hrosailing-Module Documentation

Dependencies

The hrosailing-module has the following third-party dependencies

- numpy
- matplotlib
- tabulate
- scipy

How To Use This Module

After installing/downloading one can easily use the hrosailing-module via

```
>>> import hrosailing
```

or

```
>>> from hrosailing import something
```

Contents Of This Module

The hrosailing-module defines the following functions:

hrosailing.**to_csv**(csv_path, obj)

Parameters :

csv_path : string

Path where a .csv-file is located or where a new .csv-file will be created

obj : PolarDiagram

An hrosailing.**PolarDiagram** instance which will be written to the .csv-file

Calls the .**to_csv**-function of the hrosailing.**PolarDiagram** instance.

hrosailing.**from_csv**(csv_path, fmt='hro', tws=True, twa=True)

Parameters :

`csv_path` : `string`

Path to an existing .csv file which you want to be read

`fmt` : `string`

The format of the .csv file. Currently supported formats are:
`'hro'`, `'orc'`, `'opencpn'`, `'array'`

`tws` : `bool`

Specifies wether or not the occuring wind speeds are true wind.
Will be passed to the constructor of the `hrosailing`.**PolarDiagram**
instance

`twa` : `bool`

Specifies wether or not the occuring wind angles are true wind
Will be passed to the constructor of the `hrosailing`.**PolarDiagram**
instance

Creates an `hrosailing`.**PolarDiagram** instance from the data that is written in the given .csv file via the `csv`.**reader**-class, see reader.

The .csv file needs to adhere to the format specified by the parameter `fmt`.

`'hro'`: The format created by the `hrosailing`.**to_csv** function
`'orc'`: The format found at ORC (without beat and run angles)
`'opencpn'`: The format created by the OpenCPN Polar Plugin
`'array'`:

`hrosailing`.**pickling**(`pkl_path, obj`)

Parameters :

`pkl_path` : `string`

Path to an existing .pkl file or where the created .pkl file will be located

`obj` : `PolarDiagram`

An `hrosailing`.**PolarDiagram** instance which will be written to the .pkl file

Calls the .**pickling**-function of the `hrosailing`.**PolarDiagram** instance.

`hrosailing`.**depickling**(`pkl_path`)

Parameters :

`pkl_path` : `string`

Path to an existing .pkl file which is to be read

Creates an `hrosailing`.**PolarDiagram** instance from the data that is written in the given .pkl file, via the `pickle`.**load**-function, see load.

`hrosailing`.**convert**(`obj, convert_type`)

Parameters :

`obj` : `PolarDiagram`

An instance of a subclass of `hrosailing`.**PolarDiagram**

`convert_type` : `PolarDiagram`

A subclass of `hrosailing`.**PolarDiagram**

Converts `obj` to an instance of `convert_type`. Currently only works with the subclasses `hrosailing`.**PolarDiagramTable** and `hrosailing`.**PolarDiagramPointcloud** hrosailing.**symmetric_polar_diagram**(obj)

Parameters :

obj : PolarDiagram

An instance of a subclass of hrosailing.**PolarDiagram**

Symmetrizes a given instance of a subclass of hrosailing.**PolarDiagram**. I.E. for every tuple of (wind speed, wind angle, boat speed) that is contained in obj in some form, the function creates a new instance of the same subclass of hrosailing.**PolarDiagram**, such that the tuples (wind speed, wind angle, boat speed) and (wind speed, 360 - wind angle, boat speed) are containted within it in some form.

Currently only works for the subclasses hrosailing.**PolarDiagramTable** and hrosailing.**PolarDiagramPointcloud**

Should only be used for instances of hrosailing.**PolarDiagramTable** if the wind speed resolution ranges from 0 to 180 or 180 to 360 to avoid conflicting data

The hrosailing-module defines the following public classes:

hrosailing.**PolarDiagram**()

An abstract base class for most classes in the hrosailing-module

The **PolarDiagram** class defines the following public methods:

PolarDiagram.**pickling**(self, pkl_path)

Parameters :

pkl_path : string

Path to an existing .pkl file or where the created .pkl file will be located

Creates or overwrites a .pkl file, with the class data of the instance which called the function, via the pickle.**dump**-function, see dump.

The **PolarDiagram** class also defines the following abstract methods:

PolarDiagram.**to_csv**(csv_path)

PolarDiagram.**polar_plot_slice**(wind_speed, ax=None, **kwargs)

PolarDiagram.**flat_plot_slice**(wind_speed, ax=None, **kwargs)

PolarDiagram.**polar_plot**(wind_speed_range, ax=None, min_color='green', max_color='red', **kwargs)

PolarDiagram.**flat_plot**(wind_speed_range, ax=None, min_color='green', max_color='r', **kwargs)

PolarDiagram.**plot_color_gradient**(ax=None, min_color='green', max_color='red')

PolarDiagram.**plot_convex_hull_slice**(wind_speed, ax=None, **kwargs)

hrosailing.**PolarDiagramTable**(wind_speed_resolution=None, wind_angle_resolution=None, data=None, tws=True, twa=True)

A class to represent, visualize and work with a polar performance diagram in form of a table.

The parameter wind_speed_resolution (resp. wind_angle_resolution) can either be Iterable (of int and/or float values), int or float and determines the number of columns (resp. rows) the Table will have.

If an Iterable is passed, the number of columns (resp. rows) will be the same as the number of elements in the Iterable, if an int or float is passed, the number of columns (resp. rows) will be the number of elements of numpy.arange(wind_speed_resolution, 40, wind_speed_resolution) (resp. numpy.arange(wind_angle_resolution, 360, wind_angle_resolution))

If no custom wind_speed_resolution (resp. wind_angle_resolution) is passed, it will default to numpy.arange(2,42,2) (resp. numpy.arange(0, 360, 5))

The parameter tws (resp. twa) is a bool that specifies wether the wind speeds in wind_speed_resolution (resp. the wind angles in wind_angle_resolution) are to be viewed as true wind.

If tws (resp. twa) is set to *False*, the wind speeds (resp. wind angles) will be converted into true wind.

The parameter data is a numpy.ndarray of matching shape that contains the boat speeds matching the wind speeds and angles in the resolution. If no custom data is passed, it will default to numpy.zeros((rdim, cdim)) where rdim and cdim are number of rows and columns respectively, determined by wind_angle_resolution and wind_speed_resolution

The **PolarDiagramTable** class has the following (private) attriubutes:

_resolution_wind_speed

_resolution_wind_angle

_data

The **PolarDiagramTable** class defines the following dunder methods:

PolarDiagramTable.**__str__**()

PolarDiagramTable.**__repr__**()

PolarDiagramTable.**__getitem__**(wind_tup)

Parameters :

wind_tup : tuple of length 2

Tuple to specify the row and column entry of the table, given as elements of the wind angle and wind speed resolution

Returns specified entry of the table

The **PolarDiagramTable** class defines the following public methods:

PolarDiagramTable.**wind_speeds**

Returns a read only version of *_resolution_wind_speed*

PolarDiagramTable.**wind_angles**

Returns a read only version of *_resolution_wind_angle*

PolarDiagramTable.**boat_speeds**

Returns a read only version of *_data*

PolarDiagramTable.**to_csv**(csv_path)

Parameters :

csv_path : string

Path to an existing .csv file or where the created .csv file will be located

Creates or overwrites a .csv file with the class data of object which called the function via the csv.**writer**-class, see writer.

The format of the .csv file will be as follows:

PolarDiagramTable
Wind speed resolution:
self.wind_speeds

Wind angle resolution:
self.wind_angles
Boat speeds:
self.boat_speeds

with the delimiter being ','.

`PolarDiagramTable.`**change_entries**`(new_data, wind_speeds=None, wind_angles=None, tws=True, twa=True)`

Parameters :

`new_data` : `int`, `float` or `array_like` of matching shape

New data that will be written in the specified entries

`wind_speeds` : `Iterable`, `int` or `float`

Column entries where the data is to be changed, given by elements of
the wind speed resolution

`wind_angles` : `Iterable`, `int` or `float`

Row entries where the data is to be changed, given by elements of
the wind angle resolution

`tws` : `bool`

Specifies wether or not wind_speeds is to be viewed as true wind
If set to *False*, `wind_speeds` will be converted to true wind

`twa` : `bool`

Specifies wether or not wind_angles is to be viewed as true wind
If set to *False*, `wind_angles` will be converted to true wind

Changes the data in the specified entries in the table to the input new data. This function alters *_data*

`PolarDiagramTable.`**polar_plot_slice**`(wind_speed, ax=None, **kwargs)`

Parameters :

`wind_speed` : `int` or `float`

Slice of the polar diagram that is to be plotted, given as an element of the wind speed resolution

`ax` : `matplotlib.axes.Axes`

A `matplotlib.axes.Axes` instance on which will be plotted on
Needs to be a `matplotlib.axes_subplots.PolarAxesSubplot`
If nothing is passed, the function will create a `matplotlib.axes.Axes`
instance via the `matplotlib.pyplot.`**gca** function, see gca

`kwargs` : Keyword arguments to change the appearence of the created plot.

Supports the same keyword arguments as the
`matplotlib.pyplot.`**plot** function
However if no 'linestyle' (resp. 'markerstyle') is passed
it will default to '' (resp. 'o')

Creates a polar plot of a given slice of the polar diagram, via the
`matplotlib.pyplot.`**plot** function, see plot

PolarDiagramTable.**flat_plot_slice**(wind_speed, ax=None, **kwargs)

Parameters :

wind_speed : int or float

Slice of the polar diagram that is to be plotted, given as an element of the wind speed resolution

ax : matplotlib.axes.Axes

A matplotlib.axes.Axes instance on which will be plotted on
Needs to be a matplotlib.axes._subplots.AxesSubplot
If nothing is passed, the function will create a matplotlib.axes.Axes
instance via the matplotlib.pyplot.**gca** function, see gca

kwargs : Keyword arguments to change the appearence of the created plot.

Supports the same keyword arguments as the
matplotlib.pyplot.**plot** function
However if no 'linestyle' (resp. 'markerstyle') is passed
it will default to '' (resp. 'o')

Creates a cartesian plot of a given slice of the polar diagram, via the matplotlib.pyplot.**plot** function, see plot

PolarDiagramTable.**polar_plot**
(wind_speed_range, ax=None, min_color='green',

max_color='red', **kwargs)

Parameters :

wind_speed_range : Iterable

The range of wind speeds to be plotted, given as an Iterable of elements of the wind speed resolution

ax : matplotlib.axes.Axes

A matplotlib.axes.Axes instance on which will be plotted on
Needs to be a matplotlib.axes_subplots.PolarAxesSubplot
If nothing is passed, the function will create a matplotlib.axes.Axes
instance via the matplotlib.pyplot.**gca** function, see gca

min_color :

max_color :

kwargs : Keyword arguments to change the appearence of the created plot.

Supports the same keyword arguments as the
matplotlib.pyplot.**plot**-function
However if no 'linestyle' (resp. 'markerstyle') is passed
it will default to '' (resp. 'o')

Creates a color coded polar plot of multiple slices, given by wind_speed_range, of the polar diagram, via the matplotlib.pyplot.**plot** function, see plot

PolarDiagramTable.**flat_plot** (wind_speed_range, ax=None,

min_color='green', max_color='red', **kwargs)

Parameters :

wind_speed_range : Iterable

The range of wind speeds to be plotted, given as an `Iterable` of elements of the wind speed resolution

`ax` : `matplotlib.axes.Axes`

A `matplotlib.axes.Axes` instance on which will be plotted on
Needs to be a `matplotlib.axes._subplots.AxesSubplot`
If nothing is passed, the function will create a `matplotlib.axes.Axes` instance via the `matplotlib.pyplot.`**gca** function, see gca

`min_color` :

`max_color` :

`kwargs` : Keyword arguments to change the appearence of the created plot.

Supports the same keyword arguments as the
`matplotlib.pyplot.`**plot**-function
However if no 'linestyle' (resp. 'markerstyle') is passed
it will default to '' (resp. 'o')

Creates a color coded cartesian plot of multiple slices, given by `wind_speed_range`, of the polar diagram, via the `matplotlib.pyplot.`**plot** function, see plot

`PolarDiagramTable.`**plot_color_gradient**`(ax=None, min_color='green', max_color='red')`

Parameters :

`ax` : `matplotlib.axes.Axes`

A `matplotlib.axes.Axes` instance on which will be plotted on
Needs to be a `matplotlib.axes._subplots.AxesSubplot`
If nothing is passed, the function will create a `matplotlib.axes.Axes` instance via the `matplotlib.pyplot.`**gca** function, see gca

`min_color` :

`max_color` :

`PolarDiagramTable.`**plot_convex_hull_slice**`(wind_speed, ax=None, **kwargs)`

Parameters :

`wind_speed` : `int` or `float`

Slice of the polar diagram that is to be plotted, given as an element
of the wind speed resolution

`ax` : `matplotlib.axes.Axes`

A `matplotlib.axes.Axes` instance on which will be plotted on
Needs to be a `matplotlib.axes_subplots.PolarAxesSubplot`
If nothing is passed, the function will create a `matplotlib.axes.Axes` instance via the `matplotlib.pyplot.`**gca** function, see gca

`kwargs` : Keyword arguments to change the appearence of the created plot.

Supports the same keyword arguments as the
`matplotlib.pyplot.`**plot**-function

Computes the convex hull of a given slice of the polar diagram table, via the `scipy.spatial.`**ConvexHull** function, see ConvexHull
and then creates a polar plot of the convex hull, via the `matplotlib.pyplot.`**plot** function, see plot

hrosailing.**PolarDiagramCurve**(f, *params)

A class to represent, visualize and work with a polar performance diagram given as a fitted curve with a list of optimal parameters

The parameter f should be a function of the form f(x, *params), where x should be array_like with dimension 2 (the rows should correspond to pairs of wind speeds and wind angles), and determines the curve which describes the polar diagram.

The parameter *params should contain the resulting parameters that are obtained via a fitting of f.

The **PolarDiagramCurve** class has the following (private) attributes:

_f

_params

The **PolarDiagramCurve** class defines the following dunder methods:

PolarDiagramCurve.**__repr__**()

PolarDiagramCurve.**__call__**(wind_speed, wind_angle)

Parameters :

wind_speed : numpy.ndarray, int or float

wind_angle : numpy.ndarray, int or float

Calls self.**curve** with the specified values. wind_speed and wind_angle should be of matching shape

The **PolarDiagramCurve** class defines the following public methods:

PolarDiagramCurve.**curve**

Returns a read only version of self.*_f*

PolarDiagramCurve.**parameters**

Returns a read only version of self.*_params*

PolarDiagramCurve.**to_csv**(csv_path)

Parameters :

csv_path : string

Path to an existing .csv file or where the created .csv file will be located

Creates or overwrites a .csv file with the class data of object which called the function via the csv.**writer**-class, see writer.

The format of the .csv file will be as follows:

PolarDiagramCurve
Function: self.curve
Parameters: self.parameters

with the delimiter ':'

PolarDiagramCurve.**polar_plot_slice**(wind_speed, ax=None, **kwargs)

Parameters :

wind_speed : int or float

A slice of the polar diagram that is to be plotted, given as the true wind speed

ax : matplotlib.axes.Axes

A matplotlib.axes.Axes instance on which will be plotted on Needs to be a matplotlib.axes_subplots.PolarAxesSubplot

If nothing is passed, the function will create a `matplotlib.axes.Axes` instance via the `matplotlib.pyplot`.**gca** function, see gca

`kwargs` : Keyword arguments to change the appearence of the created plot.

Supports the same keyword arguments as the
`matplotlib.pyplot`.**plot**-function
However if no 'linestyle' (resp. 'markerstyle') is passed
it will default to '' (resp. 'o')

Creates a polar plot of a given slice of the polar diagram, via the `matplotlib.pyplot`.**plot** function, see plot

`PolarDiagramCurve.`**flat_plot_slice**`(wind_speed, ax=None, **kwargs)`

Parameters :

`wind_speed` : `int` or `float`

A slice of the polar diagram that is to be plotted, given as the true wind speed

`ax` : `matplotlib.axes.Axes`

A `matplotlib.axes.Axes` instance on which will be plotted on
Needs to be a `matplotlib.axes._subplots.AxesSubplot`
If nothing is passed, the function will create a `matplotlib.axes.Axes` instance via the `matplotlib.pyplot`.**gca** function, see gca

`kwargs` : Keyword arguments to change the appearence of the created plot.

Supports the same keyword arguments as the
`matplotlib.pyplot`.**plot**-function
However if no 'linestyle' (resp. 'markerstyle') is passed
it will default to '' (resp. 'o')

Creates a cartesian plot of a given slice of the polar diagram, via the `matplotlib.pyplot`.**plot** function, see plot

`PolarDiagramCurve.`**polar_plot**`(wind_speed_range=(0,20), ax=None, min_color='green', max_color='red', **kwargs)`

Parameters :

`wind_speed_range` : `tuple` of length 2

The range of wind speeds to be plotted, given as a lower and upper bound of the true wind speed

`ax` : `matplotlib.axes.Axes`

A `matplotlib.axes.Axes` instance on which will be plotted on
Needs to be a `matplotlib.axes_subplots.PolarAxesSubplot`
If nothing is passed, the function will create a `matplotlib.axes.Axes` instance via the `matplotlib.pyplot`.**gca** function, see gca

`min_color` :

`max_color` :

`kwargs` : Keyword arguments to change the appearence of the created plot.

Supports the same keyword arguments as the
`matplotlib.pyplot.`**plot**-function
However if no 'linestyle' (resp. 'markerstyle') is passed
it will default to '' (resp. 'o')

Creates a color coded polar plot of multiple slices, given by `wind_speed_range` of the polar diagram, vie the `matplotlib.pyplot.`**plot** function, see plot

`PolarDiagramCurve.`**flat_plot**`(wind_speed_range=(0,20), ax=None, min_color = 'green', max_color='red', **kwargs)`

Parameters :

`wind_speed_range` : `tuple` of length 2

The range of wind speeds to be plotted, given as a lower and upper
bound of the true wind speed

`ax` : `matplotlib.axes.Axes`

A `matplotlib.axes.Axes` instance on which will be plotted on
Needs to be a `matplotlib.axes._subplots.AxesSubplot`
If nothing is passed, the function will create a `matplotlib.axes.Axes`
instance via the `matplotlib.pyplot.`**gca** function, see gca

`min_color` :

`max_color` :

`kwargs` : Keyword arguments to change the appearence of the created plot.

Supports the same keyword arguments as the
`matplotlib.pyplot.`**plot**-function
However if no 'linestyle' (resp. 'markerstyle') is passed
it will default to '' (resp. 'o')

Creates a color coded cartesian plot of multiple slices, given by `wind_speed_range`, of the polar diagram, via the `matplotlib.pyplot.`**plot** function, see plot

`PolarDiagramCurve.`**plot_color_gradient**`(wind_speed_range=(0,20), ax=None, min_color='green', max_color='red')`

Parameters :

`wind_speed_range` : `tuple` of length 2

The range of wind speeds to be plotted, given as a lower and upper
bound of the true wind speed

`ax` : `matplotlib.axes.Axes`

A `matplotlib.axes.Axes` instance on which will be plotted on
Needs to be a `matplotlib.axes._subplots.AxesSubplot`
If nothing is passed, the function will create a `matplotlib.axes.Axes`
instance via the `matplotlib.pyplot.`**gca** function, see gca

`min_color` :

`max_color` :

`PolarDiagramCurve.`**plot_convex_hull_slice**`(wind_speed, ax=None **kwargs)`

Parameters :

`wind_speed` : `int` or `float`

A slice of the polar diagram that is to be plotted, given as the true wind speed

`ax` : `matplotlib.axes.Axes`

A `matplotlib.axes.Axes` instance on which will be plotted on
Needs to be a `matplotlib.axes_subplots.PolarAxesSubplot`
If nothing is passed, the function will create a `matplotlib.axes.Axes` instance via the `matplotlib.pyplot.`**gca** function, see gca

`kwargs` : Keyword arguments to change the appearence of the created plot.

Supports the same keyword arguments as the `matplotlib.pyplot.`**plot**-function

Computes the convex hull of a given slice of the polar diagram table, via the `scipy.spatial.`**ConvexHull** function, see ConvexHull
and then creates a polar plot of the convex hull, via the `matplotlib.pyplot.`**plot** function, see plot

`hrosailing.`**PolarDiagramPointcloud**(`points=None, tws=True, twa=True`)

A class to present, visualize and work with a polar performance diagram in form of a point cloud.

The parameter `points` should be `array_like` of shape (_, 3) and determines the points that are in the point cloud at the beginning. A point should be of length 3 such that the first entry corresponds to the wind speed, the second to the wind angle and the last to the boat speed.

If no `points` are passed, it will default to an empty array numpy.array([])

The parameter `tws` (resp. `twa`) specifies wether or not the wind speeds (resp. wind angles) given in `points` should be viewed as true wind.

If `tws` (resp. `twa`) is set to *False*, the wind speeds (resp. wind angles) will be converted into true wind.

The **PolarDiagramPointcloud** class has to following (private) attributes:

_data

The **PolarDiagramPointcloud** class defines the following dunder methods:

`PolarDiagramPointcloud.`**__str__**()

`PolarDiagramPointcloud.`**__repr__**()

`PolarDiagramPointcloud`**__getitem__**()

Returns

The **PolarDiagramPointcloud** class defines the following public methods:

`PolarDiagramPointcloud.`**wind_speeds**

Returns a list of all occuring wind speeds

`PolarDiagramPointcloud.`**wind_angles**

Returns a list of all occuring wind angles

`PolarDiagramPointcloud.`**points**

Returns a read only version of `self.`*_data*

`PolarDiagramPointcloud.`**to_csv**(`csv_path`)

Parameters :

`csv_path` : `string`

Path to an existing .csv file or where the created .csv file will be located

Creates or overwrites a .csv file with the class data of object which called the function via the `csv`.**writer**-class, see writer.

The format of the .csv file will be as follows:

PolarDiagramPointcloud
True Wind Speed ,True Wind Angle ,Boat Speed
self.points

with the delimiter ','

`PolarDiagramPointcloud.`**add_points**(`new_points, tws=True, twa=True`)

Parameters :

`new_points` : `array_like` of shape (_, 3)

New points that are to be added to the point cloud. The point should
be of length 3, with the first entry being the wind speed,
the second being the wind angle and the last being the boat speed

`tws` : `bool`

Specifies wether or not the wind speeds are to be viewed as true wind
If set to *False*, the given wind speeds will be converted to true wind

`twa` : `bool`

Specifies wether or not the wind angles are to be viewed as true wind
If set to *False*, the given wind angles will be converted to true wind

`PolarDiagramPointcloud.`**change_points**()

Parameters :

`PolarDiagramPointcloud.`**polar_plot_slice**(`wind_speed, ax=None, **kwargs`)

Parameters :

`wind_speed` : `int` or `float`

A slice of the polar diagram that is to be plotted, given as the
true wind speed

`ax` : `matplotlib.axes.Axes`

A `matplotlib.axes.Axes` instance on which will be plotted on
Needs to be a `matplotlib.axes_subplots.PolarAxesSubplot`
If nothing is passed, the function will create a `matplotlib.axes.Axes`
instance via the `matplotlib.pyplot`.**gca** function, see gca

`kwargs` : Keyword arguments to change the appearence of the created plot.

Supports the same keyword arguments as the
`matplotlib.pyplot`.**plot**-function
However if no 'linestyle' (resp. 'markerstyle') is passed
it will default to '' (resp. 'o')

`PolarDiagramPointcloud.`**flat_plot_slice**(`wind_speed, ax=None, **kwargs`)

Parameters :

`wind_speed` : `int` or `float`

A slice of the polar diagram that is to be plotted, given as the true wind speed

`ax` : `matplotlib.axes.Axes`

A `matplotlib.axes.Axes` instance on which will be plotted on
Needs to be a `matplotlib.axes._subplots.AxesSubplot`
If nothing is passed, the function will create a `matplotlib.axes.Axes` instance via the `matplotlib.pyplot.`**gca** function, see gca

`kwargs` : Keyword arguments to change the appearence of the created plot.

Supports the same keyword arguments as the
`matplotlib.pyplot.`**plot**-function
However if no 'linestyle' (resp. 'markerstyle') is passed
it will default to '' (resp. 'o')

Creates a cartesian plot of a given slice of the polar diagram, via the `matplotlib.pyplot.`**plot** function, see plot

`PolarDiagramPointcloud.`**polar_plot**`(wind_speed_range=(0, numpy.inf), ax=None, min_color='green', max_color='red', **kwargs)`

Parameters :

`wind_speed_range` : `tuple` of length 2

The range of wind speeds to be plotted, given as a lower and upper bound of the true wind speed

`ax` : `matplotlib.axes.Axes`

A `matplotlib.axes.Axes` instance on which will be plotted on
Needs to be a `matplotlib.axes_subplots.PolarAxesSubplot`
If nothing is passed, the function will create a `matplotlib.axes.Axes` instance via the `matplotlib.pyplot.`**gca** function, see gca

`min_color` :

`max_color` :

`kwargs` : Keyword arguments to change the appearence of the created plot.

Supports the same keyword arguments as the
`matplotlib.pyplot.`**plot**-function
However if no 'linestyle' (resp. 'markerstyle') is passed
it will default to '' (resp. 'o')

Creates a color coded polar plot of multiple slices, given by `wind_speed_range` of the polar diagram, vie the `matplotlib.pyplot.`**plot** function, see plot

`PolarDiagramPointcloud.`**flat_plot**`(wind_speed_range=(0, numpy.inf), ax=None, min_color='green', max_color='red', **kwargs)`

Parameters :

`wind_speed_range` : `tuple` of length 2

The range of wind speeds to be plotted, given as a lower and upper bound of the true wind speed

`ax` : `matplotlib.axes.Axes`

A `matplotlib.axes.Axes` instance on which will be plotted on
Needs to be a `matplotlib.axes._subplots.AxesSubplot`

If nothing is passed, the function will create a `matplotlib.axes.Axes` instance via the `matplotlib.pyplot`.**gca** function, see gca

`min_color` :

`max_color` :

`kwargs` : Keyword arguments to change the appearence of the created plot.

Supports the same keyword arguments as the
`matplotlib.pyplot`.**plot**-function
However if no 'linestyle' (resp. 'markerstyle') is passed
it will default to '' (resp. 'o')

Creates a color coded cartesian plot of multiple slices, given by `wind_speed_range`, of the polar diagram, via the `matplotlib.pyplot`.**plot** function, see plot

`PolarDiagramPointcloud`.**plot_color_gradient**(`ax=None, min_color='green', max_color='red'`):

Parameters :

`ax` : `matplotlib.axes.Axes`

A `matplotlib.axes.Axes` instance on which will be plotted on
Needs to be a `matplotlib.axes._subplots.AxesSubplot`
If nothing is passed, the function will create a `matplotlib.axes.Axes` instance via the `matplotlib.pyplot`.**gca** function, see gca

`min_color` :

`max_color` :

`kwargs` : Keyword arguments to change the appearence of the created plot.

Supports the same keyword arguments as the
`matplotlib.pyplot`.**plot**-function

`PolarDiagramPointcloud`.**plot_convex_hull_slice**(`wind_speed, ax=None, **kwargs`)

Parameters :

`wind_speed` : `int` or `float`

A slice of the polar diagram that is to be plotted, given as the
true wind speed

`ax` : `matplotlib.axes.Axes`

A `matplotlib.axes.Axes` instance on which will be plotted on
Needs to be a `matplotlib.axes_subplots.PolarAxesSubplot`
If nothing is passed, the function will create a `matplotlib.axes.Axes` instance via the `matplotlib.pyplot`.**gca** function, see gca

`kwargs` : Keyword arguments to change the appearence of the created plot.

Supports the same keyword arguments as the
`matplotlib.pyplot`.**plot**-function

Computes the convex hull of a given slice of the polar diagram table, via the
`scipy.spatial`.**ConvexHull** function, see ConvexHull
and then creates a polar plot of the convex hull, via the `matplotlib.pyplot`.**plot**
function, see plot