`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 public functions:

`hrosailing.polar_diagram.`**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.polar_diagram.`**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.polar_diagram.`**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.polar_diagram.`**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.polar_diagram.`**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.polar_diagram.`**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

`hrosailing.data_processing.`**filter_points**(`w_points, f_func=None, f_mode='bound', **filter_kw`)

Parameters :

`w_points` : `WeightedPoints`

`f_func` : `function`

Function, that will determine which
points will be filtered out, depending
on a given weight.

`f_mode` : `str`

Filtering-method if no `f_func` is passed
Currently the two available methods are
bound where the points are filtered by
a given upper and lower bound and
percentage where the points are filtered
according to an empirical percentile.

`filter_kw` : Keyword arguments

Either possible keyword arguments of `f_func`
or the following:

`l_b` (`int/float`) - sets the lower bound for `f_mode` *bound*, defaults to `numpy`.**NINF**, see NINF

`u_b` (`int/float`) - sets the upper bound for `f_mode` *bound*, defaults to `numpy`.**inf**, see inf

`percent` (`int/float`) - determines empirical percentile for `f_mode` *percentage*, defaults to *50*

`hrosailing.data_processing.`**interpolate_points**(`points, w_res=None, i_func=None`)

Parameters :

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

`w_res` : `tuple` of length 2 or `str` *"auto"*

`i_func` : `function`

Function to interpolate `points`
If no function is passed, the default method

of interpolation uses the
`scipy.interpolate.`**bisplrep** and
`scipy.interpolate.`**bisplev** functions, see
bisplrep and bisplev

`hrosailing.data_processing.`**create_polar_diagram**(`data, p_type=PolarDiagramTable, w_func=None, f_func=None, i_func=None, w_res=None tws=True, twa=True, w_func_kw=None, **filter_kw`)

Parameters :

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

`p_type` : `PolarDiagram`, optional

A `hrosailing.polar_diagram.`**PolarDiagram** subclass

`tws` : `bool`, optional

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

`twa` : `bool`, optional

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

`w_func` : `function`, optional

Weight-Function passed on to
`hrosailing.data_processing.`**WeightedPoints**

`f_func` : `function`, optional

Filter-Function passed on to
`hrosailing.data_processing.`**filter_points**

`i_func` : `function`, optional

Interpolating-Function passed on to
`hrosailing.data_processing.`**interpolate_points**

`w_res` : `tuple` of length 2 or `str` *"auto"* , optional

Only needed, if `p_type` is PolarDiagramTable
Tuple containing the wind speed resolution and
wind angle resolution for the created
PolarDiagramTable instance

`w_func_kw` : `dict`

Keyword arguments passed on to
`hrosailing.data_processing.`**WeightedPoints**

`filter_kw` : Keyword arguments

Keyword arguments passed on to

hrosailing.data_processing.**filter_points**

The hrosailing-module defines the following public classes:

hrosailing.polar_diagram.**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**(ws, ax=None, **plot_kw)

PolarDiagram.**flat_plot_slice**(ws, ax=None, **plot_kw)

PolarDiagram.**polar_plot**(ws_range, ax=None, colors=('green', 'red'), show_legend=True, legend_kw=None, **plot_kw)

PolarDiagram.**flat_plot**(ws_range, ax=None, colors=('green', 'red'), show_legend=True, legend_kw=None, **plot_kw)

PolarDiagram.**plot_color_gradient**(ax=None, colors=('green', 'red'), marker=None, show_legend=True, legend_kw=None)

PolarDiagram.**plot_convex_hull_slice**(ws, ax=None, **plot_kw)

hrosailing.polar_diagram.**PolarDiagramTable**(ws_res=None, wa_res=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 ws_res (resp. wa_res)
can either be an Iterable (of int and float values), int or float
It 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 is passed, the number of columns (resp. rows) will be the number of
elements in numpy.arange(ws_res, 40, ws_res)
(resp. numpy.arange(wa_res, 360, wa_res))
If a float is passed, the number of columns (resp. rows) will be the number of
elements in numpy.linspace(``)
(resp. numpy.linspace(``))

If no custom ws_res (resp. wa_res) 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 `ws_res` (resp. the wind angles in `wa_res`)
are to be viewed as true wind
If `tws` (resp. `twa`) is set to *False*, the wind speeds (resp. wind angles)
will be converted to 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 `wa_res` and `ws_res`

|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, ws=None, wa=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
If no `ws` and no `wa` is passed,
the required shape is the shape of *_data*

`ws` : `Iterable`, `int` or `float`

Column entries where the data is to be changed, given by elements of the wind speed resolution
If `None` is passed, the function changes all entries in the rows specified by `wa`.
If `wa` is also `None`, the function changes all
entries in the table

`wa` : `Iterable`, `int` or `float`

Row entries where the data is to be changed, given by elements of the wind angle resolution
If `None` is passed, the function changes all entries in the
columns specified by `ws`
If `ws` is also `None`, the function changes all
entries in the table

`tws` : `bool`

Specifies wether or not wind_speeds is to be viewed as true wind
If set to *False*, `ws` 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*, `wa` 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**(ws, ax=None, **plot_kw)

Parameters :

ws : 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, optional

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

plot_kw : 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**(ws, ax=None, **plot_kw)

Parameters :

ws : 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, optional

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

plot_kw : 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** (ws_range=None, ax=None,

colors=('green', 'red'), show_legend=True, legend_kw=None, **plot_kw)

Parameters :

ws_range : Iterable, optional

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

`ax` : `matplotlib.axes.Axes`, optional

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

`colors` : `Iterable`, optional

Specifies the colors to be used for the different slices
If there are at most as many slices as colors, each slice will be plotted with the specified color
If there are more slices than colors the function will either cycle through the specified colors until all slices have been plotted
or if there are exactly two colors specified, the function will plot the slices with a color gradient using those two colors
Elements of the `Iterable` can be of any type accepted by the `matplotlib.colors`.**to_rgb** function, see to_rgb and colors

`show_legend` : `bool`, optional

Specifies wether or not a legend should be added to the plot.
The type of legend depends on the amount of slices and colors.
If `colors` is of length 2 and `ws_range` is of length greater 2, the legend will be a
`matplotlib.colorbar.`**Colorbar** instance, see Colorbar
If `colors` and `ws_range` are of the same length,
the legend will be a `matplotlib.legend.`**Legend** instance, see Legend

`legend_kw` : `dict`, optional

Keyword arguments to change the appearence and location of the legend
Supports the same keyword arguments as either the
`matplotlib.axes.Axes.`**legend** function, see legend
or the `matplotlib.pyplot`**colorbar** function, see colorbar

`plot_kw` : 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')
If 'colors' (or 'c') is passed, it will be deleted. Use the
parameters `colors` instead

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** (`ws_range=None, ax=None, colors=('green', 'red'), show_legend=True, legend_kw=None, **plot_kw`)

Parameters :

`ws_range` : `Iterable`, optional

The range of wind speeds to be plotted, given as an `Iterable` of elements of the wind speed resolution `ax` : `matplotlib.axes.Axes`, optional

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

`colors` : `Iterable`, optional

Specifies the colors to be used for the different slices
If there are at most as many slices as colors, each slice will be
plotted with the specified color
If there are more slices than colors the function will either cycle
through the specified colors until all slices have been plotted
or if there are exactly two colors specified, the function will
plot the slices with a color gradient using those two colors
Elements of the `Iterable` can be of any type accepted by the
`matplotlib.colors`.**to_rgb** function, see to_rgb and colors

`show_legend` : `bool`, optional

Specifies wether or not a legend should be added to the plot.
The type of legend depends on the amount of slices and colors.
If `colors` is of length 2 and `ws_range` is of length
greater 2, the legend will be a
`matplotlib.colorbar.`**Colorbar** instance, see Colorbar
If `colors` and `ws_range` are of the same length,
the legend will be a `matplotlib.legend.`**Legend** instance, see Legend

`legend_kw` : `dict`, optional

Keyword arguments to change the appearence and location of the
legend
Supports the same keyword arguments as either the
`matplotlib.axes.Axes.`**legend** function, see legend
or the `matplotlib.pyplot`**colorbar** function, see colorbar

`plot_kw` : 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')
If 'colors' (or 'c') is passed, it will be deleted. Use the
parameters `colors` instead

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, colors=('green', 'red'), marker=None, show_legend=True, *legend_kw`)

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

`colors` : `tuple` of length 2

`marker` : `matplotlib.markers.Markerstyle`, optional

Specifies the style of the markers in the plot
For all possible styles, see marker
Defaults to 'o'

`show_legend` : `bool`, optional

Specifies wether or not a legend should be added to the plot.
Legend will be a `matplotlib.colorbar.`**Colorbar** instance, see Colorbar

`legend_kw` : Keyword arguments to change the appearence and position of the legend

Supports the same keyword arguments as the
`matplotlib.pyplot`**colorbar** function, see colorbar

`PolarDiagramTable`.**plot_convex_hull_slice**(`ws, ax=None, **plot_kw`)

Parameters :

`ws` : `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`, optional

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

`plot_kw` : 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.polar_diagram`.**PolarDiagramCurve**(`f,`

`radians=False, *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` of shape(_, 2) (the rows should correspond to pairs of
wind speeds and wind angles), and determines the curve which describes the polar diagram.

The parameter `radians` is `bool` that specifies, wether `f`
takes the wind angles to be in radians or degrees

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

_radians

_params

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

`PolarDiagramCurve.`**__repr__**()

`PolarDiagramCurve.`**__call__**(`ws, wa`)

Parameters :

`ws` : `numpy.ndarray`, `int` or `float`

`wa` : `numpy.ndarray`, `int` or `float`

Calls `self.`**curve** with the specified values
`ws` and `wa` should be of matching shape and type

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

`PolarDiagramCurve.`**curve**

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

`PolarDiagramCurve.`**radians**

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

`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
Radians: self.radians
Parameters: self.parameters

with the delimiter ':'

`PolarDiagramCurve.`**polar_plot_slice**(`ws, ax=None, **plot_kw`)

Parameters :

`ws` : `int` or `float`

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

`ax` : `matplotlib.axes.Axes`, optional

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

`plot_kw` : 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**`(ws, ax=None, **plot_kw)`

Parameters :

`ws` : `int` or `float`

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

`ax` : `matplotlib.axes.Axes`, optional

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

`plot_kw` : 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**`(ws_range=(0,20,5), ax=None,`

`colors=('green', 'red'), show_legend=True, legend_kw=None, **plot_kw)`

Parameters :

`ws_range` : `tuple` of length 3, optional

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

`ax` : `matplotlib.axes.Axes`, optional

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

`colors` : `Iterable`, optional

Specifies the colors to be used for the different slices
If there are at most as many slices as colors, each slice will be
plotted with the specified color

If there are more slices than colors the function will either cycle
through the specified colors until all slices have been plotted
or if there are exactly two colors specified, the function will
plot the slices with a color gradient using those two colors
Elements of the `Iterable` can be of any type accepted by the
`matplotlib.colors.`**to_rgb** function, see to_rgb and colors

`show_legend` : `bool`, optional

Specifies wether or not a legend should be added to the plot.
The type of legend depends on the amount of slices and colors.
If `colors` is of length 2 and `ws_range` is of length
greater 2, the legend will be a
`matplotlib.colorbar.`**Colorbar** instance, see Colorbar
If `colors` and `ws_range` are of the same length,
the legend will be a `matplotlib.legend.`**Legend** instance, see Legend

`legend_kw` : `dict`, optional

Keyword arguments to change the appearence and location of the
legend
Supports the same keyword arguments as either the
`matplotlib.axes.Axes.`**legend** function, see legend
or the `matplotlib.pyplot`**colorbar** function, see colorbar

`plot_kw` : 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')
If 'colors' (or 'c') is passed, it will be deleted. Use the
parameters `colors` instead

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**`(ws_range=(0,20,5), ax=None, colors=('green', 'red'), show_legend=True, legend_kw=None, **plot_kw)`

Parameters :

`ws_range` : `tuple` of length 2 , optional

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

`ax` : `matplotlib.axes.Axes`, optional

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

`colors` : `Iterable`, optional

Specifies the colors to be used for the different slices
If there are at most as many slices as colors, each slice will be

plotted with the specified color
If there are more slices than colors the function will either cycle through the specified colors until all slices have been plotted or if there are exactly two colors specified, the function will plot the slices with a color gradient using those two colors
Elements of the `Iterable` can be of any type accepted by the `matplotlib.colors`.**to_rgb** function, see to_rgb and colors

`show_legend` : `bool`, optional

Specifies wether or not a legend should be added to the plot.
The type of legend depends on the amount of slices and colors.
If `colors` is of length 2 and `ws_range` is of length greater 2, the legend will be a `matplotlib.colorbar.`**Colorbar** instance, see Colorbar
If `colors` and `ws_range` are of the same length, the legend will be a `matplotlib.legend.`**Legend** instance, see Legend

`legend_kw` : `dict`, optional

Keyword arguments to change the appearence and location of the legend
Supports the same keyword arguments as either the `matplotlib.axes.Axes.`**legend** function, see legend
or the `matplotlib.pyplot`**colorbar** function, see colorbar

`plot_kw` : 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')
If 'colors' (or 'c') is passed, it will be deleted. Use the parameters `colors` instead

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**(`ws_range=(0,20), ax=None, colors=('green', 'red'), marker=None, show_legend=True, **legend_kw`)

Parameters :

`ws_range` : `tuple` of length 2, optional

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

`ax` : `matplotlib.axes.Axes`, optinal

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

`colors` : `tuple` of length 2

`marker` : `matplotlib.markers.Markerstyle`, optional

Specifies the style of the markers in the plot
For all possible styles, see marker
Defaults to 'o'

`show_legend` : `bool`, optional

Specifies wether or not a legend should be added to the plot.
Legend will be a `matplotlib.colorbar.`**Colorbar** instance, see Colorbar

`legend_kw` : Keyword arguments to change the appearence and position of the legend

Supports the same keyword arguments as the
`matplotlib.pyplot`**colorbar** function, see colorbar

`PolarDiagramCurve.`**plot_convex_hull_slice**(`ws, ax=None **plot_kw`)

Parameters :

`ws` : `int` or `float`

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

`ax` : `matplotlib.axes.Axes`, optional

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

`plot_kw` : 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.polar_diagram.`**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__**()

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**(`ws, ax=None, **plot_kw`)

Parameters :

`ws` : `int` or `float`

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

`ax` : `matplotlib.axes.Axes`, optional

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**(`ws, ax=None, **plot_kw`)

Parameters :

`ws` : `int` or `float`

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

`ax` : `matplotlib.axes.Axes`, optional

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**(`ws_range=(0, numpy.inf), ax=None, colors=('green', 'red'), show_legend=True, legend_kw=None, **plot_kw`)

Parameters :

`ws_range` : `tuple` of length 2, optional

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

`ax` : `matplotlib.axes.Axes`, optional

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

`colors` : `Iterable`, optional

Specifies the colors to be used for the different slices
If there are at most as many slices as colors, each slice will be
plotted with the specified color
If there are more slices than colors the function will either cycle
through the specified colors until all slices have been plotted
or if there are exactly two colors specified, the function will
plot the slices with a color gradient using those two colors

Elements of the `Iterable` can be of any type accepted by the `matplotlib.colors.`**to_rgb** function, see to_rgb and colors

`show_legend` : `bool`, optional

Specifies wether or not a legend should be added to the plot.
The type of legend depends on the amount of slices and colors.
If `colors` is of length 2 and `ws_range` is of length
greater 2, the legend will be a
`matplotlib.colorbar.`**Colorbar** instance, see Colorbar
If `colors` and `ws_range` are of the same length,
the legend will be a `matplotlib.legend.`**Legend** instance, see Legend

`legend_kw` : `dict`, optional

Keyword arguments to change the appearence and location of the
legend
Supports the same keyword arguments as either the
`matplotlib.axes.Axes.`**legend** function, see legend
or the `matplotlib.pyplot`**colorbar** function, see colorbar

`plot_kw` : 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')
If 'colors' (or 'c') is passed, it will be deleted. Use the
parameters `colors` instead

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**(`ws_range=(0, numpy.inf), ax=None, colors=('green', 'red'), show_legend=True, legend_kw=None, **plot_kw`)

Parameters :

`ws_range` : `tuple` of length 2, optional

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

`ax` : `matplotlib.axes.Axes`, optional

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

`colors` : `Iterable`, optional

Specifies the colors to be used for the different slices
If there are at most as many slices as colors, each slice will be
plotted with the specified color
If there are more slices than colors the function will either cycle
through the specified colors until all slices have been plotted
or if there are exactly two colors specified, the function will
plot the slices with a color gradient using those two colors

Elements of the `Iterable` can be of any type accepted by the `matplotlib.colors.`**to_rgb** function, see to_rgb and colors

`show_legend` : `bool`, optional

Specifies wether or not a legend should be added to the plot.
The type of legend depends on the amount of slices and colors.
If `colors` is of length 2 and `ws_range` is of length
greater 2, the legend will be a
`matplotlib.colorbar.`**Colorbar** instance, see Colorbar
If `colors` and `ws_range` are of the same length,
the legend will be a `matplotlib.legend.`**Legend** instance, see Legend

`legend_kw` : `dict`, optional

Keyword arguments to change the appearence and location of the
legend
Supports the same keyword arguments as either the
`matplotlib.axes.Axes.`**legend** function, see legend
or the `matplotlib.pyplot`**colorbar** function, see colorbar

`plot_kw` : 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')
If 'colors' (or 'c') is passed, it will be deleted. Use the
parameters `colors` instead

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, colors=('green', 'red'), marker=None, show_legend=True, **legend_kw`):

Parameters :

`ax` : `matplotlib.axes.Axes`, optional

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

`colors` : `tuple` of length 2

`marker` : `matplotlib.markers.Markerstyle`, optional

Specifies the style of the markers in the plot
For all possible styles, see marker
Defaults to 'o'

`show_legend` : `bool`, optional

Specifies wether or not a legend should be added to the plot.
Legend will be a `matplotlib.colorbar.`**Colorbar** instance, see Colorbar

`legend_kw` : Keyword arguments to change the appearence and position of the legend

Supports the same keyword arguments as the
`matplotlib.pyplot`**colorbar** function, see colorbar

`PolarDiagramPointcloud.`**plot_convex_hull_slice**(`ws, ax=None, **plot_kw`)

Parameters :

`ws` : `int` or `float`

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

`ax` : `matplotlib.axes.Axes`, optional

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

`plot_kw` : 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.data_processing.`**WeightedPoints**(`points, w_func=None, tws=True, twa=True, **w_func_kw`)

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

_points

_weights

The **WeightedPoints** class defines the following public methods

`WeightedPoints.`**points**

Returns a read only version of *_points*

`WeightedPoints.`*_weights**

Returns a read only version of *_weights*

`hrosailing.data_processing.`**default_w_func**(`points, **w_func_kw`)

Parameters :

`points` : `numpy.ndarray` of shape (n,3)

`w_func_kw` : Keyword arguments

The possible keyword arguments are

st_point (`int`) -
outlier (`float`) -

The default w_func for the **WeightedPoints** class will give the Weight 1 to the first `st_point` Points.

Then it will through the remaining points, calculating the standard deviation of the wind speed, wind angle and boat speed of the `st_point` points that come before, using the `numpy`.**std** function, see std

It will then filter the occuring standard deviations by excluding the outermost `outlier` percent, by computing the associated empirical percentile.

After that, the function gives the wind speeds, wind angles and boat speeds the weight 1/standardvariation^2, or 0 if they were filtered out.

The weight of the points will then be the arithmetic mean of their respective wind speed, wind angle and boat speed

At last the function will normalize the weights and return them.