Wind_data python library

Cyril Gadal

December 17, 2018

The library simplify the downloading and the treatment of ERA-Interim data. It has been optimised for the study of 10-meters wind velocities, but can be easily adapted to other quantities with few modifications. For now, the reading of the grid files only works under linux. Other functions are pure python and can be used under any Python3 environment.

1 Requirements

1.1 ECMWF

The ERA-interim wind data are provided by the ECMWF, and this library use their api to download the data. A lot of information can be found in their website `https://software.ecmwf.int/wiki/display/WEBAPI/Access+ECMWF+Public+Datasets`. In particular, a *key* is needed to access their data, and can be obtained providing a simple registration on their website (see `https://confluence.ecmwf.int//display/WEBAPI/Access+ECMWF+Public+Datasets#AccessECMWFPublicDatasets-key`).

1.2 Python

This library requires any version of Python 3. The following packages are used:

- numpy
- matplotlib
- os
- windrose (see `https://pypi.org/project/windrose/`)
- ecmwfapi (see `https://confluence.ecmwf.int//display/WEBAPI/Access+ECMWF+Public+Datasets#AccessECMWFPublicDatasets-python`)

1.3 wgrib

Grib files are read by the 'wgrib' program that can be found here `http://www.cpc.ncep.noaa.gov/products/wesley/wgrib.html` with full documentation.

2 The Era_Interim library and the class Wind_data

The purpose of the class wind_data is to simplify the treatment of the ERA Interim wind data.

2.1 Attributes

The three first ones are to be imposed by the user in order to get any wind data fron the ECMWWF serveur:

- self.**name**, *string*
  Name of the data. For example, 'Sahara' or 'Skeleton_Coast'

- self.**grid_bounds**, *list of list of floats*
  Coordinates of the top west and bottom East point of the grid of the data as [[lat,lon], [lat,lon]].

- self.**years**, *list of list of integers*
  Dates of the start and the end of the time series as [[aaaa,mm,dd], [aaaa, mm, dd]].

The followings are 'None', and can be set afterwards by using functions defined in the next subsection:

- self.**grib_name**, *string*
  Name of the grib file. Automatically set by calling the function self.Update_grib_name().

- self.**coordinates**, *numpy array*
  Coordinates of the grid points. Automatically set by calling the function self.Update_coordinates(). Of dimension (Npoint_x*Npoint_y, Npoint_t).

- self.**Uwind**, *numpy array*
  10-meter U wind velocity at every point of the grid for every time step. Of dimension (Npoint_x, Npoint_y, Npoint_t). Set by self.load_wind_data().

- self.**Vwind**, *numpy array*
  10-meter V wind velocity at every point of the grid for every time step. Of dimension (Npoint_x, Npoint_y, Npoint_t). Set by self.load_wind_data()

- self.**Ustrengh**, *numpy array*
  10-meter norm of the wind velocity vector at every point of the grid for every time step. Of dimension (Npoint_x, Npoint_y, Npoint_t). Set by self.Cartesian_to_polar()

- self.**Uorientation**, *numpy array*
  10-meter direction of the wind velocity vector at every point of the grid for every time step. Of dimension (Npoint_x, Npoint_y, Npoint_t). Set by self.Cartesian_to_polar()

2.2 Class functions

▶ self.**Getting_wind_data(self, area_wanted, Nsplit)**

Arguments:

- **area_wanted**, *list of list of floats*, Coordinates of the top west and bottom East point of the grid of the data as [[lat,lon], [lat,lon]]. You can pass self.grid_bounds to this argument, or use a new one. For both cases,self.grid_bounds is updated at the end to match the exact grid points used by the ERA-Interim model.
- **Nplit**, *int*, Split the request into *Nsplit* ones for large requests.

Retrieving wind data from the serveur.

▶ self.**Update_grib_name(self)**

Arguments: None

Update self.grib_name.

▶ self.**Extract_UV(self, path_to_wgrib)**

Arguments:

- **path_to_wgrib**, *string*, Path to executable wgrib. Optional. Default is None, if wgrib is in already in python searching path, that can be seen by checking os.environ['PATH'].

Create two .txt files from the grib file, containing the data for U and V respectively.

▶ self.**load_wind_data(self)**

Arguments: None

Load the wind data from the .txt files into self.Uwind and self.Vwind.

▶ self.**Cartesian_to_polar(self)**

Arguments: None

Update self.Uorientation and self.Ustrength from self.Uwind and self.Vwind.

▶ self.**Write_wind_data(self,dir, pattern)**

Arguments:

- **dir**, *string*, directory in which the wind data will be written.
- **pattern**, *string*, Pattern for naming the files. Optional. Default is 'wind_data_'.

Write the wind data into .txt files named 'pattern1.txt', 'pattern2.txt', etc. First column of each file is the wind orientation (from self.Uorientation) and the second the wind strength (from self.Ustrength). Lines are time series, and file numbers correspond to grid points, whose coordinates can be found in self.coordinates.

▶ self.**Write_wind_rose(self, dir, ext, **kwargs)**

Arguments:

- **dir**, *string*, directory in which the wind rose will be written.
- **ext**, *string*, extension for the saving format of the roses. Optional. Default is '.pdf'.
- ****kwargs**, *multiple*, All optional arguments that can be passed to customize the wind rose plot, as 'cmap', 'bins', etc . Detail can be found in the windrose project website by downloading the manual.

Write the wind roses in the specified directory with names 'wind_rose_1.pdf', etc.

▶ self.**Write_flux_rose(self, dir, ext = '.pdf', grain_size = 180*10**-6, **kwargs)**

Arguments:

- **dir**, *string*, directory in which the wind rose will be written.
- **ext**, *string*, extension for the saving format of the roses. Optional. Default is '.pdf'.
- **grain_size**, *float*, Grain size used to calculate the sand flux distribution. See the description of PDF_flux() function.
- ****kwargs**, *multiple*, All optional arguments that can be passed to customize the wind rose plot, as 'color', 'bins', etc . Detail can be found in the description of the flux_rose() function.

Write the flux roses in the specified directory with names 'flux_rose_1.pdf', etc.

▶ self.**Write_spec(self,name)**

Arguments:

- **name**, *str*, Name under which the specifications of the class will be saved

Write self.name, self.grid_bounds, self.years to a .txt file.

▶ self.**Write_spec(self,name)**

Arguments:

- **name**, *str*, Name under which the specifications of the class will be loaded

Load self.name, self.grid_bounds, self.years from a .txt file.

▶ self.**Update_coordinates(self)**

Arguments: None

Update self.coordinates from self.grid_bounds.

▶ self.**Write_coordinates(self)**

Arguments: None

Write self.coordinates to a .txt file named 'Coordinates.txt'.

▶ self.**Create_KMZ(self)**

Arguments: None

Create a .kml file from 'Coordinates.txt' with the grid points. Meant to be opened with Google Earth.

2.3 Other functions

▶ **format_area(point_coordinates)**

Arguments:

- **point_coordinates**, *list of float*, Coordinate of a point as [lat,lon].

Convert list of coordinates into a string readable by the ECMWWF api.

▶ **format_time(date)**

Arguments:

- **date**, *list of float*, Date as [aaaa,mm, dd].

Convert a date as list into a string readable by the ECMWWF api.

▶ **file_lenght(fname)**

Arguments:

- **fname**, *string*, .txt file

Return the length of a file.

3 Wind_treament library

This library contains the function used for the treatment of wind data. The functions plotting the roses come from the windrose python package. Therefore, they are highly customizable providing a bit of reading their documentation, and some modifications of the code of these functions.

► **wind_rose(Angle,Intensity, place, fig, **kwargs)**

Arguments:

- **Angle**, *1D array like*, Orientation of the wind in degree.
- **Intensity**, *1D array like*, Strength of the wind
- **fig**, *matplotlib figure*, Matplotlib figure in which the wind rose is drawn. Optional. Default is None.
- **place**, *list of float*, Setting the wind rose axe position in the figure. Of the form [left, bottom, width, height], in figure coordinates.
- ****kwargs**, *multiple*, All optional arguments that can be passed to customize the wind rose plot, as 'cmap', 'bins', etc . Detail can be found in the windrose project website by downloading the manual.

Write a wind rose from a wind time series using wind orientation and strength.

► **flux_rose(Angle,PdfQ, withaxe, place, fig, color, nsector, opening)**

Arguments:

- **Angle**, *1D array like*, Angles corresponding to the sand flux distribution.
- **PdfQ**, *1D array like*, Sand flux distribution.
- ArgumentwithaxeintegerIf withaxe =0, the flux rose is drawn without any axes. Optional. Default is 1.
- **fig**, *matplotlib figure*, Matplotlib figure in which the wind rose is drawn. Optional. Default is None.
- **place**, *list of float*, Setting the wind rose axe position in the figure. Of the form [left, bottom, width, height], in figure coordinates.
- **color**, *string*, Color of the flux rose. Optional. Default is 'green'.
- **nsector**, *integer*, Number of bins in term of angle. Optional. Default is 20.
- **opening**, *float*, Space between the bins. 1 is without any space. Optional. Default is 0.6.

Write a flux rose from a wind time series using wind orientation and strength.

▶ **PDF_flux(wind_data, grain_size)**

Arguments:

- **wind_data**, *2D numpy array*, Wind time series. First column is the wind direction, second column is the wind speed.
- **grain_size**, *float*, Grain size used to calculate the sand flux distribution.

Calculate sand flux distribution from a wind time series. Infer the wind shear velocity from the law of the wall, and the threshold velocity from $u_{\rm th} = 0.08 * \sqrt{(\rho_{\rm sed} - \rho_{\rm air}) * g * d/\rho_{\rm air}}$.

4 Minimal examples

```
# @Author: gadal
# @Date:   2018-12-14T18:00:01+01:00
# @Email:  gadal@ipgp.fr
# @Last modified by:   gadal
# @Last modified time: 2018-12-17T11:29:35+01:00



import Era_Interim

######################################  Create the wanted wind data Data base :
Skeleton_Coast = Era_Interim.Wind_data()
Skeleton_Coast.name =  'Skeleton_Coast'

####################################################### Define its attributes :
Skeleton_Coast.grid_bounds = [[-23.8,14.07], [-26.3,16.7]]
Skeleton_Coast.years = [[1979,1,1], [2017,12,31]]

#Write these attributes to .txt file that can be re-loaded.
Skeleton_Coast.Write_spec('info.txt')

######################### Retrieve the wind data fron the Era Interim serveur :
#### Do not forget to split your request it's too large.
Nsplit = 3
Skeleton_Coast.Update_grib_name()
Skeleton_Coast.Getting_wind_data(Skeleton_Coast.grid_bounds, Nsplit)


################################################### Proceed the raw wind data :
```

```
Skeleton_Coast.Extract_UV(path_to_wgrib = '/home/gadal/Bin_local')
Skeleton_Coast.load_wind_data()
Skeleton_Coast.Cartesian_to_polar()

# Optional, write the wind data to separate .txt files.
Skeleton_Coast.Write_wind_data('wind_data')

###############################################  Plot the wind and flux roses :
Skeleton_Coast.Write_wind_rose('wind_rose', ext = '.pdf',  normed=True,
opening=1, edgecolor='k', nsector = 20, bins = 6)
Skeleton_Coast.Write_flux_rose('flux_rose', ext = '.pdf', withaxe = 1,
opening = 0.9)

####################### CReate the KML points to see the grid on google Earth:

Skeleton_Coast.Update_coordinates()
Skeleton_Coast.Write_coordinates()
Skeleton_Coast.Create_KMZ()
```