[1]][e$FarmVars$StartPoint:e$FarmVars$EndPoint,
e$FarmVars$StartPoint:e$FarmVars$EndPoint]
Yield(0.5, 0.7, Adj)

## Replace the function by another function
## also called 'Yield', embedded in environment e.
## Also, see the vignette.

if(requireNamespace("pso")){
library(pso)
e$Yield <- function(x, y, AEP) #x, y \in R
{
return(x + y)
}
set.seed(1357)
Result <- psoptim(par = runif(NumTurbines * 2), fn = Profit,
  lower = rep(0, NumTurbines * 2), upper = rep(1, NumTurbines * 2))
Result
rm(Yield, envir = e)
}
```

# Index