altimetry tools Documentation

Release 0

Renaud Dussurget

January 08, 2014

CONTENTS

CHAPTER

ONE

DESCRIPTION

py-altimetry is a python module made to easily handle and process altimetry data. It can furthermore handle a wider variety of data (in-situ or imagery data), and also has a number of tools, eg. spectral analysis, filtering data, interpolating, plotting maps, handle date vectors, handle positional data ...

This is module can be used as is, and is not aimed (nor designed) for operationnal constraints.

CHAPTER

TWO

CONTENTS

2.1 Description of the modules

You can find here the documentation about all modules.

2.1.1 Contents:

altimetry.data Module

This module contains special classes and tools to handle oceanographic data.

- ***Data objects:***
  alti_data class - handles altimetry data:
  hydro_data class - handles oceanographic (in-situ or remotely sensed) data:

Data objects:

alti_data class - handles altimetry data:

class `altimetry.data.`**`alti_data`**(*file_pattern*, *time_range=None*, *output_is_dict=True*, ***kwargs*)

An `altimetry.data.hydro_data` object dedicated to handling along-track altimetry data.

Example To load different sets of data, try these :

- Concatenate a set of **AVISO's MFSTEP NRT along-track data** and plot a 2D hovmoller matrix:

```
#Define parameters
trange_str = ['24/09/2012','05/09/2013']
trange,tdatetime=AT.cnes_convert(trange_str) #convert time range
alti_pattern = '/path/to/nrt/mfstep/nrt_mfstep_j2_sla_vfec_*.nc'

#Load data
alti=alti_data(alti_pattern,verbose=verbose,datatype='DT',time_range=trange,slaext=True)

#2D reordering of the data
alti.reorder()

#Plot results
pcolormesh(data.lat,data.cycle,data.sla); show() #plot the hovmoller
```

- Loads a set of **PISTACH L3 5Hz** files and create a new SLA variable and slice the object using a given time range :

```
#Load data
alti_pattern = '/path/to/data/PISTACH_L3_Product_NWMED_MLE4_tr*_5hz.nc'
alti=alti_data(alti_pattern,limit=limit,verbose=verbose)

alti.create_Variable('sla',                              #new variable name
                     alti.ssh_mss_filtree_21pts,         #data
                     {'time':alti._dimensions['time']}, #set dimensions
                     extend=False)                       #extend option

#get daily updates of the object
for date in xrange(21300,21320):

    #get a deep copy of the object, not to erase the whole dataset
    subset=alti.copy(deep=True)

    #update the object with the proper slice
    fg=subset.slice('date', [date,date+1])
    subset.update(fg)

    do_something(subset)
```

- Loads a set of **PISTACH hydro** files :

```
data=AD.alti_data('%s/*_2PTP*_*.nc' % RepData,verbose=opts.verbose,datatype='RAW',remove_
```

- Load any **NetCDF file** using `altimetry.tools.nctools.nc` :

```
data=AD.alti_data(fout,verbose=opts.verbose,datatype='RAW',transpose=False)
```

__init__ (*file_pattern*, *time_range=None*, *output_is_dict=True*, ***kwargs*)
returns a dataset from a single file or multiple concatenated files. cf. `altimetry.data.hydro_data` for further informations

Parameters

- **time_range** – get start dates from file names (cf. notes on file names when using this option)
- **kwargs** – additionnal keywords to be passed to `altimetry.data.hydro_data.__init__()`

Note: Naming convetion should respect AVISO formatting

•start dates should be the 3rd field from the end

•satellite name should be the 3rd from the start

•eg. my_file_sat_20010101_20010108_20010109.nc

---

cycle_list (**args*)
return the list of cycles contained if the dataset

pass_time()
Compute the central time for each passes.

Note: this must be called AFTER having called altimetry.data.alti_data.reorder() as it looks for the CYCLE and RECORD dimensions.

Note: The methodology to compute the central time is to interpolate the time along the track at missing points, and then reading the value at point N/2.

---

read(*filename*, *datatype=None*, *slaext=False*, ***kwargs*)
reader method.

Parameters

- **filename** – name of the file to load.
- **datatype** – choose between DT/NRT/PISTACH/CTOH or other formats to call the corresponding reader. If datatype is :
  - DT or NRT or PISTACH : calls altimetry.data.alti_data.read_sla() or altimetry.data.alti_data.read_slaext()
  - CTOH : calls altimetry.data.alti_data.read_CTOH()
  - else : calls altimetry.data.alti_data.read_nc(), based on altimetry.tools.nctools.nc object.
- **slaext** – force using altimetry.data.alti_data.read_slaext()

Note: This method is call from altimetry.data.hydro_data.__init__() and returns a data structure to be handled by altimetry.data.hydro_data.update_dataset()

---

read_CTOH(*filename*, *params=None*, *force=False*, *timerange=None*, *datatype=None*, ***kwargs*)
Read AVISO Along-Track SLA regional products

Return outStr Output data structure containing all recorded parameters as specificied by NetCDF file PARAMETER list.

Author Renaud Dussurget

read_nc(*filename*, ***kwargs*)
data reader based on altimetry.tools.nctools.nc object.

Note: THIS can be VERY powerful!

---

read_sla(*filename*, *params=None*, *force=False*, *timerange=None*, *datatype=None*, ***kwargs*)
Read AVISO Along-Track products

Return outStr Output data structure containing all recorded parameters as specificied by NetCDF file PARAMETER list.

Author Renaud Dussurget

read_slaext(*filename*, *params=None*, *force=False*, *timerange=None*, *datatype=None*, ***kwargs*)
Read AVISO Along-Track SLAEXT regional products

Return outStr Output data structure containing all recorded parameters as specificied by NetCDF file PARAMETER list.

Author Renaud Dussurget

reorder(**args*, ***kwargs*)
Reorders data vectors in 2D (ie. with dimensions (CYCLE,RECORD)). This is useful to get a hovmoller-type matrix of each variable.

Example To plot a hovmoller for a given variable, do

```
.. code-block:: pyhton
```

data=alti_data('/my/dir/my_files_pattern*.nc') #concatenate the files data.reorder() #reorder data pcolormesh(data.lat,data.cycle,data.sla); show() #plot the hovmoller

---

Note: This only works for data reprojected along a nominal track.

---

set_sats()
set satellite name using (cf. notes on file names in *altimetry.data.alti_data.__init__*)

track_list(**args*)
return the list of tracks contained if the dataset

hydro_data class - handles oceanographic (in-situ or remotely sensed) data:

class altimetry.data.**hydro_data**(*file_pattern*, *limit=None*, *verbose=1*, *round=True*, *zero_2pi=True*, *output_is_dict=True*, *flatten=False*, ***kwargs*)
A base object dedicated to handle oceanographic data (in-situ or remote sensing) with upper level processing methods.

Note: This object SHOULD NOT be called directly but through a subclass heritating of it (eg. altimetry.data.alti_data)

Error(*ErrorMsg*)
raises an exception

__init__(*file_pattern*, *limit=None*, *verbose=1*, *round=True*, *zero_2pi=True*, *output_is_dict=True*, *flatten=False*, ***kwargs*)
Returns the object filled with the data loaded from a single file or a concatenated set of files

Parameters

- **file_pattern** – a pattern of files to be globbed (glob.glob()) or a list of file names.
- **limit** – the limits of the domain to handle ([latmin,lonmin,latmax,lonmax]).
- **verbose** – verbosity level on a scale of 0 (silent) to 4 (max verobsity)
- **round** – round limits (cf. altimetry.tools.in_limits())
- **zero_2pi** – limits goes from 0 to 360 degrees (not -180/180).
- **output_is_dict** – data structures are dictionnaries (eg. my_hydro_data.variable['data']). If false uses an object with attributes (eg. my_hydro_data.variable.data).

Note: This methodes init all the attributes, then loads the data from files (altimetry.data.hydro_data.read()) and appends it to the object (altimetry.data.hydro_data.update_dataset()) before checking its content (altimetry.data.hydro_data.check_variables()).

Note: The method `altimetry.data.hydro_data.read()` MUST be defined (typically by overloading it). This method must return a data structure.

__weakref__
list of weak references to the object (if defined)

check_variables()
Forces variables to respect dimensions

contour_transect(*x*, *z*, *var*, *xrange=None*, *zrange=None*, *xstep=1*, *zstep=10*, *vmin=None*, *vmax=None*, *marker='.k'*, ***kwargs*)
shows a 2d space-depth section by interpolating the data along the section.

Note: This method interpolates using `scipy.interpolate.griddata()` and plots using `matplotlib.pyplot.meshcolorgrid()`

---

copy(**args*, ***kwargs*)
Returns a copy of the current data object

Parameters

- **flag** – if an argument is provided, this returns an updated copy of current object (ie. equivalent to obj.copy();obj.update(flag)), optimising the memory (
- **deep** (*True*) – deep copies the object (object data will be copied as well).

count = None
number of files loaded

create_Dim(*name*, *value*)
Adds a dimension to class.

Parameters

- **name** – dimension name
- **value** – dimension value

create_Variable(*name*, *value*, *dimensions*, *toCreate=None*, *createDim=None*, *extend=True*)
create_Variable : This function adds data to `altimetry.data.hydro_data`

Parameters

- **name** – name of the parameter to create
- **value** – values associated to the variable. Must be a numpy masked_array or a data structure.
- **dimensions** – dimensional structure (cf. notes).

Note: altimetry tools package handles the NetCDF data using specific structures.

NetCDF data is structured this way:

```
NetCDF_data = {'_dimensions':dimension_structure,  #File dimensions  (COMPULSORY)
               '_attributes':attribute_structure,  #Global attributes
               'dimension_1':data_structure,       #Data associated to the dimensions. (COMP
               ...,
               'variable_1':data_structure,        #Variables
```

```
                ...
               }
```

In standard NetCDF files, dimensions are always associated to a variable. If it is not the case, an array of indices the length of the dimension is generated and a warning is issued.

Moreover, dimensions MUST be defined to be accepted by `altimetry.tools.nctools.nc` (empty NetCDF files would fail).

•a dimensional structure should be of the form :

```
dimension_structure = {'_ndims':N,         #Attribute setting the number of dimensions.

                       'dims':{'dim_A':A, #Structure containing the name
                               'dim_B':B, #of the dimensions and their size.
                               ...,
                               'dim_N':N
                              }
                      }
```

•an attribute structure is a very simple structure containing the attribute names and values:

```
data_structure = {'attribute_1':attribute_1,
                  ...,
                  'attribute_N':attribute_N}
```

•a data structure should be of the form :

```
data_structure = {'_dimensions':dimension_structure, #dimensions of hte variable (COMPULSORY
                  'data':data,                       #data associated to the variable (COMPU

                  'long_name':long_name,             #Variable attributes
                  'units':units,
                  ...
                 }
```

DATA and _DIMENSIONS fields are compulsory. Other fields are optional and will be treated as attributes.

Furthermore, code will have a special look at **scale**, **scale_factor** and **add_offset** while reading and writing data and to **_FillValue** and missing_value while reading (_FillValue being automatically filled by `NetCDF4.Dataset` when writing)

---

delete_Variable (*name*)
pops a variable from class and delete it from parameter list

Parameters **name** – name of the parameter to delete

dim_list = None
array containing the dimensions of each parameter

dirname = None
Directory name of the file pattern being globbed (`glob.glob()`). Defaulted to current directory

extension (*flag=None*, *round=True*)
returns the limits of the dataset.

Parameters

- **flag** – an indexation flag array

- **round** – round the limits to the south-west and north-east.

fileid = None
array of file IDs

filelist = None
list of files being loaded

filelist_count = None
number of counted values by files

get (*name*)
retunrs a variable

get_currentDim ()
returns the current dimensions of the object

get_file (*pattern*)
returns a flag array of the data loaded from a given file pattern

Parameters **pattern** – pattern to match in the file list.

get_object_stats ()
get some statistics about the whole dataset.

get_platform_stats (*id*)
get statistics based on *altimetry.data.hydro_data.id*

get_stats (*flag*)
get some statistics about a part of the dataset

in_limits (*limit=None*)
wrapper to `altimetry.tools.in_limits()` based on dataset limits.

limit = None
limits of the domain : [latmin,lonmin,latmax,lonmax] (default = [-90.,0.,90.,360.])/

Note: limits are automatically reset using `altimetry.tools.recale_limits()`

---

map (*flag=None*, *fname=None*, *zoom=False*, *pmap=None*, *show=True*, ***kwargs*)
display (or not) a map based on a `altimetry.tools.plot_map` object.

Parameters **show** – set to False not to show (and neither apply `altimetry.tools.plot_map.setup_map()`)

Note: This function creates a `altimetry.tools.plot_map` instance, plot a partion of the dataset using `altimetry.data.hydro_data.plot_track()` and displays it if asked to.

---

message (*MSG_LEVEL*, *str*)
print function wrapper. Print a message depending on the verbose level

Parameters **MSG_LEVEL** (*{in}{required}{type=int}*) – level of the message to be compared with self.verbose

Example To write a message

```
self.message(0,'This message will be shown for any verbose level')
```

ncstruct (***kwargs*)
returns a data structure (dict) of the dataset.

Parameters **params** – Add this keyword to provide a list of variables to export. Default : all variables contained is self.par_list

par_list = None
array of parameters

platform_summary(*id*, *col='.k'*)
outputs a summary of the statistics for a given platform

plot_track(*pmap*, *flag=None*, *col='.k'*, *endpoint='*r'*, *endpoint_size=None*, *title=None*, *fontsize=8*, *textcolor='b'*, *ms=5*, *linewidth=1*, ***kwargs*)
plot trajectories based on platform IDs

Parameters

- **pmap** – a `altimetry.tools.plot_map` instance
- **col** – color to be used along the trajectory. If this is an array of values, calls `altimetry.tools.plot_map.scatter()` instead of `altimetry.tools.plot_map.plot()`

Note: This method loops on data IDs. Then it calls `altimetry.tools.plot_map.plot()` or `altimetry.tools.plot_map.scatter()` to plot the trajectory and then labels the trajectory using `altimetry.tools.plot_map.text()`

---

plot_track_old(**args*, ***kwargs*)
plot a surface map of sampling track

> **Warning:** DEPRECATED method!

plot_transect(*x*, *z*, *var*, *xrange=None*, *zrange=None*, *vmin=None*, *vmax=None*, *xstep=1*, *zstep=10*, *s=10*, *edgecolor='none'*, ***kwargs*)
shows a 2d space-depth section plotting point (using `altimetry.tools.plot_map.scatter()`)

Example plot a temperature section along a glider transect

pop(**args*, ***kwargs*)
This is a wrapper to `altimetry.data.hydro_data.delete_Variable()`

push_nc(**args*, ***kwargs*)
append a data structure to an exisiting netcdf file

read_ArgoNC(*filename*, *params=None*, *force=False*, *dephrange=None*, *timerange=None*, ***kwargs*)
An Argo network NetCDF reader

Return outStr Output data stricture (dict) containing all recorded parameters as specificied by NetCDF file PARAMETER list.

Author Renaud Dussurget

size = None
length of the dataset

slice(*param*, *range*, *surf=False*)
get a flag for indexing based on values (ange of fixed values).

Parameters

- **param** – variable name
- **range** – numpy array defining the range of the values. If size(range) == 2 :

- flag is computed between min and max values of range
- flag is computed based on equality to range value.

summary (*all=False*, *fig=None*, *col='.k'*, *legend=None*, ***kwargs*)
outputs a summary of the whole current dataset

time_range (*flag=None*)
time range of the current dataset

Parameters **flag** – use a flag array to know the time range of an indexed slice of the object

time_slice (*timerange*, *surf=False*)
slice object given a time range

Parameters **timerange** – rime range to be used.

update (**args*, ***kwargs*)
Wrapper to `altimetry.data.hydro_data.update_with_slice()`.

update_Dim (*name*, *value*)
update a dimension by appending the number of added elements to the dimensions

```
<updated dimension> = <old dimension> + <number of added elements along this dimension>
```

update_dataset (*dataStr*, *flatten=False*)
update class with a data structure.

Parameters **flatten** – use this to automatically flatten variables (squeeze dimensions)

update_fid_list (*filename*, *N*)
update file indices attribute *altimetry.data.hydro_data.fileid*

update_with_slice (*flag*)
update object with a given time slice flag

Parameters **array) flag** ((*boolean*) – a flag for indexing data along the ‘’time” dimension

updated_copy (*flag*, *deep=True*)
Returns a sliced (updated) copy of current data object

Summary This has the same effect as *obj.copy();obj.update(flag)* but is much less memory consumming.

Note: TypeError could arise if some object attributes are setted outside the `__init__()` function (eg. for data objects derived from `hydro_data`). If this is the case, initialise these attributes within their respective `__init__()`.

---

verbose = None
verbosity level on a scale of 0 (silent) to 4 (max verbosity)

warning (*MSG_LEVEL*, *str*)
Wrapper to `warning.warn()`. Returns a warning when verbose level is not 0.

Parameters **MSG_LEVEL** – level of the message to be compared with self.verbose

Example To issued a warning

```
self.warning(1,'Warning being issued)
```

write_nc (*filename*, *clobber=False*, ***kwargs*)
write a NetCDF file from current dataset

Parameters **kwargs** – additional arguments are passed to `altimetry.tools.nctools.nc.write()`

altimetry.tools Module

This module contain a number of tools for processing the data.

These are :

Spectral analysis

The `altimetry.tools.spectrum` module contains tools dedicated to spectral analysis.

About spectral analysis Spectral analysis of along-track data is a common thing. There are 2 main steps when computing a spectrum:

- **preprocess the data**

  It consists in detecting gaps, interpolating over short gaps and rejecting longer gaps, subsampling the data into subsegments of valid data of a given length.

  This step is performed using `altimetry.tools.spectrum.preprocess()`

- **compute the spectrum**

  This step is made through a transform of the signal to the spectral domain (eg. FFT). Then frequency, energy and power spectral densities are computed and averaged. It is also possible to use **spectral tapers** to lower the noise of the spectrum.

  This step is performed using `altimetry.tools.spectrum.spectral_analysis()` (and `altimetry.tools.spectrum.get_spec()` at lower level)

Notes on spectral tapering Tapering and padding are mathematical manipulations sometimes performed on the time series before periodogram analysis to improve the statistical properties of the spectral estimates or to speed up the computations.

Tapering can be applied:

- to reduce the noise level by oversampling the data in overlapping subsegments (eg. when we don't have enough samples)
- to better localise spectral peaks and changes in the spectral slope.

However, you should be aware that:

- tapering may induce a loss of overall energy, resulting the tapered spectrum to be under (though less noisy) the original spectrum.
- oversampling the data will result in removing a part of the lower frequencies because of the shorter subsegments.

`altimetry.tools.spectrum.preprocess()` allows using tapers through its `tapering` keyword.

Warning: though it is taken into account in `altimetry.tools.spectrum.spectral_analysis()`, energy loss caused by the tapering may not be properly resolved.
It may be therefore necessary to correct this loss by multiplying the tapered spectrum by the ratio of energies of both spectra $\frac{E_{original}}{E_{tapered}}$

Notes on AR spectrum (auto-regression methods) AR (auto-regressive methods) can be used to model a spectrum from the signal.

Such method, as the Yule-Walker equations, can be used to model the spectrum, and therefore:

- clean the spectrum (by having an auto-regression approach)
- compute the energy (or power) at any frequency (ie. not being dependant on the length of input array).

This approach is made possible through the `ARspec` keyword of `altimetry.tools.spectrum.spectral_analysis()` (itself calling `altimetry.tools.spectrum.yule_walker_regression()`).

List of useful functions

- `altimetry.tools.spectrum.spectral_analysis()` : Compute the average spectrum over a set of data
- `altimetry.tools.spectrum.preprocess()` : Preprocess the data to be admissible to spectral analysis
- `altimetry.tools.spectrum.get_slope()` : Compute the spectral slope
- `altimetry.tools.spectrum.optimal_AR_spectrum()` : Get the order of the optimal AR spectrum

Functions

`altimetry.tools.spectrum.`**`spectral_analysis`**(*dx*, *Ain*, *tapering=None*, *overlap=None*, *wsize=None*, *alpha=3.0*, *detrend=False*, *normalise=False*, *integration=True*, *average=True*, *ARspec=None*)

Spectral_Analysis : This function performs a spatial spectral analysis with different options on a time series of SLA profiles.

Parameters

- **dx** – sampling distance
- **Ain** – 2D table of sla data with time along 2nd axis (NXxNT with NX the spatial length and NT the time length)
- **tapering** – apply tapering to the data
  - If this keyword is of type bool : apply hamming window.
  - If this keyword is a string : apply a hamming ('hamm'), hann ('hann'), kaiser-bessel ('kaiser'), kaiser-bessel ('blackman') or no ('none') tapering function.
  - If this keyword is an `numpy.array` object : apply this array as taper.
- **overlap** – overlap coefficient of the windows (0.75 means 75% overlap).
- **wsize** – size of the sub-segments.
- **normalise** – If True, normalise the spectrum by its overall energy content.
- **detrend** – If True, removes a linear trend to the segmented signal (if tapered) or to the whole signal (if not tapered).
- **integration** – If True, integrate the spectrum between 2 frequencies.
- **sla** – data

Returns

a spectrum structure

```
{'esd':esd,        #Energy Spectral Density
 'psd':psd,        #Power Spectral Density
 'fq':fq,          #frequency
 'p':p,            #wavelength
 'params':params}  #tapering parameters.
```

Author Renaud DUSSURGET (RD) - LER/PAC, Ifremer

Change Created by RD, December 2012

altimetry.tools.spectrum.**preprocess**(*lat*, *lon*, *sla*, *N_min=None*, *per_min=15.0*, *max_gap=None*, *leave_gaps=False*, *remove_edges=True*, *interp_over_continents=False*, *truncate_if_continents=True*, *discard_continental_gaps=True*, *flag_interp=False*, *return_lonlat=False*, *return_interpolated=False*, *last=True*, *mid=None*, *first=None*, *verbose=1*)

Preprocessing of the SLA data ::

- **process positions :**
  - interpolate over gaps
  - find continents (extend the positions over continents to get the discontinuity)
  - find track edges
  - find gap lengths
- **clean SLA data::**
  - Remove gaps greater than maximum allowed length over which interpolate is OK.
  - Remove time steps with not enough coverage
  - get sub-segments of valid data of a given length

Parameters

- **lon** – longitude
- **lat** – longitude
- **sla** – data
- **N_min** – Length of subsegments (cf `altimetry.tools.spectrum.get_segments()`)
- **per_min** – Minimum percentage of valid data to allow.
- **max_gap** – Maximum gap length to interpolate over (interpolation is done 1st, THEN long gaps are eliminated)
- **leave_gaps** – Leave gaps (equivalent to setting max_gap to number of points in track).
- **remove_edges** – discard data at track edges.
- **truncate_if_continents** – Force truncating data if a continent is found within a segment of data.
- **last** – Get segments of data sticked to the last point in track (cf `altimetry.tools.spectrum.get_segments()`)

- **first** – Get segments of data sticked to the first point in track (cf `altimetry.tools.spectrum.get_segments()`)
- **mid** – Get segments of data sticked to the middle point in track (cf `altimetry.tools.spectrum.get_segments()`)

`altimetry.tools.spectrum.`**`get_kx`**(*N*, *dx*)

GET_KX :summary: Returns the frequencies to be used with FFT analysis

Parameters

- **N** – number of samples in data
- **dx** – sampling step

Returns Returns * k: frequency * L: length * imx: index of maximum frequency (for separating positive and negative frequencies)

Author Renaud DUSSURGET (RD) - LER/PAC, Ifremer

Change Created by RD, July 2012

`altimetry.tools.spectrum.`**`get_spec`**(*dx*, *Vin*, *verbose=False*, *gain=1.0*, *integration=True*)

GET_SPEC :summary: Returns the spectrum of a regularly sampled dataset

Parameters

- **dq** – sampling interval (1D)
- **V** – data to analyse (1D).

Note NaN can not be used.

Returns

- psd: Power Spectral Density
- esd: Energy Spectral Density
- fq: frequency
- p: wavelength (period)

Author Renaud DUSSURGET (RD) - LER/PAC, Ifremer

Change Created by RD, July 2012. Changes * 29/08/2012 : Changed the computation of frequencies and the spectral integration (spectrum is averaged at mid-width frequencies) * 30/11/2012 : Outstanding changes : corrected the spectral integration for computing psd and corrected the normalisation

`altimetry.tools.spectrum.`**`get_segment`**(*sla*, *N*, *last=True*, *mid=None*, *first=None*, *remove_edges=True*, *truncate_if_continents=True*)

Intelligent segmentation of data.

Parameters

- **remove_edges** – discard data at track edges.
- **truncate_if_continents** – Force truncating data if a continent is found within a segment of data.
- **last** – Get segments of data sticked to the last point in track
- **first** – Get segments of data sticked to the first point in track
- **mid** – Get segments of data sticked to the middle point in track

altimetry.tools.spectrum.**get_slope**(*fq*, *spec*, *degree=1*, *frange=None*, *threshold=0.0*)

GET_SLOPE :summary: This function returns the spectral slope of a spectrum using a least-square regression

Parameters

- **fq** – frequency
- **spec** – spectrum data
- **degree** – Degree of the least-square regression model

Returns

- slope : spectral slope (or model coefficients for a higher order model)
- intercept : Energy at unit frequency (1 cpkm)

Author Renaud DUSSURGET (RD) - LER/PAC, Ifremer

Change Created by RD, August 2012

altimetry.tools.spectrum.**yule_walker**(*acf*, *orden*)

Program to solve Yule-Walker equations for AutoRegressive Models

Author XAVI LLORT (llort(at)grahi.upc.edu)

Created MAY 2007

Changes adapted to python by R.Dussurget

Parameters

- **acf** – AutoCorrelation Function
- **orden** – Order of the AutoRegressive Model

Returns

- parameters : Parameters
- sigma_e : Standard deviation of the noise term

altimetry.tools.spectrum.**yule_walker_regression**(*dx*, *Y*, *deg*, *res=None*)

Parameters

- **X** – time vector (disabled)
- **Y** – stationary time series
- **deg** – AR model degree

Returns

- a : Yule Walker parameters
- sig : Standard deviation of the noise term
- aicc : corrected Akaike Information Criterion
- gamma : Autocorrelation function
- ar : Fitted function
- argamma : Fitted autocorrelation function
- arspec : Fitted spectral model
- F : Relative frequency

Note: To know more about yule-walker and autoregressive methods, see

•Example of AR(p) model auto-regression using yule-walker equations

•Other notes on the autoregressive method

Example IDL example :

```
#Define an n-element vector of time-series samples
X = [6.63, 6.59, 6.46, 6.49, 6.45, 6.41, 6.38, 6.26, 6.09, 5.99, $
    5.92, 5.93, 5.83, 5.82, 5.95, 5.91, 5.81, 5.64, 5.51, 5.31, $
    5.36, 5.17, 5.07, 4.97, 5.00, 5.01, 4.85, 4.79, 4.73, 4.76]

#Compute auto_correlation function
acorr=A_CORRELATE(X,INDGEN(30))

#Solve YW equation to get auto-regression coefficients for AR(2) model
YULE_WALKER, acorr, 2, a, sig

#Process auto-regression model
ar=DBLARR(28)
FOR i = 2, 29 DO ar[i-2] = SQRT(a[0]*X[i-1]*X[i] + a[1]*x[i-2]*x[i]+sig*x[i])

#Compute spectrum
spec=spectrogram(TRANSPOSE(X), INDGEN(N), WSIZE=N, OVERLAY=1.0, DISPLAY_IMAGE=0)

#Compute AR(2) model spectrum
ar2=spectrogram(TRANSPOSE(ar), INDGEN(28), WSIZE=28, OVERLAY=1.0, DISPLAY_IMAGE=0)
```

altimetry.tools.spectrum.**optimal_AR_spectrum**(*dx*, *Y*, *ndegrees=None*, *return_min=True*)
Get the optimal order AR spectrum by minimizing the BIC.

NetCDF tools

The altimetry.tools.nctools module contains tools dedicated to easily handle NetCDF data.

- *An easy to use wrapper to NetCDF4 package - altimetry.tools.nctools.nc*
- *Addionnal function*

An easy to use wrapper to NetCDF4 package - altimetry.tools.nctools.nc

class altimetry.tools.nctools.**nc**(*limit=[-90.0, 0.0, 90.0, 360.0], verbose=0, zero_2pi=False, transpose=False, use_local_dims=False, **kwargs*)
A class for easy-handling of NetCDF data based on NetCDF4 package.

Example To load different sets of data, try these :

- Simply load a NetCDF file
  - The file has standard dimensions (eg. called longitude & latitude)

```
ncr=nc()
data=ncr.read(file)

lon=data.lon
```

```
lat=data.lat
Z=data.Z
```

- We do not want to match for standard dimension names and keep original names

```
ncr=nc()
data=ncr.read(file,use_local_dims=True)

lon=data.longitude
lat=data.latitude
Z=data.Z
```

- We extract a region and depth range between 2 dates:
  - We extract between 30-40°N & 15-20°E (limit).
  - We extract between 100 & 200 m deep (depth).
  - We get data from 2010/01/01 to 2010/01/07 (time).
  - File has standard dimensions called longitude, latitude, level and time

```
ncr=nc()
limit=[30,15,40,20]
depth=[100,200]
time=[21915,21921]

data=ncr.read(file,
              limit=limit,
              timerange=time,
              depthrange=depth)

lon=data.lon
lat=data.lat
dep=data.depth
dat=data.time
Z=data.Z
```

- More sophisticated example using a file containing bathymetetry data
  - Load a file and extract a regions and subsample it to a lower resolution
    - The file has dimensions NbLongitudes & NbLatitudes.
    - We extract between 30-40°N & 15-20°E (limit).
    - We subsample every 3 points (stride).

```
limit=[30,15,40,20]
stride = (3,)
ncr=nc(use_local_dims=True)
bathy=ncr.load(file,
               NbLongitudes=(limit[1],limit[3])+stride,
               NbLatitudes=(limit[0],limit[2])+stride)
```

  - Then we save the data to another file (output).

```
#save data
bathy.write_nc(output)
```

– We update the **history** global attribute of data structure

```
#Get attribute structure
attrStr=bathy.get('_attributes',{})

#Get arguments called from the shell
cmd=[os.path.basename(sys.argv[0])]
for a in argv : cmd.append(a)

#update attribute stucture (pop history and concatenate with current commands=.
attrStr.update({'history':attrStr.pop('history','')+' '.join(cmd)+'\n'})

#update NetCDF data structure
bathy.update({'_attributes':attrStr})

#save data
bathy.write_nc(output)
```

– We now want to flag all values from variable Z above 0 by setting them to fill_value and append this modified variable to the output file

```
#load variable
Z = bathy.Z

#flag variable
Z.mask[Z >= 0] = False

#update attributes
Z['_attributes']['long_name'] = 'flagged bathymetry'

#append modified bathymetry to a variable named Z_2 in output file.
bathy.push(output,'Z2',Z)
```

attributes (*filename*, ***kwargs*)
Get attributes of a NetCDF file

:return {type:dict} outStr: Attribute structure. :author: Renaud Dussurget

count = None
number of files loaded

fileid = None
array of file IDs

limit = None
limits of the domain : [latmin,lonmin,latmax,lonmax] (default = [-90.,0.,90.,360.])

Note: limits are automatically reset using `altimetry.tools.recale_limits()`

---

load (*filename*, *params=None*, *force=False*, *depthrange=None*, *timerange=None*, *output_is_dict=True*, ***kwargs*)
NetCDF data loader

Parameters

- **filename** – file name
- **params** – a list of variables to load (default : load ALL variables).
- **depthrange** – if a depth dimension is found, subset along this dimension.
- **timerange** – if a time dimension is found, subset along this dimension.

Note: using `altimetry.tools.nctools.limit` allows subsetting to a given region.

Parameters **kwargs** – additional arguments for subsetting along given dimensions.

Note: You can index along any dimension by providing the name of the dimensions to subsample along. Values associated to the provided keywords should be a length 2 or 3 tuple (min,max,<step>) (cf. `altimetry.data.nctools.load_ncVar()`).

Parameters **output_is_dict** – data structures are dictionnaries (eg. my_hydro_data.variable['data']). If false uses an object with attributes (eg. my_hydro_data.variable.data).

:return {type:dict} outStr: Output data structure containing all recorded parameters as specificied by NetCDF file PARAMETER list.

Author Renaud Dussurget

message(*MSG_LEVEL*, *str*)
print function wrapper. Print a message depending on the verbose level

Parameters **MSG_LEVEL** (*{in}{required}{type=int}*) – level of the message to be compared with self.verbose

Example display a message

```
self.log(0,'This message will be shown for any verbose level')
```

Author Renaud DUSSURGET (RD), LER PAC/IFREMER

Change Added a case for variables with missing dimensions

push(**args*, ***kwargs*)
append a variable from a given data structure to the existing dataset.

Parameters

- **file** (*optional*) –
- **name** – variable name
- **value** – data
- **start** – broadcast the data to a portion of the dataset. starting index.
- **counts** – broadcast the data to a portion of the dataset. number of counts.
- **stride** – broadcast the data to a portion of the dataset. stepping along dimension.

read(*file_pattern*, ***kwargs*)
Read data from a NetCDF file

Parameters

- **file_pattern** – a file pattern to be globbed (`glob.glob()`) or a file list.
- **kwargs** – additional keywords to be passed to `altimetry.tools.nctools.nc.load()` (eg. extracting a subset of the file)

`size` = None
length of the dataset

`use_local_dims` = None
this option prevent from trying to detect standard CF dimensions such longitude, latitude, time in the file and keep the original dimensions of the file

---

Note: Set this option to True when file is not standard (eg. not following CF conventions).

Note: Normal behaviour is to match dimensions (ie. a dimension and the associated variable of the same name) with specific names. Resulting variables associated with these dimensions will be called : * lon (longitudes) : matches dimensions starting with 'lon' * lat (latitudes) : matches dimensions starting with 'lat' * time (time) : matches dimensions starting with 'date' or 'time' * depth (date) : matches dimensions starting with 'dep' or 'lev'

`verbose` = None
verbosity level on a scale of 0 (silent) to 4 (max verbosity)

`write` (*data*, *outfile*, *clobber=False*, *format='NETCDF4'*)
Write a netCDF file using a data structure.

Parameters

- **data** – data structure
- **outfile** – output file
- **clobber** – erase file if it already exists
- **format** – NetCDF file format.

Note: the data structure requires a "_dimensions" field (dimension structure)

Addionnal function

`altimetry.tools.nctools.`**`load_ncVar`** (*varName*, *nc=None*, ***kwargs*)
Loads a variable from the NetCDF file and saves it as a data structure.

Parameters **varName** – variable name

Keywords **kwargs** additional keyword arguments for slicing the dataset. Keywords should be named the name of the dimensions to subsample along and associated value should be a length 2 or 3 tuple (min,max,<step>).

2.2 Some examples to illustrate py-altimetry functionalities

2.2.1 Handle data

altimetry tools have a number of classes and function to handle data, mainly based on :mod:NetCDF4 package.

You can find examples of :

- *Loading NetCDF files*
- *Handle altimetry data using altimetry.data.alti_data object*

Note: Handling data may require knowlegde about how the NetCDF data is structured. You should therefore have a look at:

- *informations about data structures*

2.2.2 Others

- *Spectral analysis tools for altimetry data*

2.3 Install, configure & modify

Not available yet!

2.3.1 Contents:

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

a

INDEX

R

S

T

U

V

W

Y