# hrosailing-Module Documentation

## Dependencies

The hrosailing-module has the following dependencies

- csv:
- loggin:
- pickle:
- numpy:
- matplotlib:
- abc:
- 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

obj : PolarDiagram

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

hrosailing.**from_csv**(csv_path)

**Parameters :**

csv_path **:** string

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

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

The .csv file needs to be in the right format which is found in the documentation of the corresponding .**to_csv**-function or in the source code. Note that this function doesn't work for the hrosailing.**PolarDiagramCurve** class.

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**-object which is to be written to the .pkl file

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

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

**Parameters :**

`pkl_path : string`

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

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

`hrosailing`.**convex_hull_polar**(`points_radians, points_angles`)

**Parameters :**

`points_radians` : `array_like`

`points_angles` : `array_like`

Computes the convex hull of the given point cloud via the `scipy.spatial`.**ConvexHull**-class, see ConvexHull.

`hrosailing.utils`.**polar_to_kartesian**(radians, angles)

**Parameters :**

`radians` : `array_like`

`angles` : `array_like`

The `hrosailing`-module defines the following classes and methods

`hrosailing`.**PolarDiagramException**(`type, *args`)

`hrosailing`.**PolarDiagram**()

`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 via with the class data of the object which called the function via the `pickle`.**dump**-function, see dump.

`hrosailing`.**PolarDiagramTable**(`**kwargs`)

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

```
>>> wind_angle_resolution = [52,60,75,90,110,120,135,150]
>>> wind_speed_resolution = [6,8,10,12,14,16,20]
>>> data = [[4.06,4.82,5.42,5.83,6.04,6.13,6.16],
...         [4.31,5.11,5.69,6.01,6.2,6.31,6.36],
...         [4.5,5.35,5.89,6.16,6.36,6.52,6.72],
...         [4.45,5.31,5.91,6.21,6.44,6.66,6.99],
...         [4.11,4.98,5.71,6.13,6.39,6.62,7.12],
...         [3.85,4.72,5.49,6,6.29,6.53,7.03],
...         [3.39,4.27,5,5.64,6.06,6.32,6.78],
...         [2.91,3.78,4.5,5.15,5.72,6.09,6.55]]
>>> polar_table = hrosailing.PolarDiagramTable(data=data,
                                               wind_speed_resolution=wind_speed_resolution,
                                               wind_angle_resolution=wind_angle_resolution)
```

Once initiated one can present the table in a nice way

```
>>> print(polar_table)
  TWA \ TWS     6     8    10    12    14    16    20
-----------  ----  ----  ----  ----  ----  ----  ----
         52  4.06  4.82  5.42  5.83  6.04  6.13  6.16
         60  4.31  5.11  5.69  6.01  6.2   6.31  6.36
         75  4.5   5.35  5.89  6.16  6.36  6.52  6.72
         90  4.45  5.31  5.91  6.21  6.44  6.66  6.99
        110  4.11  4.98  5.71  6.13  6.39  6.62  7.12
        120  3.85  4.72  5.49  6     6.29  6.53  7.03
        135  3.39  4.27  5     5.64  6.06  6.32  6.78
        150  2.91  3.78  4.5   5.15  5.72  6.09  6.55
```

PolarDiagramTable objects have the following public methods:

`PolarDiagramTable.__`**str**`__()`

Returns a tabulate of the PolarDiagramTable object via the `tabulate`.**tabulate**-function, see tabulate

If self._resolution_wind_speed has more than 15 elements, only the first 15 are used to create the table.

`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._resolution_wind_speed
Wind angle resolution:
self._resolution_wind_angle
Boat speeds:
self._data

with the delimiter ','.

`PolarDiagramTable`.**change_entry**(`**kwargs`)

**Parameters :**

`kwargs` : Keywords containing the entries in the "table" that are to be changed and the new data.

- `data` : `int`, `float` or `array_like` of matching shape
- `true_wind_speed` : `int`, `float` or `Iterable`
- `true_wind_angle` : `int`, `float` or `Iterable`

Updates `self`.**_data** on the specified entries with the given new data.

```
>>> polar_table.change_entry(data=4,
...                          true_wind_angle=52,
```

```
...                             true_wind_speed=6)
>>> print(polar_table)
  TWA \ TWS     6     8    10    12    14    16    20
-----------  ----  ----  ----  ----  ----  ----  ----
         52  4     4.82  5.42  5.83  6.04  6.13  6.16
         60  4.31  5.11  5.69  6.01  6.2   6.31  6.36
         75  4.5   5.35  5.89  6.16  6.36  6.52  6.72
         90  4.45  5.31  5.91  6.21  6.44  6.66  6.99
        110  4.11  4.98  5.71  6.13  6.39  6.62  7.12
        120  3.85  4.72  5.49  6     6.29  6.53  7.03
        135  3.39  4.27  5     5.64  6.06  6.32  6.78
        150  2.91  3.78  4.5   5.15  5.72  6.09  6.55
```

Can be used to change a whole row/column in one go:

```
>>> data = [6, 6.16,6.3,6.4,6.35,6.26,6.01,6.03]
>>> polar_table.change_entry(data=data,
...                             true_wind_angle=14)
>>> print(polar_table)
  TWA \ TWS     6     8    10    12    14    16    20
-----------  ----  ----  ----  ----  ----  ----  ----
         52  4     4.82  5.42  5.83  6     6.13  6.16
         60  4.31  5.11  5.69  6.01  6.16  6.31  6.36
         75  4.5   5.35  5.89  6.16  6.3   6.52  6.72
         90  4.45  5.31  5.91  6.21  6.4   6.66  6.99
        110  4.11  4.98  5.71  6.13  6.35  6.62  7.12
        120  3.85  4.72  5.49  6     6.26  6.53  7.03
        135  3.39  4.27  5     5.64  6.01  6.32  6.78
        150  2.91  3.78  4.5   5.15  6.03  6.09  6.55
```

Note that if both *true_wind_speed* and *true_wind_angle* are of type `Iterable`, the function will throw an error to prevent ambiguity.

`PolarDiagramTable`.**get_slice_data**(`true_wind_speed`)

**Parameters :**

`true_wind_speed` : `int` *or* `float`

Element in `self`.**_resolution_wind_speed**

Retrieves the corresponding column of `self`.**_data**. Throws an error if `true_wind_speed` is not in `self`.**_resolution_wind_speed**.

`PolarDiagramTable`.**polar_plot_slice**(`true_wind_speed, **kwargs`)

**Parameters :**

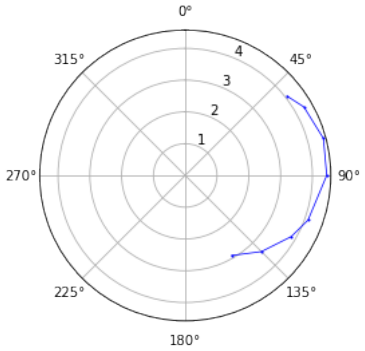
`true_wind_speed` : `int` *or* `float`

Element in `self`.**_resolution_wind_speed**

`kwargs` : Keyword arguments to change the appearence of the created plot. Supports the same keyword arguments as the `matplotlib.pyplot`.**plot**-function

For a given column of `self._data` corresponding to the input element of `self._resolution_wind_speed`, the function returns a polar plot of the column together with the corresponding elements in `self._resolution_wind_angle` via the `matlibplot.pyplot.plot`-function, see plot

```
>>> polar_table.polar_plot_slice(6, ms=1, marker='o', lw=0.75, ls='-')
```




`PolarDiagramTable.`**`flat_plot_slice`**`(true_wind_speed, **kwargs)`

**Parameters :**

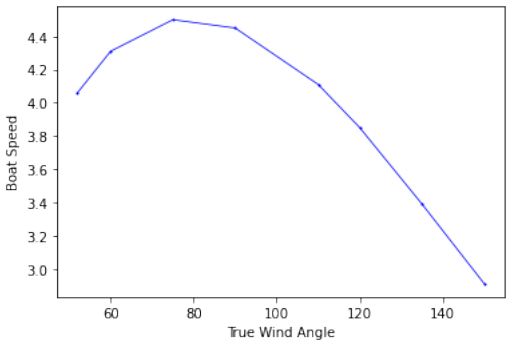
`true_wind_speed` : `int` *or* `float`

Element in `self._resolution_wind_speed`

`kwargs` : Keyword arguments to change the appearence of the created plot. Supports the same keyword arguments as the `matplotlib.pyplot.plot`-function

For a given column of `self._data` corresponding to the input element of `self._resolution_wind_speed`, the function returns a plot of the column entries as y-coordiantes together with the corresponding elements in `self._resolution_wind_angle` as x-coordinates via the `matlibplot.pyplot.plot`-function, see plot

```
>>> polar_table.flat_plot_slice(6, ms=1, marker='o', lw=0.75, ls='-')
```




`PolarDiagramTable.`**`plot_convex_hull_slice`**`(true_wind_speed, **kwargs)`

**Parameters :**

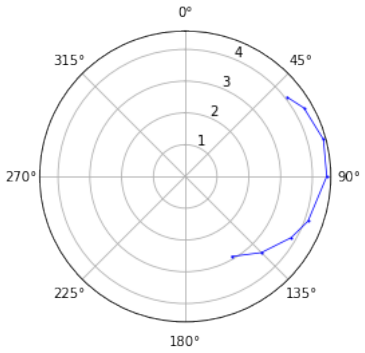
`true_wind_speed` : `int` *or* `float`

Element in `self._resolution_wind_speed`

`kwargs` : Keyword arguments to change the appearence of the created plot. Supports the same keyword arguments as the `matplotlib.pyplot.`**plot**-function

For a given column of `self.`**_data** corresponding to the input element of `self.`**_resolution_wind_speed**, the function computes the convex hull of the column entries together with the corresponding elements in `self.`**_resolution_wind_angle** via the `hrosailing.`**convex_hull_polar**-function and returns a polar plot of the computed convex hull via the `matplotlib.pyplot.`**plot**-function, see plot

```
>>> polar_table.plot_convex_hull_slice(6, ms=1, marker='o', lw=0.75, ls='-')
```




`hrosailing.`**PolarDiagramCurve**`(f, *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
Wind speed resolution:
self._resolution_wind_speed
Functions(s):
self._f
Parameters:
self._params

with the delimiter ','

`hrosailing.`**PolarDiagramPointcloud**`(data)`

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

PolarDiagramPointcloud objects have the following public methods:

`PolarDiagramPointcloud.__`**str**`__()`

`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._data

with the delimiter ','

`PolarDiagramPointcloud`.**add_points**(`points`)

**Parameters :**

`points` : `array_like`

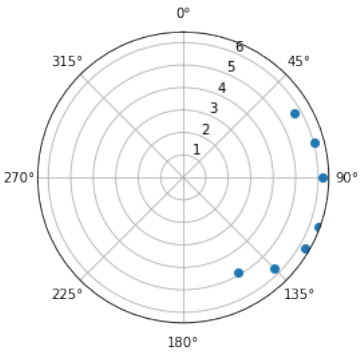
`PolarDiagramPointcloud`.**polar_plot_slice**(`true_wind_speed, **kwargs`)

**Parameters :**

`true_wind_speed` : `int` or `float`

`kwargs` : Keyword arguments to change the appearence of the created plot. Supports the same keyword arguments as the `matplotlib.pyplot`.**plot**-function

```
>>> polar_pointcloud.polar_plot_slice(8)
```




`PolarDiagramPointcloud`.**flat_plot_slice**(`true_wind_speed, **kwargs`)

**Parameters :**

`true_wind_speed` : `int` or `float`

`kwargs` : Keyword arguments to change the appearence of the created plot. Supports the same keyword arguments as the `matplotlib.pyplot`.**plot**-function

```
>>> polar_pointcloud.flat_plot_slice(8)
```

`PolarDiagramPointcloud.`**plot_convex_hull_slice**`(true_wind_speed, **kwargs)`

**Parameters :**

`true_wind_speed` : `int` or `float`

`kwargs` : Keyword arguments to change the appearence of the created plot. Supports the same keyword arguments as the `matplotlib.pyplot.`**plot**-function

```
>>> polar_point_cloud.plot_convex_hull_slice(8)
```