MPPy Documentation

Release 0.0.1

Tobias Machnitzki

Dec 12, 2017

CONTENTS

This package provides some tools for working with the data from the Barbados Cloud Observatory.

CHAPTER

ONE

FIRST STEPS

In this section I will try to cover some basics to work with this module.

1.1 Instruments

1.2 Tools

CHAPTER TWO

MODULES

2.1 Instruments

2.1.1 Radar

This Module contains the Radar class. This class is for easy working with the BCO radar data.

`Radar`(start, end[, device, version])	Class for working with radar data from Barbados.

Radar class

class `MPPy.Instruments.Radar.`**`Radar`**(*start*, *end*, *device='CORAL'*, *version=2*)

Class for working with radar data from Barbados.

Currently supported devices: CORAL, KATRIN

Parameters

- **`start`** – Either String or datetime.datetime-object indicating the start of the timefwindow
- **`end`** – Either String or datetime.datetime-object indicating the end of the timefwindow
- **`device`** – the device you want to use. Currently supported: CORAL, KATRIN
- **`version`** – The version of the dataset to use. Currently supported: 1,2,3 [note: 3 is in beta-phase]

Example

The following example initiates a radar object for the CORAL with a timewindow form the 1st January 2017 to the 2nd January 2017 to 3:30 pm:

```
>>> coral = Radar(start="20170101",end="201701021530", device="CORAL")
```

To review the attributes of your class you can use:

```
>>> print(coral)
CORAL Radar.
Used data version 2.
Load data from 2017-01-01 00:00:00 to 2017-01-02 15:30:00.
```

To get attributes of the device you just need to call the attribute now:

```
>>> coral.lat
array(13.162699699401855, dtype=float32)
```

To get measured values you need to call the appropriate method:

```
>>> coral.getReflectivity(postprocessing="Zf")
array([[...]], dtype=float32)
```

In most cases you want the timestamp as well:

```
>>> coral.getTime()
array([datetime.datetime(2017, 1, 1, 1, 0, 18), ...,
datetime.datetime(2017, 1, 2, 0, 59, 49)], dtype=object)
```

device
String of the device being used. ('CORAL' or 'KATRIN')

start
datetime.datetime object indicating the beginning of the chosen timewindow.

end
datetime.datetime object indicating the end of the chosen timewindow.

data_version
An Integer conatining the used version of the data (1,2,3[beta]) .

lat
Latitude of the instrument.

lon
Longitude of the instrument.

azimuth
Azimuth angle of where the instrument is pointing to.

elevation
Elevation angle of where the instrument is pointing to.

north
Degrees of where from the instrument seen is north.

Methods

getTime()	Loads the time steps over the desired timeframe from all netCDF-files and returns them as one array.
getRange()	Loads the getRange-gates from the netCDF-file which contains the last entries of the desired timeframe.
getReflectivity([postprocessing])	Loads the reflecitivity over the desired timeframe from multiple netCDF-files and returns them as one array.
getVelocity([target])	Loads the doppler velocity from the netCDF-files and returns them as one array
quickplot2D(value[, save_name, save_path, ylim])	Creates a fast Quickplot from the input value.
help()	This is a function for less experienced python-users.

getTime

Radar.**getTime**()

Loads the time steps over the desired timeframe from all netCDF-files and returns them as one array.

Returns A numpy array containing datetime.datetime objects

Example

Getting the time-stamps from an an already initiated radar object 'coral':

```
>>> coral.getTime()
```

getRange

Radar.**getRange**()

Loads the getRange-gates from the netCDF-file which contains the last entries of the desired timeframe. Note: just containing the range-gates from the first valid file of all used netCDF-files. If the range-gating changes over the input-timewindow, then you might run into issues.

Returns A numpy array with height in meters

Example

Getting the range-gates of an already initiated radar object called 'coral':

```
>>> coral.getRange()
```

getReflectivity

Radar.**getReflectivity**(*postprocessing='Zf'*)

Loads the reflecitivity over the desired timeframe from multiple netCDF-files and returns them as one array.

Parameters **postprocessing** – see Radar.help() for more inforamation

Returns 2-D numpy array with getReflectivity in dbz

Example

Getting the unfiltered and mie corrected reflectivity of all hydrometeors with an an already initiated radar object 'coral':

```
>>> coral.getReflectivity(postprocessing="Zu")
```

getVelocity

Radar.**getVelocity**(*target='hydrometeors'*)

Loads the doppler velocity from the netCDF-files and returns them as one array

Parameters **`target`** – String of which target the velocity you want to get from: 'hydrometeors' or 'all'.

Returns 2-D numpy array with doppler velocity in m/s

Example

This is how you could get the velocity from all targets for the 13th August 2016 to the 15th August 2016 of CORAL:

```
>>> coral = Radar(start="20160813",end="20160815", device="CORAL")
>>> velocity = Radar.getVelocity(target="all")
```

quickplot2D

`Radar.`**`quickplot2D`**`(`*value*, *save_name=None*, *save_path=None*, *ylim=None*`)`

Creates a fast Quickplot from the input value. Start and end date are the initialization-dates. To save the picture you can provide a name for the picture (save_name). If no savepath is provided, the picture will be stored in the current working directory.

Parameters

- **`value`** – A 2-D array which you want to plot.
- **`save_name`** – String: If provided picture will be saved under the given name. Example: 'quicklook.png'
- **`save_path`** – String: If provided, the picture will be saved at this location. Example: '/user/hoe/testuer/'
- **`ylim`** – Tuple: If provided the y-axis will be limited to these values.

Example

To just get a quicklook of the reflectivity to your screen try:

```
>>> coral = Radar(start="2017040215",end="201704021530", device="CORAL")
>>> coral.quickplot2D(value=coral.getReflectivity(),ylim=(100,2000))
```

help

`static` `Radar.`**`help`**`()`

This is a function for less experienced python-users. It will print some tipps for working with this Radar class. If possible use the documentation, it will be much more likely up to date and contains more information!

Returns Just prints some help messages into the console

2.2 Tools

2.2.1 Tools

This toolbox contains some functions which are being used by the MPPy package but might be usefull to the enduser, as well.

`daterange`(start_date, end_date)	This function is for looping over datetime.datetime objects within a timeframe from start_date to end_date.
`num2time`(num)	Converts seconds since 1970 to datetime objects.
`time2num`(time)	Converts a datetime.datetime object to seconds since 1970 as float.
`datestr`(dt_obj)	Converts a datetime.datetime object to a string in the commonly used shape for this module.

daterange

`MPPy.tools.tools.`**`daterange`**`(`*start_date*, *end_date*`)`

This function is for looping over datetime.datetime objects within a timeframe from start_date to end_date. It will only loop over days.

Parameters

- **`start_date`** – datetime.datetime object
- **`end_date`** – datetime.datetime object

Yields A datetime.datetime object starting from start_date and going to end_date

Example

If you want to loop over the dates from the 1st January 2017 to the 3rd January 2017:

```
>>> start = datetime.datetime(2017,1,1)
>>> end = datetime.datetime(2017,1,3)
>>> for x in daterange(start,end):
>>>     print(str(x))
2017-01-01 00:00:00
2017-01-02 00:00:00
2017-01-03 00:00:00
```

num2time

`MPPy.tools.tools.`**`num2time`**`(`*num*`)`

Converts seconds since 1970 to datetime objects. If input is a numpy array, ouput will be a numpy array as well.

Parameters **`num`** – float/ndarray. seconds since 1970

Returns datetime.datetime object

time2num

`MPPy.tools.tools.`**`time2num`**(*time*)

Converts a datetime.datetime object to seconds since 1970 as float. If input is a numpy array, ouput will be a numpy array as well.

Parameters **`time`** – datetime.datetime object / ndarray of datetime.datetime objects.

Returns Float of seconds since 1970 / ndarray of floats.

datestr

`MPPy.tools.tools.`**`datestr`**(*dt_obj*)

Converts a datetime.datetime object to a string in the commonly used shape for this module.

Parameters **`dt_obj`** – datetime.datetime object.

Returns String of the format YYMMDD. Y=Year, M=Month, D=Day.

CHAPTER

THREE

HELP

If you need any help or have trouble with the project please contact tobias.machnitzki@mpimet.mpg.de

PYTHON MODULE INDEX

INDEX