oceanwaves Documentation

Release 0.1

Bas Hoonhout

Jul 06, 2018

Contents

Oceanwaves is a Python package that provides a generic data storage object for ocean wave data (time series and/or spectral). The package provides a series of I/O functions to use with various file formats, like SWAN, Waverider buoys and WaveDroid.

Oceanwaves is based on the xarray DataSet object, but defines special variables for time, location, frequency and direction. Many of its functionalities are obtained from the pyswan toolbox, originally developed by Gerben de Boer, and the swantools toolbox, originally developed by Caio Eadi Stringari.

The source code of the oceanwaves package can be found at https://github.com/openearth/oceanwaves-python.

Usage examples can be found in this notebook https://github.com/openearth/oceanwaves-python/blob/master/notebooks/oceanwaves.ipynb.

CHAPTER 1

Features

The OceanWaves object supports various standard conversions, like:

- From significant wave height to spectral
- From omnidirectional to directional
- From directional to omnidirectional
- From spectral to significant wave height
- From spectral to spectral wave period
- From spectral to peak wave period
- From directional to peak wave direction
- From degrees to radians
- Automated unit conversion

The OceanWaves object supports various standard plotting methods, like:

- Polar subplots for directional data on multiple locations/times
- Polar subplots on a map
- Plots supported by xarray.Dataset and Seaborn

The OceanWaves object can be instantiated from:

- Raw data
- SWaN 1D/2D spectral or table files
- An xarray.Dataset object
- Another OceanWaves object

The OceanWaves object can be written to:

- SWaN 1D/2D spectral files
- Output supported by xarray.Dataset (e.g. netcdf)

CHAPTER 2

Contents

2.1 Source code documentation

2.1.1 OceanWaves

class oceanwaves.**OceanWaves**(*time=None*, *location=None*, *frequency=None*, *direction=None*, *energy=None*, *spreading=None*, *time_units='s'*, *location_units='m'*, *frequency_units='Hz'*, *direction_units='deg'*, *energy_units='m^2/Hz'*, *spreading_units='deg'*, *time_var='time'*, *location_var='location'*, *frequency_var='frequency'*, *direction_var='direction'*, *energy_var='energy'*, *spreading_var='spreading'*, *frequency_convention='absolute'*, *direction_convention='nautical'*, *spreading_convention='cosine'*, *spectral=True*, *directional=True*, *attrs={}*, *crs=None*, ***kwargs*)

Class to store (spectral) data of ocean waves

The class is a specific variant of an xarray.Dataset that defines any selection of the following dimensions: time, location, frequency and direction. The dependent variable is some form of wave energy.

The class has methods to compute spectral moments and derived wave properties, like the significant wave height and various spectral wave periods. In addition the peak wave period and peak wave directions can be computed using dedicated methods.

The class automatically converts locations from a local coordinate reference system to lat/lon coordinates, if the local coordinate reference system is specified.

The class interpretates combined variable units and simplifies the result to practical entities.

The class provides two plotting routines: 1) plotting of spectral wave data in a raster of subplots and 2) plotting of spectral wabe data on a map.

The class supports all convenient properties of an xarray.Dataset, like writing to netCDF or converting to pandas.DataFrame.

TODO:

- improve plotting routines
- add phase functions to use with tides: phase estimates, phase interpolation, etc.

Hm0 (*f_min=0*, *f_max=inf*)
Compute significant wave height based on zeroth order moment

Parameters

- **f_min** (*float*) – Minimum frequency to include in moment
- **f_max** (*float*) – Maximum frequency to include in moment

Returns **H** – Significant wave height at each point in time and location in the dataset

Return type xarray.DataArray

Tm01 ()
Compute wave period based on first order moment

Returns **T** – Spectral wave period at each point in time and location in the dataset

Return type xarray.DataArray

Tm02 ()
Compute wave period based on second order moment

Returns **T** – Spectral wave period at each point in time and location in the dataset

Return type xarray.DataArray

Tp ()
Alias for *oceanwaves.OceanWaves.peak_period()*

__getitem__ (*key*)
Access variables or coordinates this dataset as a DataArray.

Indexing with a list of names will return a new Dataset object.

__init__ (*time=None*, *location=None*, *frequency=None*, *direction=None*, *energy=None*, *spreading=None*, *time_units='s'*, *location_units='m'*, *frequency_units='Hz'*, *direction_units='deg'*, *energy_units='m^2/Hz'*, *spreading_units='deg'*, *time_var='time'*, *location_var='location'*, *frequency_var='frequency'*, *direction_var='direction'*, *energy_var='energy'*, *spreading_var='spreading'*, *frequency_convention='absolute'*, *direction_convention='nautical'*, *spreading_convention='cosine'*, *spectral=True*, *directional=True*, *attrs={}*, *crs=None*, ***kwargs*)
Initialize class

Sets dimensions, converts coordinates and fills the dataset, if data is provided.

Parameters

- **time** (*iterable, optional*) – Time coordinates, each item can be a datetime object or float
- **location** (*iterable of 2-tuples, optional*) – Location coordinates, each item is a 2-tuple with x- and y-coordinates
- **frequency** (*iterable, optional*) – Frequency cooridinates
- **direction** (*iterable, optional*) – Direction coordinates
- **energy** (*matrix, optional*) – Wave energy
- **time_units** (*str, optional*) – Units of time coordinates (default: s)
- **location_units** (*str, optional*) – Units of location coordinates (default: m)

- **frequency_units** (*str, optional*) – Units of frequency coordinates (default: Hz)
- **direction_units** (*str, optional*) – Units of direction coordinates (default: deg)
- **energy_units** (*str, optional*) – Units of wave energy (default: m^2/Hz)
- **time_var** (*str, optional*) – Name of time variable (default: time)
- **location_var** (*str, optional*) – Name of location variable (default: location)
- **frequency_var** (*str, optional*) – Name of frequency variable (default: frequency)
- **direction_var** (*str, optional*) – Name of direction variable (default: direction)
- **energy_var** (*str, optional*) – Name of wave energy variable (default: energy)
- **frequency_convention** (*str, optional*) – Convention of frequency definition (default: absolute)
- **direction_convention** (*str, optional*) – Convention of direction definition (default: nautical)
- **attrs** (*dict-like, optional*) – Global attributes
- **crs** (*str, optional*) – Proj4 specification of local coordinate reference system
- **kwargs** (*dict, optional*) – Additional options passed to the xarray.Dataset initialization method

See also:

oceanwaves.OceanWaves.reinitialize()

__setitem__(*key*, *value*)
Add an array to this dataset.

If value is a *DataArray*, call its *select_vars()* method, rename it to *key* and merge the contents of the resulting dataset into this dataset.

If value is an *Variable* object (or tuple of form `(dims, data[, attrs])`), add it to this dataset as a new variable.

as_degrees()
Convert directions to degrees

as_directional(*direction*, *direction_units='deg'*, *theta_peak=None*, *s=None*, *normalize=True*)
Convert omnidirectional spectrum to a directional spectrum

Spreads total wave energy over a given set of directions according to a spreading factor `s`.

See *oceanwaves.spectral.directional_spreading()* for options.

Returns New OceanWaves object

Return type *OceanWaves*

as_omnidirectional()
Convert directional spectrum to an omnidirectional spectrum

Integrate spectral energy over the directions.

Returns New OceanWaves object

Return type *OceanWaves*

as_radians()
Convert directions to radians

as_spectral(*frequency*, *frequency_units='Hz'*, *Tp=None*, *gamma=3.3*, *sigma_low=0.07*, *sigma_high=0.09*, *shape='jonswap'*, *method='yamaguchi'*, *g=9.81*, *normalize=True*)
Convert wave energy to spectrum

Spreads total wave energy over a given set of frequencies according to the JONSWAP spectrum shape.

See *oceanwaves.spectral.jonswap()* for options.

Returns New OceanWaves object

Return type *OceanWaves*

convert_coordinates(*crs*)
Convert coordinates from local coordinate reference system to lat/lon

Parameters **crs** (*str*) – Proj4 specification of local coordinate reference system

directional_spreading()
Estimate directional spreading

Estimate directional spreading by assuming a gaussian distribution and computing the variance of the directional spreading. The directional spreading is assumed to be `3000/var`.

Notes

This estimate is inaccurate and should be improved based on the cosine model.

classmethod from_dataset(*dataset*, **args*, ***kwargs*)
Initialize class from xarray.Dataset or OceanWaves object

Parameters **dataset** (*xarray.Dataset or Oceanwaves*) – Base object

has_dimension(*dim*, *raise_error=False*)
Checks if dimension is present

Parameters

- **dim** (*str*) – Name of dimension
- **raise_error** (*bool, optional*) – Raise error if dimension is absent (default: False)

Returns Boolean indicating whether dimension is present

Return type bool

Raises `ValueError`

iterdim(*dim*)
Iterate over given dimension

Parameters **dim** (*str*) – Name of dimension

moment(*n*, *f_min=0.0*, *f_max=inf*)
Compute nth order moment of wave spectrum

Parameters

- **n** (*int*) – Order of moment
- **f_min** (*float*) – Minimum frequency to include in moment
- **f_max** (*float*) – Maximum frequency to include in moment

Returns **m** – nth order moment of the wave spectrum at each point in time and location in the dataset

Return type xarray.DataArray

peak_direction()
Compute peak wave direction

Returns **theta** – Peak wave direction at each point in time and location in the dataset

Return type xarray.DataArray

peak_period()
Compute peak wave period

Returns **T** – Peak wave period at each point in time and location in the dataset

Return type xarray.DataArray

reinitialize(***kwargs*)
Reinitializes current object with modified parameters

Gathers current object's initialization settings and updates them with the given initialization options. Then initializes a new object with the resulting option set. See for all supported options the initialization method of this class.

Parameters **kwargs** (*dict*) – Keyword/value pairs with initialization options that need to be overwritten

Returns New OceanWaves object

Return type *OceanWaves*

to_netcdf(**args*, ***kwargs*)
Write dataset contents to a netCDF file.

Parameters

- **path** (*str, Path or file-like object, optional*) – Path to which to save this dataset. File-like objects are only supported by the scipy engine. If no path is provided, this function returns the resulting netCDF file as bytes; in this case, we need to use scipy, which does not support netCDF version 4 (the default format becomes NETCDF3_64BIT).
- **mode** (*{'w', 'a'}, optional*) – Write ('w') or append ('a') mode. If mode='w', any existing file at this location will be overwritten. If mode='a', existing variables will be overwritten.
- **format** (*{'NETCDF4', 'NETCDF4_CLASSIC', 'NETCDF3_64BIT', 'NETCDF3_CLASSIC'}, optional*) – File format for the resulting netCDF file:
  - NETCDF4: Data is stored in an HDF5 file, using netCDF4 API features.
  - NETCDF4_CLASSIC: Data is stored in an HDF5 file, using only netCDF 3 compatible API features.
  - NETCDF3_64BIT: 64-bit offset version of the netCDF 3 file format, which fully supports 2+ GB files, but is only compatible with clients linked against netCDF version 3.6.0 or later.
  - NETCDF3_CLASSIC: The classic netCDF 3 file format. It does not handle 2+ GB files very well.

All formats are supported by the netCDF4-python library. scipy.io.netcdf only supports the last two formats.

The default format is NETCDF4 if you are saving a file to disk and have the netCDF4-python library available. Otherwise, xarray falls back to using scipy to write netCDF files and defaults to the NETCDF3_64BIT format (scipy does not support netCDF4).

- **group** (*str, optional*) – Path to the netCDF4 group in the given file to open (only works for format='NETCDF4'). The group(s) will be created if necessary.
- **engine** (*{'netcdf4', 'scipy', 'h5netcdf'}, optional*) – Engine to use when writing netCDF files. If not provided, the default engine is chosen based on available dependencies, with a preference for 'netcdf4' if writing to a file on disk.
- **encoding** (*dict, optional*) – Nested dictionary with variable names as keys and dictionaries of variable specific encodings as values, e.g., ''{'my_variable': {'dtype': 'int16', 'scale_factor': 0.1,

  'zlib': True}, ...}''

  The *h5netcdf* engine supports both the NetCDF4-style compression encoding parameters `{'zlib': True, 'complevel': 9}` and the h5py ones `{'compression': 'gzip', 'compression_opts': 9}`. This allows using any compression plugin installed in the HDF5 library, e.g. LZF.
- **unlimited_dims** (*sequence of str, optional*) – Dimension(s) that should be serialized as unlimited dimensions. By default, no dimensions are treated as unlimited dimensions. Note that unlimited_dims may also be set via `dataset.encoding['unlimited_dims']`.
- **compute** (*boolean*) – If true compute immediately, otherwise return a `dask.delayed.Delayed` object that can be computed later.

2.1.2 Spectral

`oceanwaves.spectral.`**directional_spreading**(*theta*, *theta_peak=0.0*, *s=20.0*, *units='deg'*, *normalize=True*)

Generate wave spreading

Parameters

- **theta** (*numpy.ndarray*) – Array of mean bin directions
- **theta_peak** (*float*) – Peak direction (default: 0)
- **s** (*float*) – Exponent in cosine law (default: 20)
- **units** (*str*) – Directional units (deg or rad, default: deg)
- **normalize** (*bool*) – Normalize resulting spectrum to unity

Returns **p_theta** – Array of directional weights

Return type numpy.ndarray

`oceanwaves.spectral.`**ensure_float**(*var*)

Auxiliary function to detect and fix dtypes, if needed

Parameters **var** (*anything*) – Array to check

Returns **var** – The same as the input but either as a float or an array of floats

Return type numpy.ndarray

oceanwaves.spectral.**jonswap**(*f*, *Hm0*, *Tp*, *gamma=3.3*, *sigma_low=0.07*, *sigma_high=0.09*, *g=9.81*, *method='yamaguchi'*, *normalize=True*)

Generate JONSWAP spectrum

Parameters

- **f** (*numpy.ndarray*) – Array of frequencies
- **Hm0** (*float, numpy.ndarray*) – Required zeroth order moment wave height
- **Tp** (*float, numpy.ndarray*) – Required peak wave period
- **gamma** (*float*) – JONSWAP peak-enhancement factor (default: 3.3)
- **sigma_low** (*float*) – Sigma value for frequencies <= `1/Tp` (default: 0.07)
- **sigma_high** (*float*) – Sigma value for frequencies > `1/Tp` (default: 0.09)
- **g** (*float*) – Gravitational constant (default: 9.81)
- **method** (*str*) – Method to compute alpha (default: yamaguchi)
- **normalize** (*bool*) – Normalize resulting spectrum to match `Hm0`

Returns **E** – Array of shape `f, Hm0.shape` with wave energy densities

Return type numpy.ndarray

2.1.3 Plotting

class oceanwaves.plot.**OceanWavesPlotMethods**(*darray*, *x=None*, *y=None*, ***kwargs*)

Inheritence class to add map plotting functionality to xarray.DataArray objects

spatial_map(*ax=None*, ***kwargs*)

Plot data on a spatial map

Creates a subplot for each location in the DataArray and positions the subplot on a map. Connects events to both map axes and figure to keep the subplots positioned.

Parameters

- **darray** (*xarray.DataArray*) – DataArray with spatial information
- **x** (*list*) – Array with x-coordinates of locations
- **y** (*list*) – Array with y-coordinates of locations
- **scale** (*float*) – Size of subplots in axes coordinates
- **dim** (*str*) – Name of spatial dimensions
- **figure_kw** (*dict*) – Options passed to `matplotlib.pyplot.subplots()`
- **subplot_kw** (*dict*) – Options passed to `matplotlib.pyplot.add_axes()`
- **kwargs** (*dict*) – Options passed to `xarray.DataArray.plot()`

Returns

- **ax_map** (*AxesSubplot*) – Subplot containing the map
- **axs** (*list of AxesSubplot*) – Positionsed subplots visualizing data

oceanwaves.plot.**position_subplots**(*ax_map*, *axs*, *scale=0.1*)

Updates subplot positions in map

Parameters

- **ax_map** (*AxesSubplot*) – Map axes object
- **axs** (*list of AxesSubplot*) – List of subplots to be positioned. Each axes should have a property `coords` specifying the target position in data coordinates of the map axes.
- **scale** (*float*) – Size of subplots in axes coordinates

`oceanwaves.plot.`**`spatial_map`**(*darray*, *x*, *y*, *ax=None*, *scale=0.1*, *dim='location'*, *add_colorbar=True*, *figure_kw={}*, *subplot_kw={}*, ***kwargs*)

Plot data on a spatial map

Creates a subplot for each location in the DataArray and positions the subplot on a map. Connects events to both map axes and figure to keep the subplots positioned.

Parameters

- **darray** (*xarray.DataArray*) – DataArray with spatial information
- **x** (*list*) – Array with x-coordinates of locations
- **y** (*list*) – Array with y-coordinates of locations
- **scale** (*float*) – Size of subplots in axes coordinates
- **dim** (*str*) – Name of spatial dimensions
- **figure_kw** (*dict*) – Options passed to `matplotlib.pyplot.subplots()`
- **subplot_kw** (*dict*) – Options passed to `matplotlib.pyplot.add_axes()`
- **kwargs** (*dict*) – Options passed to `xarray.DataArray.plot()`

Returns

- **ax_map** (*AxesSubplot*) – Subplot containing the map
- **axs** (*list of AxesSubplot*) – Positionsed subplots visualizing data

2.1.4 Units

`oceanwaves.units.`**`format`**(*parts, order=['kg', 'm', 's', 'Hz']*)

Format unit parts into string

Parameters

- **parts** (*list of 2-tuples*) – List of 2-tuples containing pairs of unit names and exponents
- **order** (*list, optional*) – Preferred order of units in formatted string (default: kg, m, s, Hz)

Returns **units** – Formatted unit specification

Return type str

See also:

parse()

`oceanwaves.units.`**`parse`**(*units*)

Parse unit string into parts

Parameters **units** (*str*) – Unit specification

Returns **parts** – List of 2-tuples containing pairs of unit names and exponents

Return type list of 2-tuples

See also:

format()

oceanwaves.units.**simplify**(*units*)
Simplify the notation of units

Parameters **units** (*str*) – Unit specification

Returns **units** – Simplified unit specification

Return type str

See also:

parse(), *format()*

2.2 What's New

2.2.1 v1.0.1 (unreleased)

Breaking changes

None.

Improvements

- Also read units from quantities other than VarDens (e.g. EnDens and AcDens)
- Package *pyproj* is not an optional dependency. Coordinate conversion is disabled if *pyproj* is not installed. Instead a warning is given in that situation.
- Raise NotImplemented exception if SWAN test file with ITER keyword is read.

New functions/methods

- Added *SwanBlockReader* for memory efficient reading of large files. The *SwanSpcReader* now uses this class to read spectrum files with a minimum required number of lines in memory at each point in the reading procedure.
- Added support for the *ZERO* keyword in SWAN spectrum files. *ZERO* now results in zeros, while *NODATA* results in NaN values.

Bug fixes

- Store comments from SWAN spectrum files as single string instead of a list of strings as the scipy netcdf I/O cannot cope with lists of strings.
- Do not set units attribute on time coordinate if not given, as the time is likely to be given by a list of datetime objects that are automatically encoded by xarray. Setting the units attribute manually would raise an exception if the dataset is written to netCDF.
- Do not squeeze energy matrix as that may cause conflicts with dimensions of length unity. Instead, reshape the energy matrix to the appropriate size.

- Fix a datatype bug in spectral.jonswap() that raised *ValueError: Integers to negative integer powers are not allowed* in the case Hm0 or Tp were integers. Also modify spectral.jonswap() to accept arrays as inputs for Hm0 and Tp. Only one-dimensional arrays for now. - Caio Stringari

Tests

- Added tests for converting SWAN files to netCDF.
- Added to extra example input files: a *.hot hotstart file, a *.sp2 file without data and a *.sp2 file with a single location.
- Test SWAN I/O not only for reading the proper shapes, but also the proper values.

2.2.2 v1.0.0 (15 November 2017)

Initial release

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index