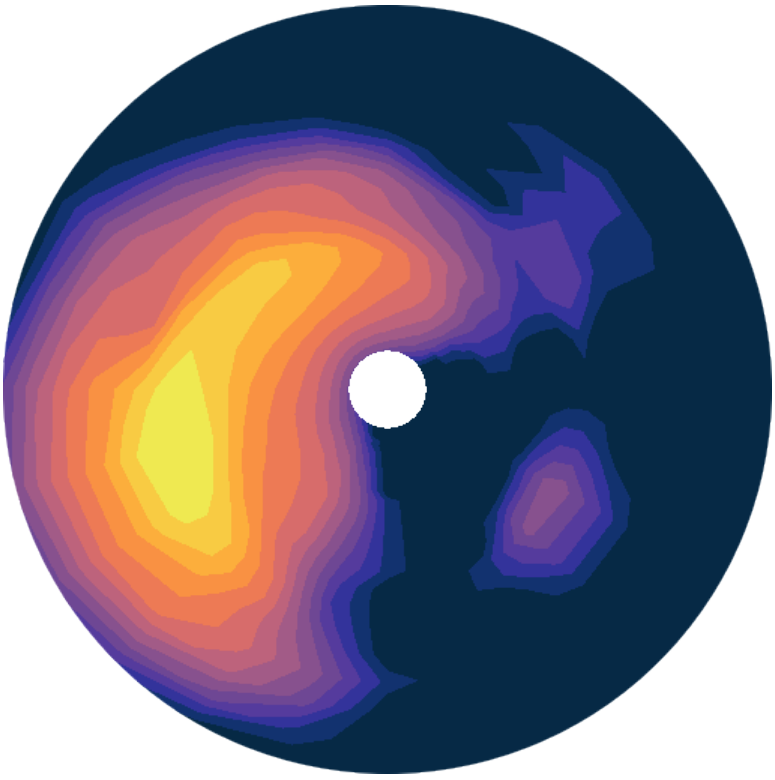


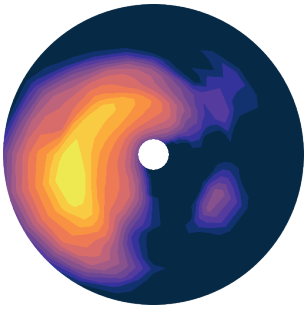
wavespectra Documentation

Release 3.14.0

Wavespectra Developers

Jul 06, 2023

CONTENTS:



CHAPTER

ONE

PYTHON LIBRARY FOR WAVE SPECTRA

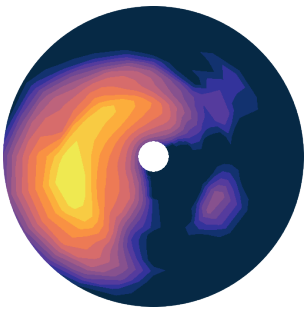
Wavespectra is an open source project for working with ocean wave spectral data hosted on https://github.com/wavespectra/wavespectra. The library is built on top of xarray, leveraging from xarray's labelled multi-dimensional arrays and making dealing with wave spectra simple and fast.

CHAPTER

TWO

DOCUMENTATION

Contents



2.1 Installation

2.1.1 Stable release

The latest stable release of wavespectra package can be installed using pip or conda.

Install using pip

To install wavespectra, run this command in your terminal:

```
$ pip install wavespectra
```

For the full install which includes netcdf4 and some other extra libraries run this command:

```
$ pip install wavespectra[extra]
```

If you don't have pip installed, this Python installation guide can guide you through the process.

Install from conda

```
$ conda install -c conda-forge wavespectra
```

Note: Wavespectra requires a Fortran compiler such as *gfortran* available on the system when installing with *pip* in order to build the watershed partitioning module. Installation from conda-forge includes pre-compiled versions of the code so the compiler is not required.

2.1.2 From sources

The sources for wavespectra can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/wavespectra/wavespectra
```

Or download the tarball:

```
$ curl -OL https://github.com/wavespectra/wavespectra/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

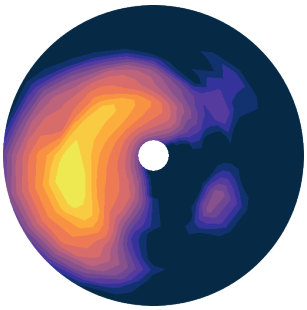
or alternatively in development mode with:

```
$ pip install -e .
```

For the full installation also install the extra requirements:

```
$ pip install -r requirements/extra.txt
```

Again, please make sure a Fortran compiler are available before running any of the commands above to install from source.



2.2 Quick start

Wavespectra is an open source project for processing ocean wave spectral data. The library is built on top of xarray and provides reading and writing of different spectral data formats, calculation of common integrated wave paramaters, spectral partitioning and spectral manipulation in a package focussed on speed and efficiency for large numbers of spectra.

2.2.1 Reading spectra from files

Several methods are provided to read various file formats including spectral wave models like WW3, SWAN, WWM and observation instruments such as TRIAXYS and SPOTTER.

```
In [1]: import matplotlib.pyplot as plt

In [2]: from wavespectra import read_ww3

In [3]: dset = read_ww3("_static/ww3file.nc")

In [4]: dset
Out[4]:
<xarray.Dataset>
Dimensions:  (dir: 24, time: 9, site: 2, freq: 25)
Coordinates:
  * dir      (dir) float32 270.0 255.0 240.0 225.0 ... 330.0 315.0 300.0 285.0
  * freq     (freq) float32 0.04118 0.0453 0.04983 ... 0.3352 0.3687 0.4056
  * site     (site) int32 1 2
  * time     (time) datetime64[ns] 2014-12-01 2014-12-01T12:00:00 ... 2014-12-05
Data variables:
    dpt      (time, site) float32 dask.array<chunksize=(9, 2), meta=np.ndarray>
    efth     (time, site, freq, dir) float32 dask.array<chunksize=(9, 2, 25, 24),␣
↪meta=np.ndarray>
    lat      (site) float32 dask.array<chunksize=(2,), meta=np.ndarray>
    lon      (site) float32 dask.array<chunksize=(2,), meta=np.ndarray>
    wspd     (time, site) float32 dask.array<chunksize=(9, 2), meta=np.ndarray>
    wdir     (time, site) float32 dask.array<chunksize=(9, 2), meta=np.ndarray>
```

2.2.2 The *spec* namespace

Wavespectra defines a new namespace accessor called *spec* which is attached to xarray objects. This namespace provides access to methods from the two main objects in wavespectra:

- *SpecArray*
- *SpecDataset*

which extend functionality from xarray's DataArray and Dataset objects respectively.

SpecArray

```
In [5]: dset.efth.spec
Out[5]:
<SpecArray 'efth' (time: 9, site: 2, freq: 25, dir: 24)>
dask.array<SpecArray shape=(9, 2, 25, 24), dtype=float32, chunksize=(9, 2, 25, 24),␣
↪chunktype=numpy.ndarray>
Coordinates:
  * dir      (dir) float32 270.0 255.0 240.0 225.0 ... 330.0 315.0 300.0 285.0
  * freq     (freq) float32 0.04118 0.0453 0.04983 ... 0.3352 0.3687 0.4056
  * site     (site) int32 1 2
  * time     (time) datetime64[ns] 2014-12-01 2014-12-01T12:00:00 ... 2014-12-05
Attributes:
    standard_name:  sea_surface_wave_directional_variance_spectral_density
    units:          m2 s degree-1
```

SpecDset

```
In [6]: dset.spec
Out[6]:
<SpecDataset>
Dimensions:  (dir: 24, time: 9, site: 2, freq: 25)
Coordinates:
  * dir      (dir) float32 270.0 255.0 240.0 225.0 ... 330.0 315.0 300.0 285.0
  * freq     (freq) float32 0.04118 0.0453 0.04983 ... 0.3352 0.3687 0.4056
  * site     (site) int32 1 2
  * time     (time) datetime64[ns] 2014-12-01 2014-12-01T12:00:00 ... 2014-12-05
Data variables:
    dpt      (time, site) float32 dask.array<SpecDataset>
    efth     (time, site, freq, dir) float32 dask.array<SpecDataset>
    lat      (site) float32 dask.array<SpecDataset>
    lon      (site) float32 dask.array<SpecDataset>
    wspd     (time, site) float32 dask.array<SpecDataset>
    wdir     (time, site) float32 dask.array<SpecDataset>
```

2.2.3 Spectral wave parameters

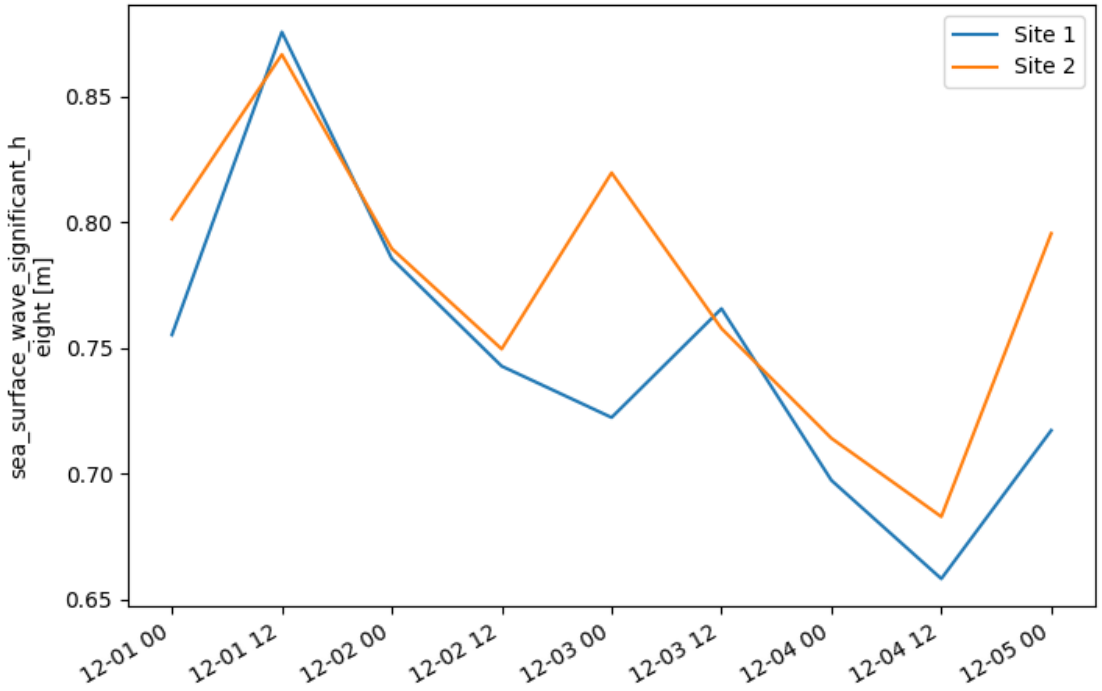
Several methods are available to calculate integrated wave parameters. They can be accessed from both SpecArray (*efth* variable) and SpecDset accessors:

```
In [7]: hs = dset.efth.spec.hs()

In [8]: hs
Out[8]:
<xarray.DataArray 'hs' (time: 9, site: 2)>
dask.array<mul, shape=(9, 2), dtype=float64, chunksize=(9, 2), chunktype=numpy.ndarray>
Coordinates:
  * site     (site) int32 1 2
  * time     (time) datetime64[ns] 2014-12-01 2014-12-01T12:00:00 ... 2014-12-05
Attributes:
```

(continues on next page)

(continued from previous page)

```
    standard_name:  sea_surface_wave_significant_height
    units:          m

In [9]: hs1 = dset.spec.hs()

In [10]: hs.identical(hs1)
Out[10]: True

In [11]: hs.plot.line(x="time");

In [12]: plt.draw()
```

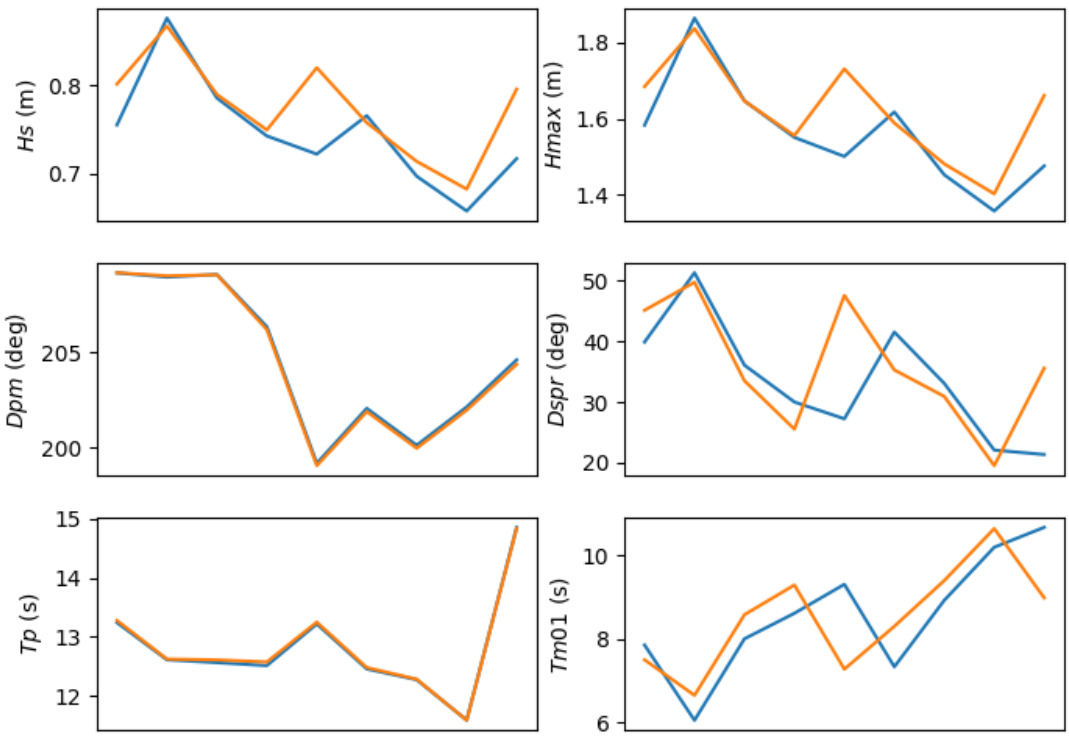



```
In [13]: stats = dset.spec.stats(
   ....:     ["hs", "hmax", "tp", "tm01", "tm02", "dpm", "dm", "dspr", "swe"]
   ....: )
   ....:

In [14]: stats
Out[14]:
<xarray.Dataset>
Dimensions:  (site: 2, time: 9)
Coordinates:
  * site     (site) int32 1 2
  * time     (time) datetime64[ns] 2014-12-01 2014-12-01T12:00:00 ... 2014-12-05
Data variables:
    hs       (time, site) float64 dask.array<chunksize=(9, 2), meta=np.ndarray>
    hmax     (time, site) float64 dask.array<chunksize=(9, 2), meta=np.ndarray>
```

(continues on next page)

(continued from previous page)

```
    tp       (time, site) float32 dask.array<chunksize=(9, 2), meta=np.ndarray>
    tm01     (time, site) float64 dask.array<chunksize=(9, 2), meta=np.ndarray>
    tm02     (time, site) float64 dask.array<chunksize=(9, 2), meta=np.ndarray>
    dpm      (time, site) float32 dask.array<chunksize=(9, 2), meta=np.ndarray>
    dm       (time, site) float32 dask.array<chunksize=(9, 2), meta=np.ndarray>
    dspr     (time, site) float64 dask.array<chunksize=(9, 2), meta=np.ndarray>
    swe      (time, site) float64 dask.array<chunksize=(9, 2), meta=np.ndarray>
Attributes:
    standard_name:  sea_surface_wave_significant_height
    units:          m

In [15]: fig, ((ax1, ax2), (ax3, ax4), (ax5, ax6)) = plt.subplots(3, 2, figsize=(8, 6))

In [16]: stats.hs.plot.line(ax=ax1, x="time");

In [17]: stats.hmax.plot.line(ax=ax2, x="time");

In [18]: stats.dpm.plot.line(ax=ax3, x="time");

In [19]: stats.dspr.plot.line(ax=ax4, x="time");

In [20]: stats.tp.plot.line(ax=ax5, x="time");

In [21]: stats.tm01.plot.line(ax=ax6, x="time");

In [22]: plt.draw()
```



2.2.4 Partitioning

Two different partitioning techniques are available, a simple spectral split based on frequency / direction cutoffs and the watershed algorithm of Hanson et al. (2008).

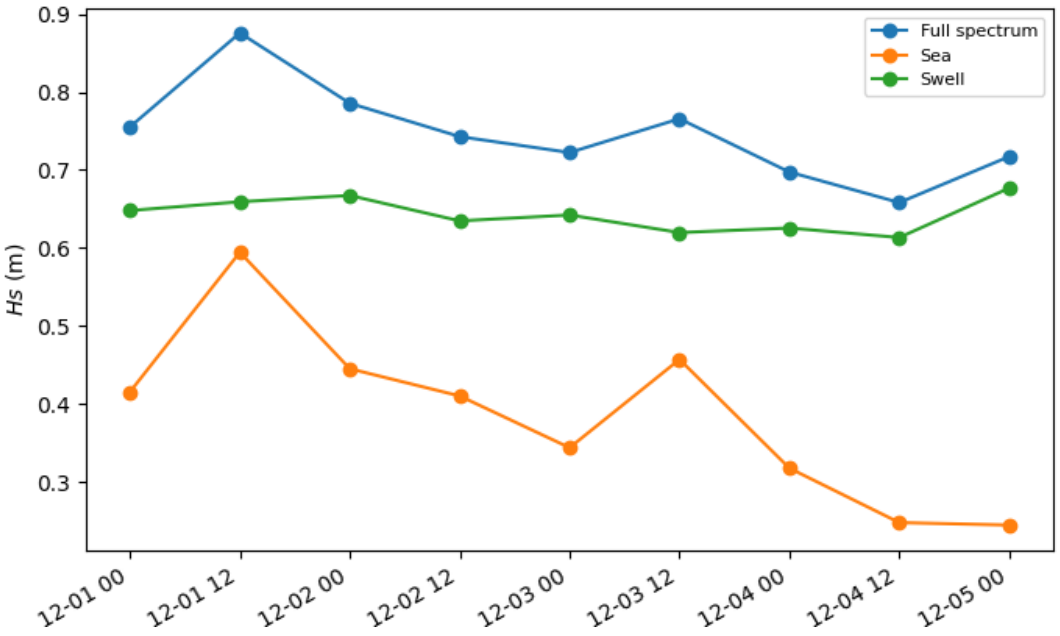
Spectral split

```
In [23]: fcut = 1 / 8

In [24]: sea = dset.spec.split(fmin=fcut)

In [25]: swell = dset.spec.split(fmax=fcut)

In [26]: dset.freq.values
Out[26]:
array([0.04118   , 0.045298  , 0.0498278 , 0.05481058, 0.06029164,
       0.06632081, 0.07295289, 0.08024818, 0.08827299, 0.09710029,
       0.10681032, 0.11749136, 0.1292405 , 0.14216454, 0.15638101,
       0.17201911, 0.18922101, 0.20814312, 0.22895744, 0.25185317,
       0.27703848, 0.30474234, 0.3352166 , 0.36873826, 0.40561208],
      dtype=float32)
```

(continues on next page)

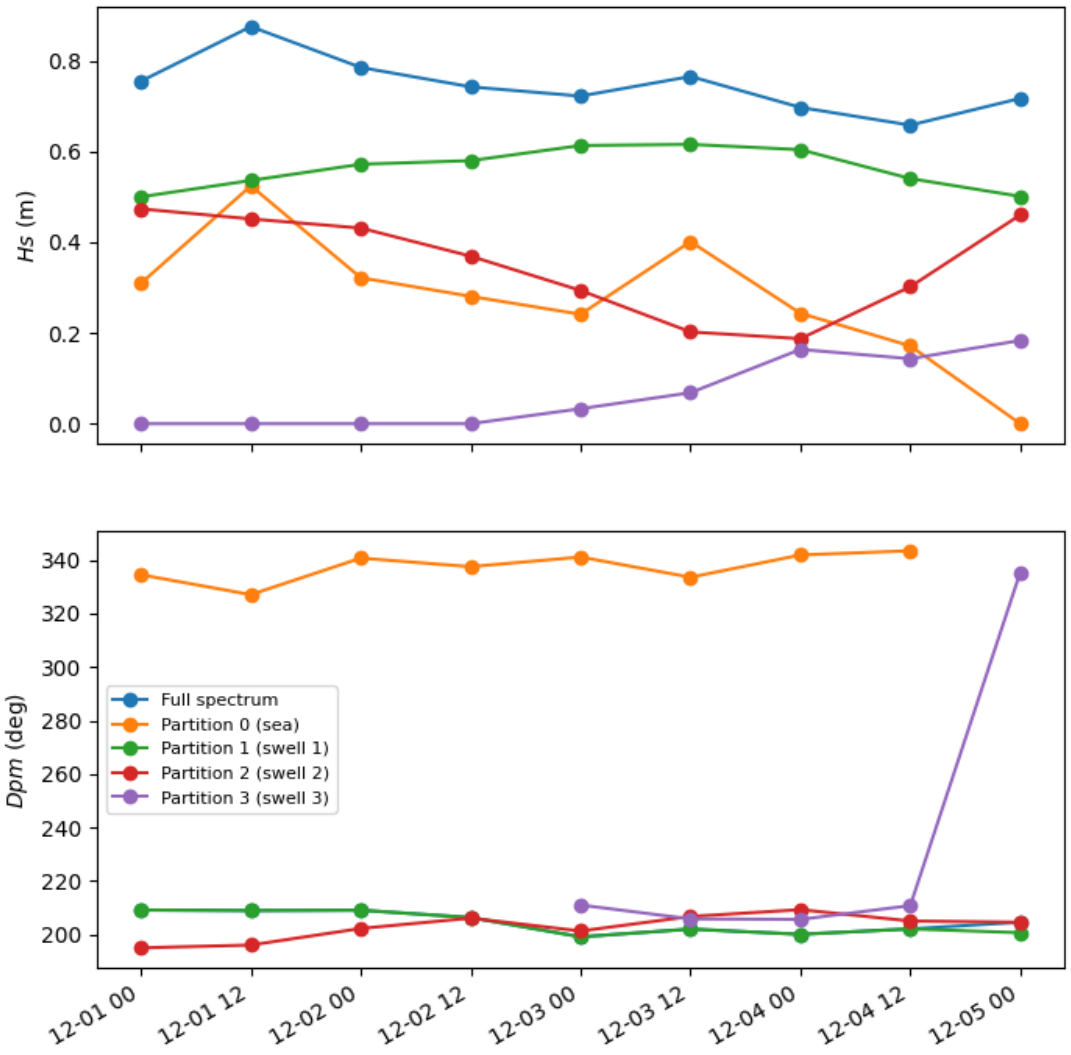
(continued from previous page)

```
In [27]: sea.freq.values
Out[27]:
array([0.125     , 0.1292405 , 0.14216454, 0.15638101, 0.17201911,
       0.18922101, 0.20814312, 0.22895744, 0.25185317, 0.27703848,
       0.30474234, 0.33521661, 0.36873826, 0.40561208])

In [28]: swell.freq.values
Out[28]:
array([0.04118   , 0.045298  , 0.0498278 , 0.05481058, 0.06029164,
       0.06632081, 0.07295289, 0.08024818, 0.08827299, 0.09710029,
       0.10681032, 0.11749136, 0.125     ])

In [29]: dset.spec.hs().isel(site=0).plot(label='Full spectrum', marker='o');

In [30]: sea.spec.hs().isel(site=0).plot(label='Sea', marker='o');

In [31]: swell.spec.hs().isel(site=0).plot(label='Swell', marker='o');

In [32]: plt.draw()
```



Watershed

```
In [33]: dspart = dset.spec.partition(dset.wspd, dset.wdir, dset.dpt)

In [34]: pstats = dspart.spec.stats(["hs", "dpm"])

In [35]: pstats
Out[35]:
<xarray.Dataset>
Dimensions:  (site: 2, time: 9, part: 4)
Coordinates:
  * site     (site) int32 1 2
  * time     (time) datetime64[ns] 2014-12-01 2014-12-01T12:00:00 ... 2014-12-05
  * part     (part) int64 0 1 2 3
Data variables:
    hs       (part, time, site) float64 dask.array<chunksize=(4, 9, 2), meta=np.ndarray>
    dpm      (part, time, site) float32 dask.array<chunksize=(4, 9, 2), meta=np.ndarray>
Attributes:
    standard_name:  sea_surface_wave_significant_height
    units:          m

In [36]: fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 8))

In [37]: hs.isel(site=0).plot(ax=ax1, label='Full spectrum', marker='o');

In [38]: pstats.hs.isel(part=0, site=0).plot(ax=ax1, label='Partition 0 (sea)', marker='o
↪');

In [39]: pstats.hs.isel(part=1, site=0).plot(ax=ax1, label='Partition 1 (swell 1)',␣
↪marker='o');

In [40]: pstats.hs.isel(part=2, site=0).plot(ax=ax1, label='Partition 2 (swell 2)',␣
↪marker='o');

In [41]: pstats.hs.isel(part=3, site=0).plot(ax=ax1, label='Partition 3 (swell 3)',␣
↪marker='o');

In [42]: dset.spec.dpm().isel(site=0).plot(ax=ax2, label='Full spectrum', marker='o');

In [43]: pstats.dpm.isel(part=0, site=0).plot(ax=ax2, label='Partition 0 (sea)', marker=
↪'o');

In [44]: pstats.dpm.isel(part=1, site=0).plot(ax=ax2, label='Partition 1 (swell 1)',␣
↪marker='o');

In [45]: pstats.dpm.isel(part=2, site=0).plot(ax=ax2, label='Partition 2 (swell 2)',␣
↪marker='o');

In [46]: pstats.dpm.isel(part=3, site=0).plot(ax=ax2, label='Partition 3 (swell 3)',␣
↪marker='o');

In [47]: plt.draw()
```



2.2.5 Plotting

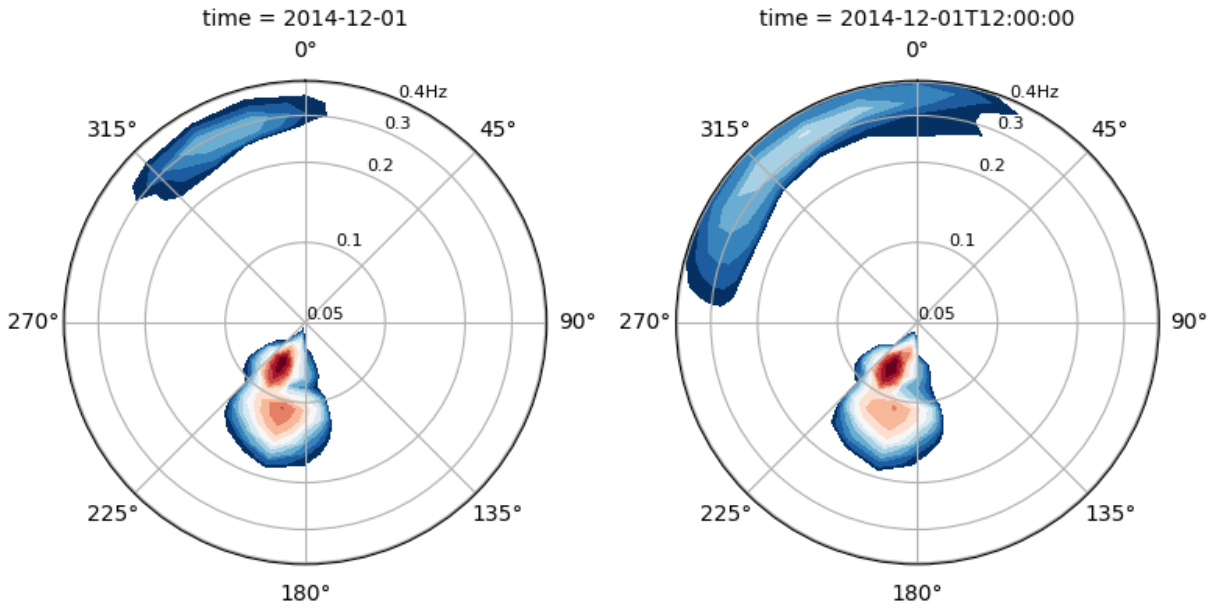
Wavespectra wraps the plotting functionality from xarray to allow easily defining frequency-direction spectral plots in polar coordinates.

```
In [48]: ds = dset.isel(site=0, time=[0, 1]).spec.split(fmin=0.05, fmax=2.0)

In [49]: ds.spec.plot(
   ....:     kind="contourf",
   ....:     col="time",
   ....:     as_period=False,
   ....:     normalised=True,
```

(continues on next page)

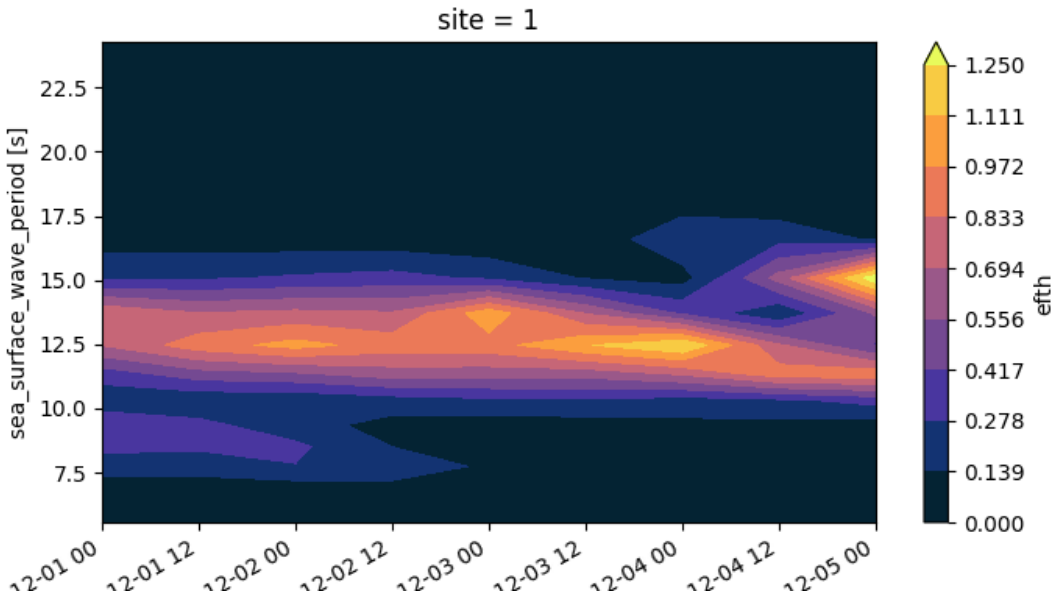
(continued from previous page)

```
....:     logradius=True,
....:     add_colorbar=False,
....:     figsize=(8, 5)
....: );
....:
```



Plotting Hovmoller diagrams of frequency spectra timeseries can be done in only a few lines.

```
In [50]: import cmocean

In [51]: ds = dset.isel(site=0).spec.split(fmax=0.18).spec.oned().rename({"freq": "period
↪"})

In [52]: ds = ds.assign_coords({"period": 1 / ds.period})

In [53]: ds.period.attrs.update({"standard_name": "sea_surface_wave_period", "units": "s
↪"})

In [54]: ds.plot.contourf(x="time", y="period", vmax=1.25, cmap=cmocean.cm.thermal,␣
↪levels=10);
```



2.2.6 Selecting

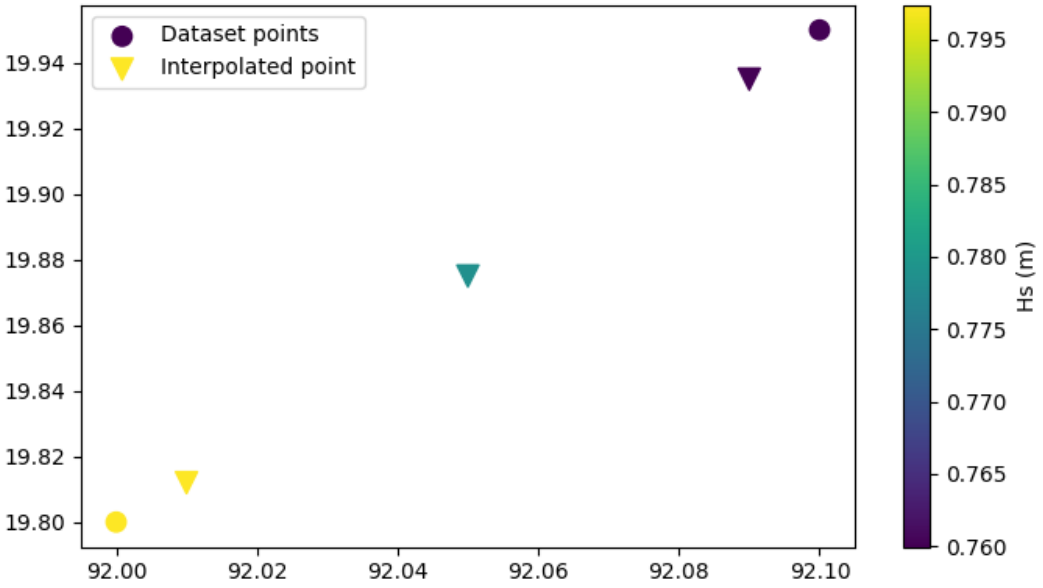
Wavespectra complements xarray's selecting and interpolating functionality with functions to select and interpolate from *site* coordinates with the `sel` method.

```
In [55]: idw = dset.spec.sel(
   ....:     lons=[92.01, 92.05, 92.09],
   ....:     lats=[19.812, 19.875, 19.935],
   ....:     method="idw"
   ....: )
   ....:

In [56]: idw
Out[56]:
<xarray.Dataset>
Dimensions:  (dir: 24, freq: 25, time: 9, site: 3)
Coordinates:
  * dir      (dir) float32 270.0 255.0 240.0 225.0 ... 330.0 315.0 300.0 285.0
  * freq     (freq) float32 0.04118 0.0453 0.04983 ... 0.3352 0.3687 0.4056
  * time     (time) datetime64[ns] 2014-12-01 2014-12-01T12:00:00 ... 2014-12-05
  * site     (site) int64 0 1 2
Data variables:
    dpt      (time, site) float32 dask.array<chunksize=(9, 1), meta=np.ndarray>
    efth     (time, site, freq, dir) float32 dask.array<chunksize=(9, 1, 25, 24),␣
↪meta=np.ndarray>
    lat      (site) float64 19.81 19.88 19.93
    lon      (site) float64 92.01 92.05 92.09
    wspd     (time, site) float32 dask.array<chunksize=(9, 1), meta=np.ndarray>
    wdir     (time, site) float32 dask.array<chunksize=(9, 1), meta=np.ndarray>

In [57]: p = plt.scatter(dset.lon, dset.lat, 80, dset.isel(time=0).spec.hs(), label=
↪"Dataset points");
```

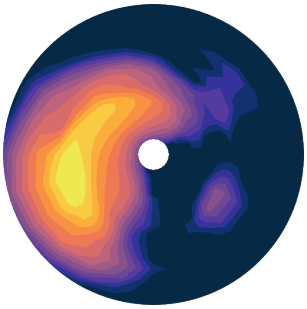
(continues on next page)

(continued from previous page)

```
In [58]: p = plt.scatter(idw.lon, idw.lat, 100, idw.isel(time=0).spec.hs(), marker="v",␣
↪label="Interpolated point");

In [59]: plt.draw()
```



The *nearest* neighbour and *bbox* options are also available besides inverse distance weighting (idw).



2.3 Input & Output

2.3.1 Input

Wavespectra provides several functions for reading datasets from different file formats into *SpecDataset* objects. The functions are defined in submodules within *wavespectra.input*. They can be imported from the main module level for convenience, for instance:

```
In [1]: from wavespectra import read_swan
```

Note: The following conventions are expected for defining reading functions:

- Funcions for different file types are defined in different modules within *wavespectra.input* subpackage.
- Modules are named as *filetype*.py, e.g., `swan.py`.
- Functions are named as read_`filetype`, e.g., `read_swan`.

Input functions can also be defined without following these conventions. However they are not accessible from the main module level and need to be imported from their full module path, e.g.

```
In [2]: from wavespectra.input.swan import read_hotswan
```

Note: Wavespectra currently supports data formats that include `NetCDF`, `ASCII` and `JSON` type files. NetCDF type datasets, i.e. those that can be open with xarray's open_dataset and open_mfdataset functions, can also be prescribed in `ZARR` format (wavespectra uses xarray's open_zarr function behind the scenes to open these files). Functions that support ZARR have a `file_format` argument option.

Note: Files in `ZARR` format can be open from both local and remote (bucket) stores.

These input functions are currently available from the main module level:

read_dataset	Format and attach SpecArray accessor to an existing xarray dataset.
read_era5	Read Spectra from ECMWF ERA5 netCDF format.
read_funwave	Read Spectra in Funwave format.
read_json	Read Spectra from json.
read_ncswan	Read Spectra from SWAN native netCDF format.
read_netcdf	Read Spectra from generic netCDF format.
read_ndbc	Read Spectra from NDBC netCDF format.
read_ndbc_ascii	Read spectra from NDBC wave buoy ASCII files.
read_octopus	Read spectra from Octopus file format.
read_spotter	Read Spectra from Spotter file.
read_swan	Read Spectra from SWAN ASCII file.
read_triaxys	Read spectra from TRIAXYS wave buoy ASCII files.
read_wwm	Read Spectra from WWMII native netCDF format.
read_ww3	Read Spectra from WAVEWATCHIII native netCDF format.

wavespectra.read_dataset

wavespectra.**read_dataset**(*dset*)

Format and attach SpecArray accessor to an existing xarray dataset.

Convenience function to define the SpecArray accessor for a dataset rather than a file. The function guesses the original file format based on variable names.

Args:

dset (xr.Dataset): Spectra dataset with dimensions, coordinates and data_vars consistent any supported file format (currently WW3, SWAN and WWM).

wavespectra.read_era5

wavespectra.**read_era5**(*filename_or_fileglob*, *chunks={}*, *freqs=None*, *dirs=None*)

Read Spectra from ECMWF ERA5 netCDF format.

Args:

- filename_or_fileglob (str): filename or fileglob specifying multiple files to read.
- chunks (dict): chunk sizes for dimensions in dataset. By default dataset is loaded using single chunk for all dimensions (see xr.open_mfdataset documentation).
- freqs (list): list of frequencies. By default use all 30 ERA5 frequencies.
- dirs (list): list of directions. By default use all 24 ERA5 directions.

Returns:

- dset (SpecDataset): spectra dataset object read from netcdf file.

Note:

- If file is large to fit in memory, consider specifying chunks for 'time' and/or 'station' dims.

wavespectra.read_funwave

wavespectra.**read_funwave**(*filename*)

Read Spectra in Funwave format.

Args:

- filename (str): Funwave file to read.

Returns:

- dset (SpecDataset): spectra dataset object read from funwave file.

Note:

- Format description: https://fengyanshi.github.io/build/html/wavemaker_para.html.
- Both 2D E(f,d) and 1d E(f) spectra are supported.
- Directions converted from Cartesian (0E, CCW, to) to wavespectra (0N, CW, from).
- Phases are ignored if present.

wavespectra.read_json

wavespectra.**read_json**(*filename*, *date_format='%Y-%m-%dT%H:%M:%SZ'*)

Read Spectra from json.

The wavespectra json format is produced from *SpecDataset.to_json* by running *Dataset.to_dict* and converting times into iso8601 strings.

Args:

- filename (str): filename of json to read.
- date_format(str): strftime format for de-serializing datetimes.

Returns:

- dset (SpecDataset): spectra dataset object read from json file.

wavespectra.read_ncswan

wavespectra.**read_ncswan**(*filename_or_fileglob*, *file_format='netcdf'*, *mapping={'density': 'efth', 'depth': 'dpt', 'direction': 'dir', 'frequency': 'freq', 'latitude': 'lat', 'longitude': 'lon', 'points': 'site', 'time': 'time'}*, *chunks={}*)

Read Spectra from SWAN native netCDF format.

Args:

- filename_or_fileglob (str): filename or fileglob specifying multiple files to read.
- file_format (str): format of file to open, one of *netcdf* or *zarr*.
- mapping (dict): coordinates mapping from original dataset to wavespectra.
- chunks (dict): chunk sizes for dimensions in dataset. By default dataset is loaded using single chunk for all dimensions (see xr.open_mfdataset documentation).

Returns:

- dset (SpecDataset): spectra dataset object read from ww3 file.

Note:

- If file is large to fit in memory, consider specifying chunks for 'time' and/or 'station' dims.

wavespectra.read_netcdf

wavespectra.**read_netcdf**(*filename_or_fileglob*, *chunks={}*, *freqname='freq'*, *dirname='dir'*, *sitename='site'*, *specname='efth'*, *lonname='lon'*, *latname='lat'*, *timename='time'*)

Read Spectra from generic netCDF format.

Args:

- filename_or_fileglob (str): filename or fileglob specifying multiple files to read.
- chunks (dict): chunk sizes for dimensions in dataset. By default dataset is loaded using single chunk for all dimensions (see xr.open_mfdataset documentation).
- <coord>name :: coordinate name in netcdf, used for standarising dataset.

Returns:

- dset (SpecDataset): spectra dataset object read from netcdf file

Note:

- Assumes frequency in Hz, direction in $degree$ and spectral energy in $m^2 degree^{-1} s$.
- If file is large to fit in memory, consider specifying chunks for 'time' and/or 'station' dims.

wavespectra.read_ndbc

wavespectra.**read_ndbc**(*url*, *directional=True*, *dd=10.0*, *chunks={}*)

Read Spectra from NDBC netCDF format.

Args:

- url (str): Thredds URL or local path of file to read.
- directional (bool): Constructs 2D spectra if True, returns 1D if False.
- dd (float): Directional resolution for 2D spectra (deg).

- chunks (dict): Chunk sizes for dimensions in dataset. By default dataset is loaded using single chunk for all dimensions.

Returns:

- dset (SpecDataset): spectra dataset object read from NDBC file.

Note:

- Any missing values within directional variables in the NDBC dataset will result in NaN in the directional spectra for the respective timestamps. These spectra can only be read as 1D by setting *directional=False*.
- If file is large to fit in memory, consider specifying chunks for 'time' or other non-spectral dims.

wavespectra.read_ndbc_ascii

`wavespectra.`**`read_ndbc_ascii`**(*filename*, *dirs=array([0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350])*)

Read spectra from NDBC wave buoy ASCII files.

Both the history and realtime formats are supported. Realtime formats are decribed at https://www.ndbc.noaa.gov/measdes.shtml.

Args:

- filename (str) or filenames (list): filename of 1D spectral density file or list of the five component files for directional spectra as [*spec*, *swdir*, *swdir2*, *swr1*, *swr2*]. There is no way to verify the component files for the historical directional spectra, so the order entered in the list is what is used. The history and realtime formats are automatically detected.
- dirs (array): vector of directional bins for spectral reconstruction.
- attrs (dict): additional global attributes.

Returns:

- dset (SpecDataset): spectra dataset object read from NDBC buoy file(s).

wavespectra.read_octopus

`wavespectra.`**`read_octopus`**(*filename*)

Read spectra from Octopus file format.

Args:

- filename (str, Path): Octopus file to read.

Returns:

- dset (SpecDataset): spectra dataset object.

wavespectra.read_spotter

wavespectra.**read_spotter**(*filename*, *filetype=None*)

Read Spectra from Spotter file.

Args:

- filename (list, str): File name or file glob specifying spotter files to read.
- filetype (str): 'json' or 'csv', if not passed inferred from filename.

Returns:

- dset (SpecDataset): spectra dataset object read from file.

wavespectra.read_swan

wavespectra.**read_swan**(*filename*, *dirorder=True*, *as_site=False*)

Read Spectra from SWAN ASCII file.

Args:

- dirorder (bool): If True reorder spectra so that directions are sorted.
- as_site (bool): If True locations are defined by 1D site dimension.

Returns:

- dset (SpecDataset): spectra dataset object read from file.

wavespectra.read_triaxys

wavespectra.**read_triaxys**(*filename_or_fileglob*, *toff=0*, *magnetic_variation=None*, *regrid_dir=True*)

Read spectra from TRIAXYS wave buoy ASCII files.

Args:

- filename_or_fileglob (str): filename or fileglob specifying one or more files to read.
- toff (float): time-zone offset from UTC in hours.
- magnetic_variation (float): The angle between the magnetic and geographic meridians to correct from magnetic north to true north. Positive and negative values indicate East and West declination respectively.
- regrid_dir (bool): Regrid directions after correcting for the magnetic variation so the direction coordinate remains the same.

Returns:

- dset (SpecDataset): spectra dataset object read from Triaxys file.

Note:

- frequencies and directions from first file are used as reference to interpolate spectra from other files in case they differ.
- Duplicate dir coordinates (e.g., 0 and 360) are removed when correcting for the magnetic variation since indices must be unique to regrid.

wavespectra.read_wwm

wavespectra.**read_wwm**(*filename_or_fileglob*, *file_format='netcdf'*, *mapping={'AC': 'efth', 'DEP': 'dpt', 'lat': 'lat', 'lon': 'lon', 'nbstation': 'site', 'ndir': 'dir', 'nfreq': 'freq', 'ocean_time': 'time'}*, *chunks={}*)

Read Spectra from WWMII native netCDF format.

Args:

- filename_or_fileglob (str): filename or fileglob specifying multiple files to read.
- file_format (str): format of file to open, one of *netcdf* or *zarr*.
- mapping (dict): coordinates mapping from original dataset to wavespectra.
- chunks (dict): chunk sizes for dimensions in dataset. By default dataset is loaded using single chunk for all dimensions (see xr.open_mfdataset documentation).

Returns:

- dset (SpecDataset): spectra dataset object read from ww3 file.

Note:

- If file is large to fit in memory, consider specifying chunks for 'time' and/or 'station' dims.

wavespectra.read_ww3

wavespectra.**read_ww3**(*filename_or_fileglob*, *file_format='netcdf'*, *mapping={'direction': 'dir', 'efth': 'efth', 'frequency': 'freq', 'latitude': 'lat', 'longitude': 'lon', 'station': 'site', 'time': 'time', 'wnd': 'wspd', 'wnddir': 'wdir'}*, *chunks={}*)

Read Spectra from WAVEWATCHIII native netCDF format.

Args:

- filename_or_fileglob (str): filename or fileglob specifying multiple files to read.
- file_format (str): format of file to open, one of *netcdf* or *zarr*.
- mapping (dict): coordinates mapping from original dataset to wavespectra.
- chunks (dict): chunk sizes for dimensions in dataset. By default dataset is loaded using single chunk for all dimensions (see xr.open_mfdataset documentation). Dimension names from original dataset or from wavespectra can be used to specify chunks dict.

Returns:

- dset (SpecDataset): spectra dataset object read from ww3 file.

Note:

- If file is large to fit in memory, consider specifying chunks for 'time' and/or 'station' dims.

These functions are not accessible from the main module level and need to be imported from their full module path:

input.swan.read_swans	Read multiple SWAN ASCII files into single Dataset.
input.swan.read_hotswan	Read partial SWAN hotfiles into single gridded hotfile Dataset.
input.swan.read_swanow	Read SWAN nowcast from fileglob, keep overlapping dates from most recent files.

wavespectra.input.swan.read_swans

wavespectra.input.swan.**read_swans**(*fileglob*, *ndays=None*, *int_freq=True*, *int_dir=False*, *dirorder=True*, *ntimes=None*)

Read multiple SWAN ASCII files into single Dataset.

Args:

- fileglob (str, list): glob pattern specifying files to read.
- ndays (float): number of days to keep from each file, choose None to keep entire period.
- **int_freq (ndarray, bool): frequency array for interpolating onto:**
  - ndarray: 1d array specifying frequencies to interpolate onto.
  - True: logarithm array is constructed such that fmin=0.0418 Hz, fmax=0.71856 Hz, df=0.1f.
  - False: No interpolation performed in frequency space.
- **int_dir (ndarray, bool): direction array for interpolating onto:**
  - ndarray: 1d array specifying directions to interpolate onto.
  - True: circular array is constructed such that dd=10 degrees.
  - False: No interpolation performed in direction space.
- dirorder (bool): if True ensures directions are sorted.
- ntimes (int): use it to read only specific number of times, useful for checking headers only.

Returns:

- dset (SpecDataset): spectra dataset object read from file with different sites and cycles concatenated along the 'site' and 'time' dimensions.

Note:

- If multiple cycles are provided, 'time' coordinate is replaced by 'cycletime' multi-index coordinate.
- If more than one cycle is prescribed from fileglob, each cycle must have same number of sites.
- Either all or none of the spectra in fileglob must have tabfile associated to provide wind/depth data.
- Concatenation is done with numpy arrays for efficiency.

wavespectra.input.swan.read_hotswan

wavespectra.input.swan.**read_hotswan**(*fileglob*, *dirorder=True*)

Read partial SWAN hotfiles into single gridded hotfile Dataset.

Args:

- fileglob (str, list): glob pattern specifying hotfiles to read and merge.
- dirorder (bool): if True ensures directions are sorted.

Returns:

- dset (SpecDataset): spectra dataset object with different grid parts merged.

Note:

- SWAN hotfiles from mpi runs are split by the number of cores over the largest dim of (lat, lon) with overlapping rows or columns that are computed in only one of the split hotfiles. Here overlappings are merged so that those with higher values are kept which assumes non-computed overlapping rows or columns are filled with zeros.

wavespectra.input.swan.read_swanow

`wavespectra.input.swan.`**`read_swanow`**(*fileglob*)

Read SWAN nowcast from fileglob, keep overlapping dates from most recent files.

Inefficient workaround. This should ideally be handled within read_swans by manipulating multi-indexes

2.3.2 Output

Wavespectra provides functions to write *SpecDataset* objects into different file types. They are defined in module within `wavespectra.output`. They are attached as methods in the SpecDataset accessor.

Note: The following conventions are expected for defining output functions:

- Funcions for different file types are defined in different modules within `wavespectra.output` subpackage.
- Modules are named as *filetype*.py, e.g., `swan.py`.
- Functions are named as to_`filetype`, e.g., `to_swan`.
- Function **must** accept `self` as the first input argument.

These output functions are currently available as methods of *SpecDataset*:

SpecDataset.to_funwave	Write spectra in Funwave format.
SpecDataset.to_json	Write spectra in json format.
SpecDataset.to_netcdf	Write spectra in netCDF format using wavespectra conventions.
SpecDataset.to_octopus	Save spectra in Octopus format.
SpecDataset.to_orcaflex	Writes the spectrum to an Orcaflex model
SpecDataset.to_swan	Write spectra in SWAN ASCII format.
SpecDataset.to_ww3	Save spectra in native WW3 netCDF format.

wavespectra.SpecDataset.to_funwave

`SpecDataset.`**`to_funwave`**(*filename*, *clip=True*)

Write spectra in Funwave format.

Args:

- filename (str): Name for output Funwave file.
- clip (bool): Clip directions outside [-90, 90] range in cartesian convention.

Note:

- Format description: https://fengyanshi.github.io/build/html/wavemaker_para.html.
- Directions converted from wavespectra (0N, CW, from) to Cartesian (0E, CCW, to).

- Funwave only seems to deal with directions in the [-90, 90] range in cartesian convention, use clip=True to clip spectra outside that range.
- Both 2D E(f,d) and 1d E(f) spectra are supported.
- If the SpecArray has more than one spectrum, multiple files are created in a zip archive defined by replacing the extension of *filename* by ".zip".

wavespectra.SpecDataset.to_json

`SpecDataset.to_json`(*filename*, *mode='w'*, *date_format='%Y-%m-%dT%H:%M:%SZ'*)

Write spectra in json format.

Xarray's *to_dict* it used to dump dataset into dictionary to save as a json file.

Args:

- filename (str): name of output json file.
- mode (str): file mode, by default *w* (create or overwrite).
- date_format(str): strftime format for serializing datetimes.

wavespectra.SpecDataset.to_netcdf

`SpecDataset.to_netcdf`(*filename*, *specname='efth'*, *ncformat='NETCDF4'*, *compress=True*, *packed=True*, *time_encoding={'units': 'seconds since 1970-01-01'}*)

Write spectra in netCDF format using wavespectra conventions.

Args:

- filename (str): name of output netcdf file.
- specname (str): name of spectra variable in dataset.
- ncformat (str): netcdf format for output, see options in native to_netcdf method.
- compress (bool): if True output is compressed, has no effect for NETCDF3.
- packed (bool): Pack spectra as int32 dtype and 1e-5 scale_factor.
- time_encoding (dict): force standard time units in output files.

wavespectra.SpecDataset.to_octopus

`SpecDataset.to_octopus`(*filename*, *site_id='spec'*, *fcut=0.125*, *missing_val=-99999*, *ntime=None*)

Save spectra in Octopus format.

Args:

- filename (str): name for output OCTOPUS file.
- site_id (str): used to construct LPoint header.
- fcut (float): frequency for splitting spectra.
- missing_value (int): missing value in output file.
- ntime (int, None): number of times to load into memory before dumping output file if full dataset does not fit into memory, choose None to load all times.

Note:

- dataset needs to have lon/lat/time coordinates.
- dataset with multiple locations dumped at same file with one location header per site.
- 1D spectra not supported.
- ntime=None optimises speed as the dataset is loaded into memory however the dataset may not fit into memory in which case a smaller number of times may be prescribed.

wavespectra.SpecDataset.to_orcaflex

`SpecDataset.`**`to_orcaflex`**(*model*, *minEnergy=1e-06*)

Writes the spectrum to an Orcaflex model

Uses the orcaflex API (OrcFxAPI) to set the wave-data of the provided orcaflex model.

The axis system conversion used is: - Orcaflex global X = Towards East - Orcaflex global Y = Towards North

This function creates a wave-train for each of the directions in this spectrum using a user-defined spectrum.

Calculation of wave-components in orcaflex is computationally expensive. To save computational time: 1. Use the minEnergy parameter of this function to define a treshold for the amount of energy in a wave-train. 2. In orcaflex itself: limit the amount of wave-components 3. Before exporting: regrid the spectrum to a lower amount of directions.

Orcaflex theory: - https://www.orcina.com/webhelp/OrcaFlex/Content/html/Wavetheory.htm - https://www.orcina.com/webhelp/OrcaFlex/Content/html/Directionconventions.htm

Example:

```
>>> from OrcFxAPI import *
>>> from wavespectra import read_triaxys
>>> m = Model()
>>> spectrum = read_triaxys("triaxys.DIRSPEC")).isel(time=0)  # get only the␣
↪fist spectrum in time
>>> spectrum.spec.to_orcaflex(m)
```

Args:

- model : orcaflex model (OrcFxAPI.model instance)
- minEnergy [1e-6] : threshold for minimum sum of energy in a direction before it is exported

Note:

- an Orcaflex license is required to work with the orcaflex API.
- Only 2D spectra E(f,d) are currently supported.
- The DataArray should contain only a single spectrum. Hint: first_spetrum = spectra.isel(time=0)

wavespectra.SpecDataset.to_swan

SpecDataset.**to_swan**(*filename*, *append=False*, *id='Created by wavespectra'*, *ntime=None*)

Write spectra in SWAN ASCII format.

Args:

- filename (str): str, name for output SWAN ASCII file.
- append (bool): if True append to existing filename.
- id (str): used for header in output file.
- ntime (int, None): number of times to load into memory before dumping output file if full dataset does not fit into memory, choose None to load all times.

Note:

- Only datasets with lat/lon coordinates are currently supported.
- Extra dimensions other than time, site, lon, lat, freq, dim not yet supported.
- Only 2D spectra E(f,d) are currently supported.
- ntime=None optimises speed as the dataset is loaded into memory however the dataset may not fit into memory in which case a smaller number of times may be prescribed.

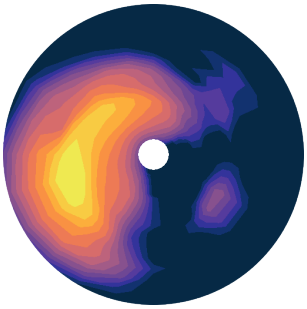
wavespectra.SpecDataset.to_ww3

SpecDataset.**to_ww3**(*filename*, *ncformat='NETCDF4'*, *compress=False*)

Save spectra in native WW3 netCDF format.

Args:

- filename (str): name of output WW3 netcdf file.
- ncformat (str): netcdf format for output, see options in native to_netcdf method.
- compress (bool): if True output is compressed, has no effect for NETCDF3.



2.4 Conventions

Wavespectra takes advantage of xarray's labelled coordinates to abstract n-dimensional wave spectra datasets and calculate integrated spectral parameters. This requires some conventions around special coordinates and variables. This naming convention is inspired on netcdf files from WAVEWATCH III wave model.

2.4.1 Coordinates

Wave frequency

Wave frequency coordinate named as `freq`, defined in Hz (required).

Wave direction

Wave direction coordinate in coming-from convention, named as `dir`, defined in $degree$ (required for 2D spectra and directional methods).

Time

Time coordinate named as `time` (required by some methods).

2.4.2 Data variables

Wave energy density

Wave energy density array named as `efth`, defined in:

- 2D spectra $E(\sigma, \theta)$: $m^2 degree^{-1} s$.
- 1D spectra $E(\sigma)$: $m^2 s$.

Wind speed

Wind speed array named as `wspd`, defined in ms^{-1} (required for watershed partitioning).

Wind direction

Wind direction array named as `wdir`, defined in $degree$ (required for watershed partitioning).

Water depth

Water depth array named as `dpt`, defined in m (required for watershed partitioning and wavenumber-based methods).

2.4.3 Attributes

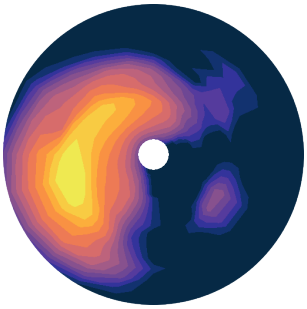
Pre-defined names and units for these and other coordintes and variables are available from module `wavespectra.core.attributes`. This module defines variable names and some CF attributes by loading information from attributes.yml file. The attributes can be accessed for example as:

```
In [1]: from wavespectra.core.attributes import attrs

In [2]: attrs.SPECNAME
Out[2]: 'efth'

In [3]: attrs.ATTRS.hs
Out[3]:
{'standard_name': 'sea_surface_wave_significant_height',
 'units': 'm',
 '_ipython_canary_method_should_not_exist_': {}}
```

The module also provides a function to standarise coordinate and variable attributes in a Dataset object using the information defined in attributes.yml:

`wavespectra.core.attributes.`**`set_spec_attributes`**(*dset*)

Standarise CF attributes in specarray variables



2.5 Plotting

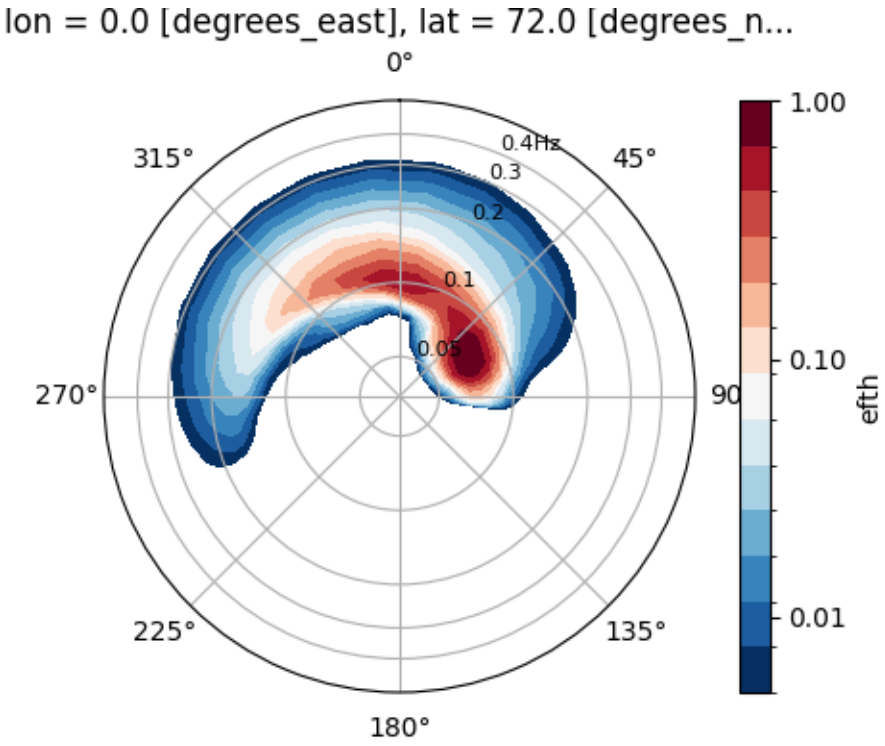
Wavespectra wraps the plotting functionality from xarray to allow easily defining frequency-direction spectral plots in polar coordinates.

```
In [1]: import matplotlib.pyplot as plt

In [2]: import cmocean

In [3]: from wavespectra import read_swan, read_era5

In [4]: dset = read_era5("_static/era5file.nc").isel(time=0)

In [5]: ds = dset.isel(lat=0, lon=0)
```

2.5.1 Simplest usage

The *plot* method is available in *SpecArray*. The simplest usage takes no arguments and defines sensible settings for plotting normalised spectra on logarithmic radii and countour levels:

```
In [6]: ds.spec.plot();
```

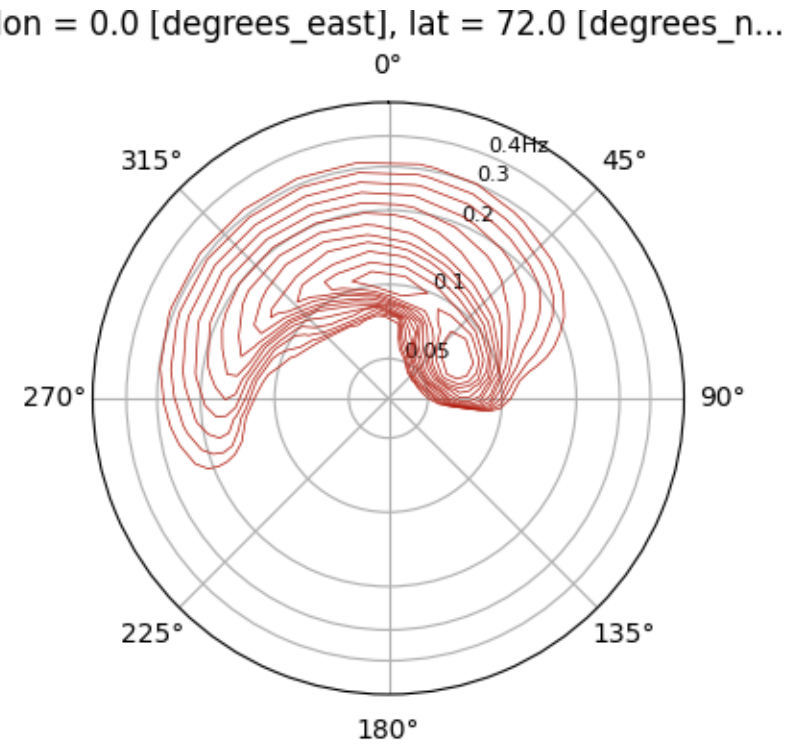


2.5.2 Plotting types

Wavespectra supports xarray's contour, `contourf`_ and pcolormesh plotting types.

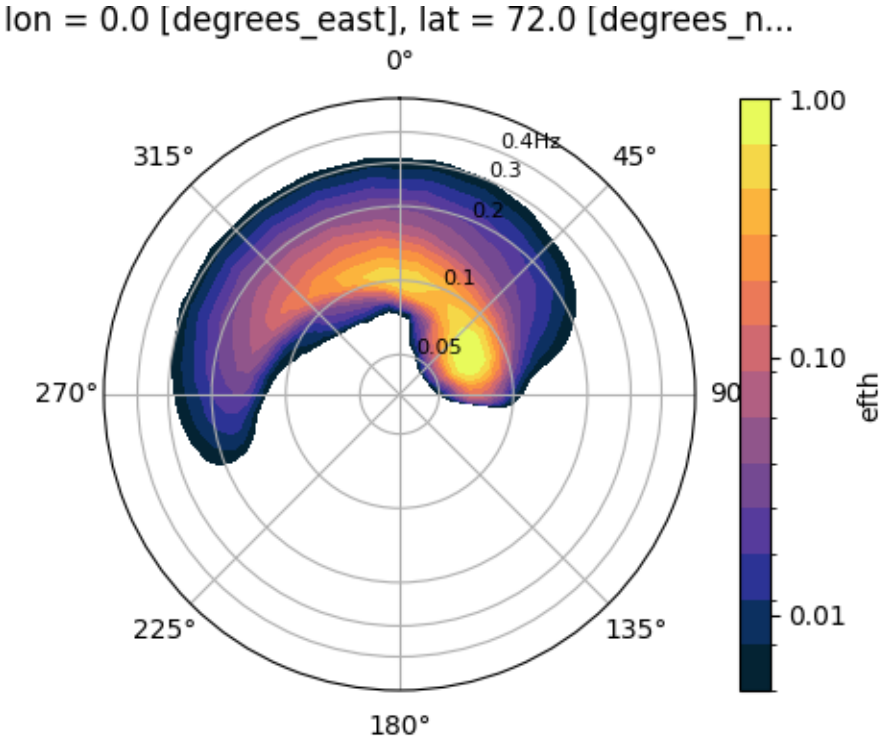
Contour

```
In [7]: ds.spec.plot(kind="contour", colors="#af1607", linewidths=0.5);
```



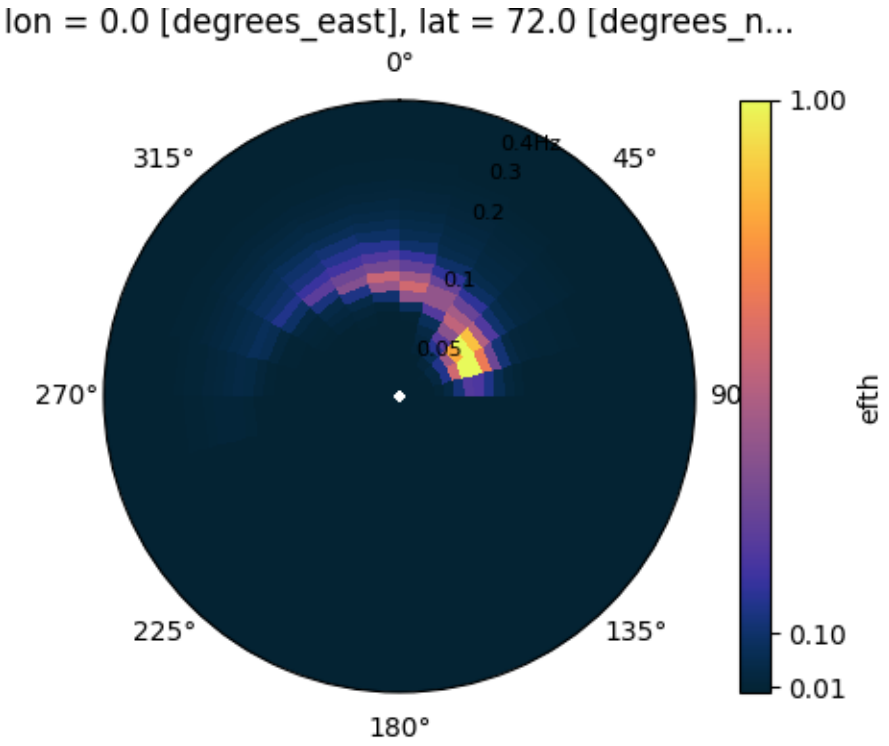
Contourf

```
In [8]: ds.spec.plot(kind="contourf", cmap=cmocean.cm.thermal);
```



Pcolormesh

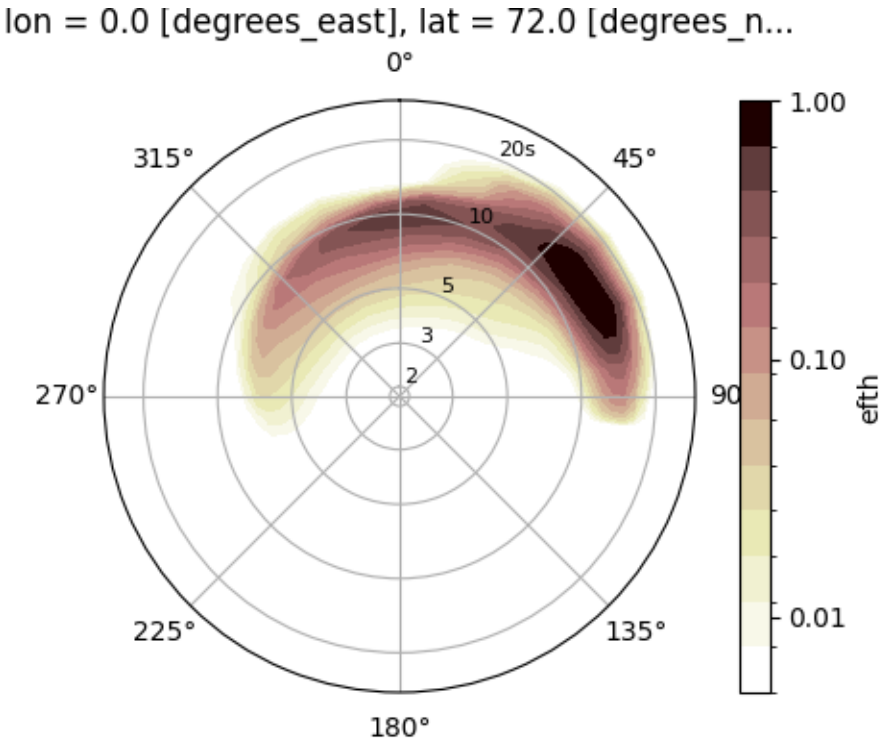
```
In [9]: ds.spec.plot(kind="pcolormesh", cmap=cmocean.cm.thermal);
```



2.5.3 Wave period spectrum

Frequency-direction spectra can be easily plotted in the period space.

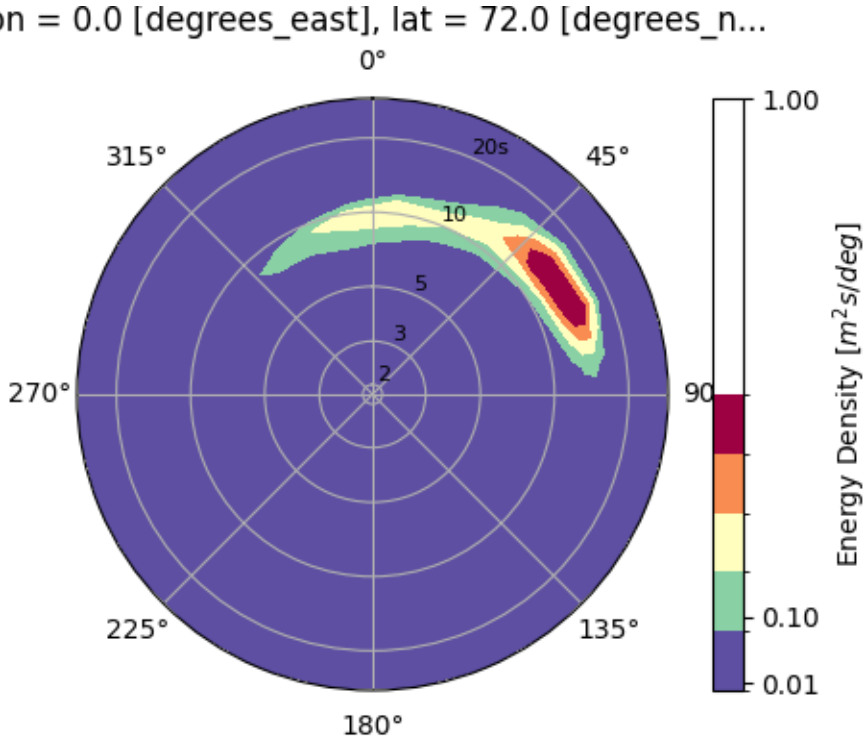
```
In [10]: ds.spec.plot(as_period=True, cmap="pink_r");
```



2.5.4 Normalised

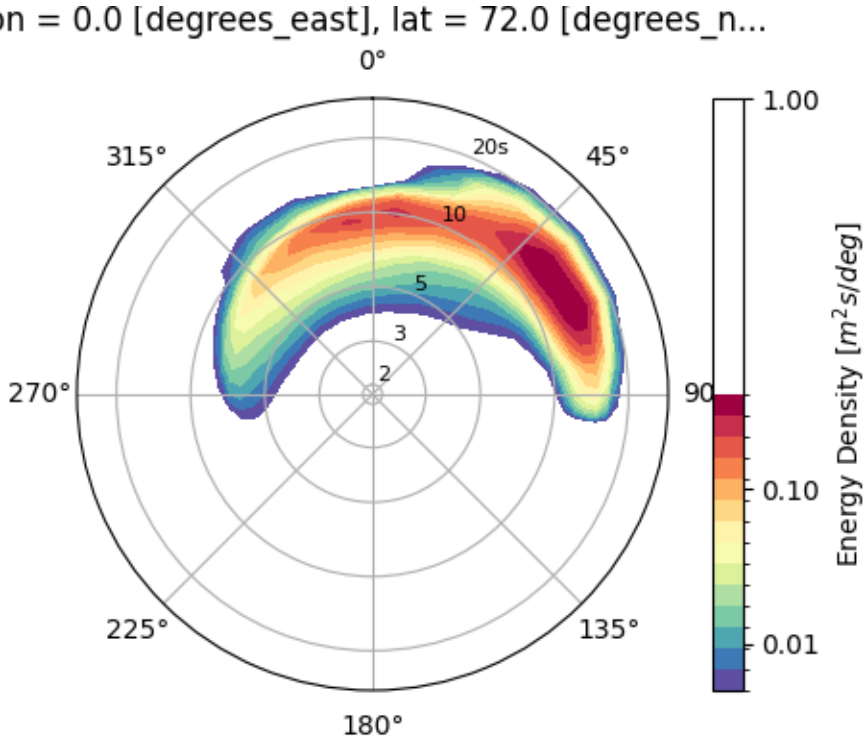
The normalised spectrum $\frac{E_d(f,d)}{\max E_d}$ is plotted by default but the actual values can be shown instead:

```
In [11]: ds.spec.plot(as_period=True, normalised=False, cmap="Spectral_r");
```



Logarithmic contour levels are only default for normalised spectra but they can be still manually specified:

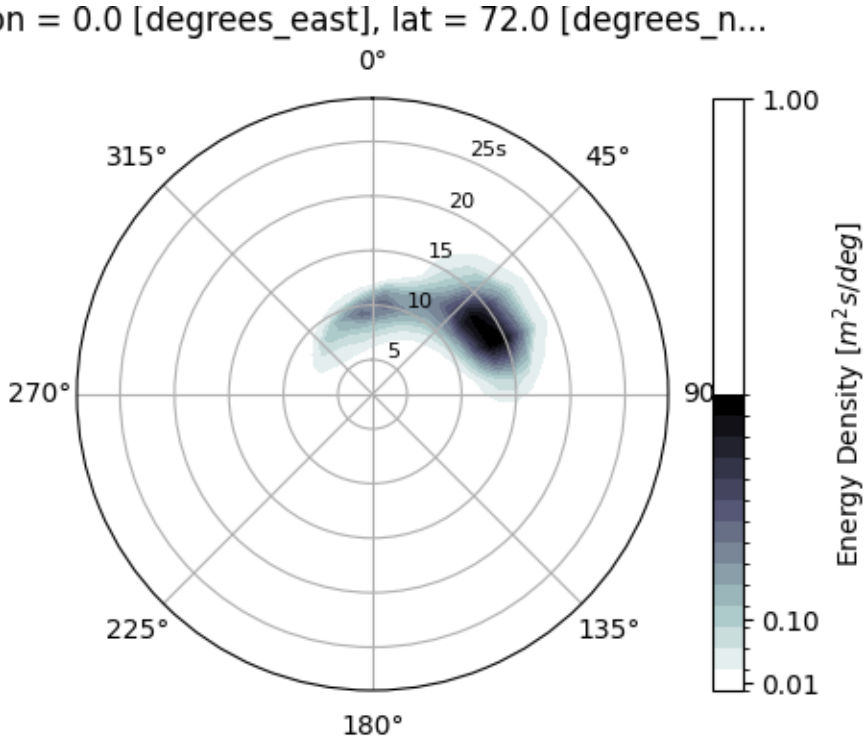
```
In [12]: ds.spec.plot(
   ....:     as_period=True,
   ....:     normalised=False,
   ....:     cmap="Spectral_r",
   ....:     levels=np.logspace(np.log10(0.005), np.log10(0.4), 15),
   ....:     cbar_ticks=[0.01, 0.1, 1],
   ....: );
   ....:
```



2.5.5 Logarithmic radii

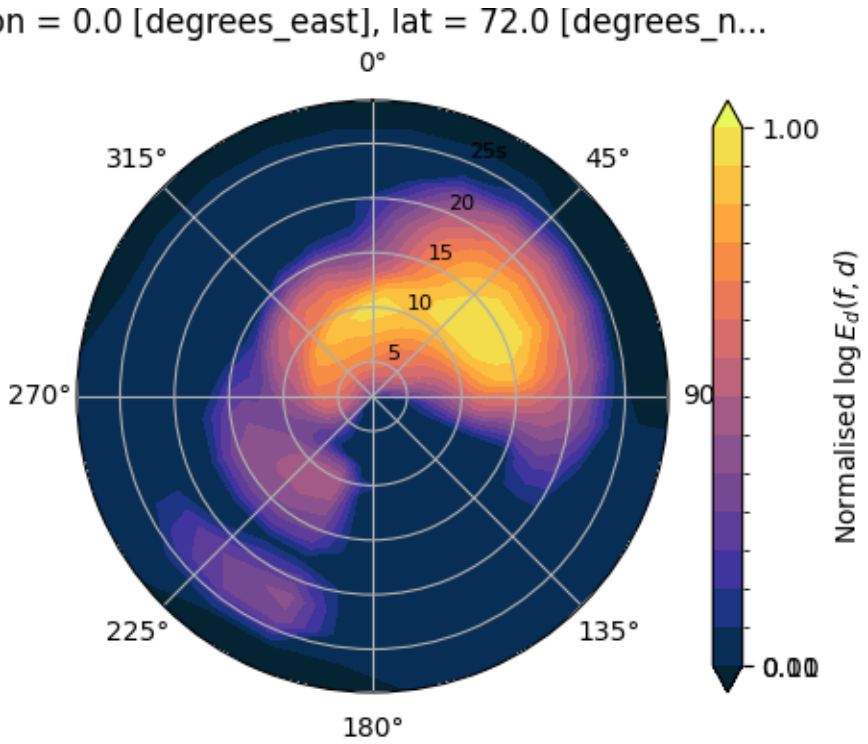
Radii are shown in a logarithmic scale by default. Linear radii can be defined by setting *logradius=False* (radii ticks can be prescribed from the *radii_ticks* paramater):

```
In [13]: ds.spec.plot(
   ....:     as_period=True,
   ....:     normalised=False,
   ....:     levels=15,
   ....:     cmap="bone_r",
   ....:     logradius=False,
   ....:     radii_ticks=[5, 10, 15, 20, 25],
   ....: );
   ....:
```



Hint: The *as_log10* option to plot the $\log E_d(f,d)$ has been deprecated but similar result can be achieved by calculating the $\log E_d(f,d)$ beforehand:

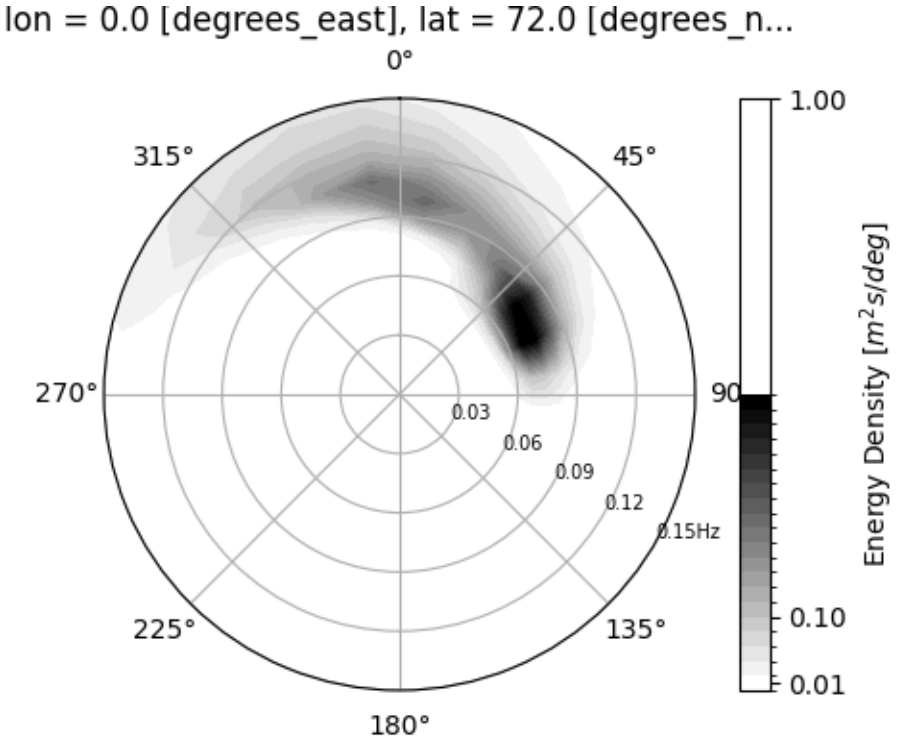
```
In [14]: ds1 = ds.where(ds>0, 1e-5) # Avoid infinity values

In [15]: ds1 = np.log10(ds1)

In [16]: ds1.spec.plot(
   ....:     as_period=True,
   ....:     logradius=False,
   ....:     cbar_kwargs={"label": "Normalised $\log{E_{d}(f,d)}$"},
   ....:     vmin=0.39,
   ....:     levels=15,
   ....:     extend="both",
   ....:     cmap=cmocean.cm.thermal,
   ....: );
   ....:
```



2.5.6 Radii extents

The radii extents are controlled from *rmin* and *rmax* parameters:

```
In [17]: ds.spec.plot(
   ....:     rmin=0,
   ....:     rmax=0.15,
   ....:     logradius=False,
   ....:     normalised=False,
   ....:     levels=25,
   ....:     cmap="gray_r",
   ....:     radii_ticks=[0.03, 0.06, 0.09, 0.12, 0.15],
   ....:     radii_labels=["0.05", "0.1", "0.15Hz"],
   ....:     radii_labels_angle=120,
   ....:     radii_labels_size=7,
   ....: );
   ....:

In [18]: plt.draw()
```



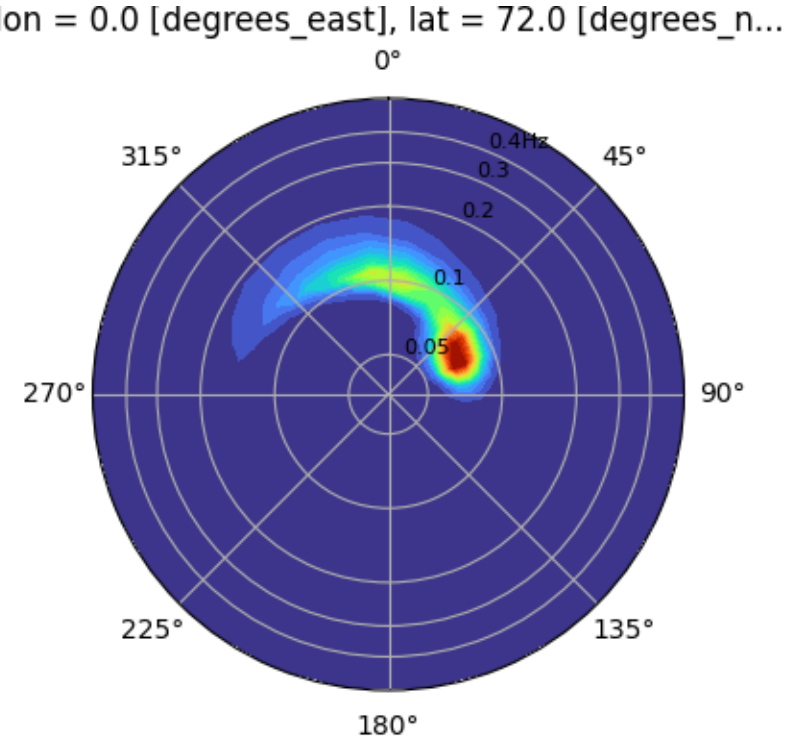
Exclusive plotting parameters from wavespectra

- **kind** ("contourf") : Plot kind, one of ("contourf", "contour", "pcolormesh").
- **normalised** (True): Plot the normalised $E(f, d)$ between 0 and 1.
- **logradius** (True): Set log radii.
- **as_period** (False): Set wave period radii instead of frequency.
- **show_radii_labels** (True): Display the radii tick labels.
- **show_theta_labels** (False): Display the directions tick labels.
- **radii_ticks** (array): Tick values for radii.
- **radii_labels_angle** (22.5): Polar angle at which radii labels are positioned.
- **radii_labels_size** (8): Fontsize for radii labels.
- **cbar_ticks**: Tick values for colorbar (default depends if normalised, logradius and as_period).
- **clean_axis** (False): Remove radii and theta ticks for a clean view.

2.5.7 Plotting parameters from xarray

Wavespectra allows passing some parameters from the functions wrapped from xarray such as contourf (excluding some that are manipulated in wavespectra such as *ax*, *x* and others):

```
In [19]: ds.spec.plot(
   ....:     kind="contourf",
   ....:     cmap="turbo",
   ....:     add_colorbar=False,
   ....:     extend="both",
   ....:     levels=25,
   ....: );
   ....:
```



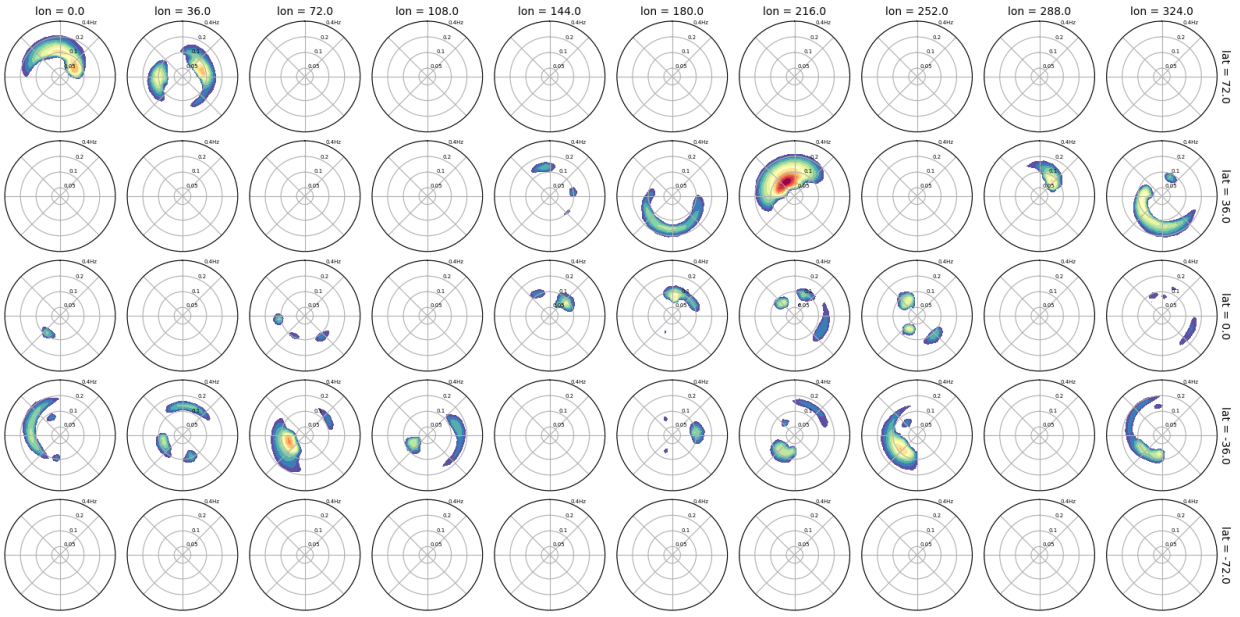
Some of the xarray parameters that are not exposed in wavespectra

- **projection**: Always set to "polar".
- **x**, **y**: Set to wavespectra coordinates naming.
- **xlabel**, **ylabel**: Disabled.
- **ax**, **aspect**, **size**: Conflict with axes defined in wavespectra.
- **xlim**, **ylim**: produce no effect.

2.5.8 Faceting

Xarray's faceting capability is fully supported.

```
In [20]: dset.spec.plot(
   ....:     col="lon",
   ....:     row="lat",
   ....:     figsize=(16,8),
   ....:     add_colorbar=False,
   ....:     show_theta_labels=False,
   ....:     show_radii_labels=True,
   ....:     radii_ticks=[0.05, 0.1, 0.2, 0.4],
   ....:     rmax=0.4,
   ....:     radii_labels_size=5,
   ....:     cmap="Spectral_r",
   ....: );
   ....:

In [21]: plt.tight_layout()

In [22]: plt.draw()
```



2.5.9 Clean axes

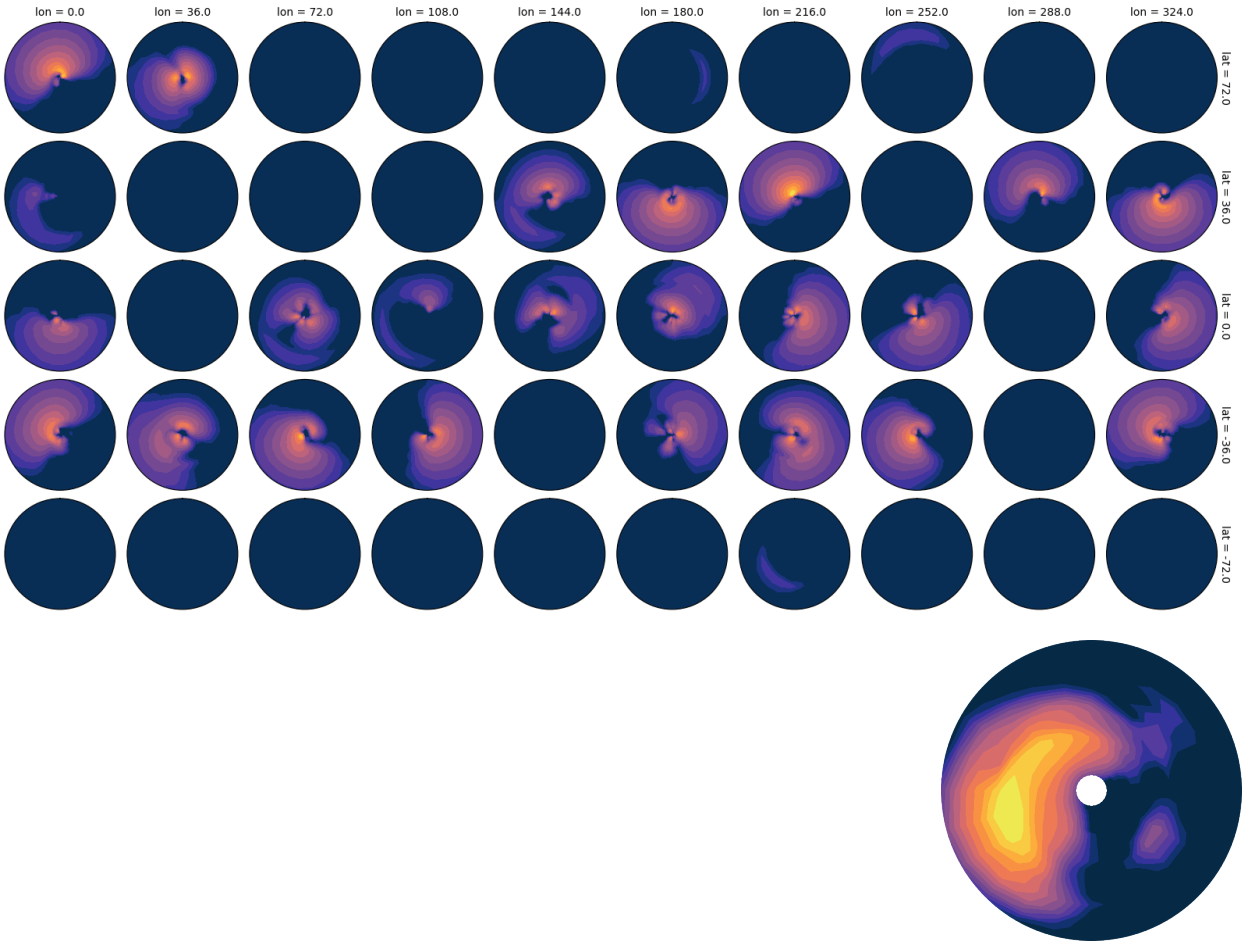
Use the *clean_axis* argument to remove radii and theta grids for a clean overview. This is equivalent to disabling ticks from the axis by calling *ax.set_rticks=[]*, *ax.set_xticks=[]*.

```
In [23]: dset1 = dset.where(dset>0, 1e-5)

In [24]: dset1 = np.log10(dset1)

In [25]: dset1.spec.plot(
```

(continues on next page)

(continued from previous page)

```
   ....:     clean_axis=True,
   ....:     col="lon",
   ....:     row="lat",
   ....:     figsize=(16,8),
   ....:     logradius=False,
   ....:     vmin=0.39,
   ....:     levels=15,
   ....:     extend="both",
   ....:     cmap=cmocean.cm.thermal,
   ....:     add_colorbar=False,
   ....: );
   ....:

In [26]: plt.tight_layout()

In [27]: plt.draw()
```



2.6 Selecting

Wavespectra complements xarray's selecting and interpolating functionality with functions to select and interpolate from `site` (1D) coordinates. The functions are defined in `wavespectra.core.select` module and can be accessed via the `sel` method from *SpecArray* and `SpecDset` accessors.

2.6.1 Nearest neighbour

Select from nearest sites.

```
In [1]: from wavespectra import read_ww3

In [2]: dset = read_ww3("_static/ww3file.nc")

In [3]: ds = dset.spec.sel(
   ...:     lons=[92.01, 92.05, 92.09],
   ...:     lats=[19.812, 19.875, 19.935],
   ...:     method="nearest"
   ...: )
   ...:

In [4]: ds
Out[4]:
<xarray.Dataset>
Dimensions:  (dir: 24, time: 9, site: 3, freq: 25)
Coordinates:
  * dir      (dir) float32 270.0 255.0 240.0 225.0 ... 330.0 315.0 300.0 285.0
  * freq     (freq) float32 0.04118 0.0453 0.04983 ... 0.3352 0.3687 0.4056
  * site     (site) int64 0 1 2
  * time     (time) datetime64[ns] 2014-12-01 2014-12-01T12:00:00 ... 2014-12-05
Data variables:
    dpt      (time, site) float32 dask.array<chunksize=(9, 3), meta=np.ndarray>
    efth     (time, site, freq, dir) float32 dask.array<chunksize=(9, 3, 25, 24),␣
↪meta=np.ndarray>
    lat      (site) float32 dask.array<chunksize=(3,), meta=np.ndarray>
    lon      (site) float32 dask.array<chunksize=(3,), meta=np.ndarray>
    wspd     (time, site) float32 dask.array<chunksize=(9, 3), meta=np.ndarray>
    wdir     (time, site) float32 dask.array<chunksize=(9, 3), meta=np.ndarray>
```

2.6.2 Inverse distance weighting

Interpolate at exact locations via inverse distance weighting algorithm.

```
In [5]: ds = dset.spec.sel(
   ...:     lons=[92.01, 92.05, 92.09],
   ...:     lats=[19.812, 19.875, 19.935],
   ...:     method="idw"
   ...: )
   ...:

In [6]: ds
```

(continues on next page)

(continued from previous page)

```
Out[6]:
<xarray.Dataset>
Dimensions:  (dir: 24, freq: 25, time: 9, site: 3)
Coordinates:
  * dir      (dir) float32 270.0 255.0 240.0 225.0 ... 330.0 315.0 300.0 285.0
  * freq     (freq) float32 0.04118 0.0453 0.04983 ... 0.3352 0.3687 0.4056
  * time     (time) datetime64[ns] 2014-12-01 2014-12-01T12:00:00 ... 2014-12-05
  * site     (site) int64 0 1 2
Data variables:
    dpt      (time, site) float32 dask.array<chunksize=(9, 1), meta=np.ndarray>
    efth     (time, site, freq, dir) float32 dask.array<chunksize=(9, 1, 25, 24),␣
↪meta=np.ndarray>
    lat      (site) float64 19.81 19.88 19.93
    lon      (site) float64 92.01 92.05 92.09
    wspd     (time, site) float32 dask.array<chunksize=(9, 1), meta=np.ndarray>
    wdir     (time, site) float32 dask.array<chunksize=(9, 1), meta=np.ndarray>
```

2.6.3 Bounding box

Select all sites withing bounding box.

```
In [7]: ds = dset.spec.sel(
   ...:     lons=[91, 93],
   ...:     lats=[19, 20],
   ...:     method="bbox"
   ...: )
   ...:

In [8]: ds
Out[8]:
<xarray.Dataset>
Dimensions:  (dir: 24, time: 9, site: 2, freq: 25)
Coordinates:
  * dir      (dir) float32 270.0 255.0 240.0 225.0 ... 330.0 315.0 300.0 285.0
  * freq     (freq) float32 0.04118 0.0453 0.04983 ... 0.3352 0.3687 0.4056
  * site     (site) int64 0 1
  * time     (time) datetime64[ns] 2014-12-01 2014-12-01T12:00:00 ... 2014-12-05
Data variables:
    dpt      (time, site) float32 dask.array<chunksize=(9, 2), meta=np.ndarray>
    efth     (time, site, freq, dir) float32 dask.array<chunksize=(9, 2, 25, 24),␣
↪meta=np.ndarray>
    lat      (site) float32 dask.array<chunksize=(2,), meta=np.ndarray>
    lon      (site) float32 dask.array<chunksize=(2,), meta=np.ndarray>
    wspd     (time, site) float32 dask.array<chunksize=(9, 2), meta=np.ndarray>
    wdir     (time, site) float32 dask.array<chunksize=(9, 2), meta=np.ndarray>
```

Note: When working with large datasets with thousands of spectra sites, it is recommended using *chunks={"site": 1}* option to open dataset lazily in an efficient way for selecting sites. The downside is that accessing entire site-dependent variables (notably `lon` and `lat`) becomes slower, affecting the performance of selecting functions. This can be circumvented by loading these variables without the *chunks* options, and using them as arguments in `sel`, e.g.

```
In [9]: coords = read_ww3("_static/ww3file.nc")[["lon", "lat"]]

In [10]: dset = read_ww3("_static/ww3file.nc", chunks={"site": 1})

In [11]: ds = dset.spec.sel(
   ....:     lons=[92.01, 92.05, 92.09],
   ....:     lats=[19.812, 19.875, 19.935],
   ....:     method="idw",
   ....:     dset_lons=coords.lon,
   ....:     dset_lats=coords.lat
   ....: )
   ....:

In [12]: ds
Out[12]:
<xarray.Dataset>
Dimensions:  (dir: 24, freq: 25, time: 9, site: 3)
Coordinates:
  * dir      (dir) float32 270.0 255.0 240.0 225.0 ... 330.0 315.0 300.0 285.0
  * freq     (freq) float32 0.04118 0.0453 0.04983 ... 0.3352 0.3687 0.4056
  * time     (time) datetime64[ns] 2014-12-01 2014-12-01T12:00:00 ... 2014-12-05
  * site     (site) int64 0 1 2
Data variables:
    dpt      (time, site) float32 dask.array<chunksize=(9, 1), meta=np.ndarray>
    efth     (time, site, freq, dir) float32 dask.array<chunksize=(9, 1, 25, 24),␣
↪meta=np.ndarray>
    lat      (site) float64 19.81 19.88 19.93
    lon      (site) float64 92.01 92.05 92.09
    wspd     (time, site) float32 dask.array<chunksize=(9, 1), meta=np.ndarray>
    wdir     (time, site) float32 dask.array<chunksize=(9, 1), meta=np.ndarray>
```

Help & Reference

- *API documentation*
- *Credits*
- *Support*
- *Contributing*
- *History*
- *Gallery*

2.7 API documentation

General description of modules, objects and functions in wavespectra.

2.7.1 Wavespectra accessors

SpecArray	Extends DataArray with methods to deal with wave spectra.
SpecDataset	Extends xarray's Dataset to deal with wave spectra datasets.

wavespectra.SpecArray

class wavespectra.**SpecArray**(*xarray_obj*)

Extends DataArray with methods to deal with wave spectra.

__init__(*xarray_obj*)

Initialise spec accessor.

Methods

__init__(xarray_obj)	Initialise spec accessor.
celerity([depth])	Wave celerity C from frequency coords.
crsd([theta])	Add description.
dm()	Mean wave direction from the 1st spectral moment Dm.
dp()	Peak wave direction Dp.
dpm()	Peak wave direction Dpm.
dspr()	Directional wave spreading Dspr.
hmax()	Maximum wave height Hmax.
hs([tail])	Spectral significant wave height Hm0.
interp([freq, dir, maintain_m0])	Interpolate onto new spectral basis.
interp_like(other[, maintain_m0])	Interpolate onto coordinates from other spectra.
momd([mom, theta])	Calculate given directional moment.
momf([mom])	Calculate given frequency moment.
oned([skipna])	Returns the one-dimensional frequency spectra.
partition(wsp_darr, wdir_darr, dep_darr[, ...])	Partition wave spectra using Hanson's watershed algorithm.
plot([kind, normalised, logradius, ...])	Plot spectra in polar axis.
scale_by_hs(expr[, hs_min, hs_max, tp_min, ...])	Scale spectra using expression based on Significant Wave Height hs.
split([fmin, fmax, dmin, dmax, rechunk])	Split spectra over freq and/or dir dims.
stats(stats[, fmin, fmax, dmin, dmax, names])	Calculate multiple spectral stats into a Dataset.
sw([mask])	Spectral width parameter by Longuet-Higgins (1975).
swe()	Spectral width parameter by Cartwright and Longuet-Higgins (1956).
tm01()	Mean absolute wave period Tm01.
tm02()	Mean absolute wave period Tm02.
to_energy([standard_name])	Convert from energy density (m2/Hz/degree) into wave energy spectra (m2).
tp([smooth])	Peak wave period Tp.
wavelen([depth])	Wavelength L from frequency coords.

Attributes

dd	Direction resolution float.
df	Frequency resolution numpy array.
dfarr	Frequency resolution DataArray.
dir	Direction DataArray.
freq	Frequency DataArray.

wavespectra.SpecDataset

class wavespectra.**SpecDataset**(*xarray_dset*)

Extends xarray's Dataset to deal with wave spectra datasets.

Plugin functions defined in wavespectra/output/<module> are attached as methods in this accessor class.

__init__(*xarray_dset*)

Methods

__init__(xarray_dset)	
sel(lons, lats[, method, tolerance, ...])	Select stations near or at locations defined by (lons, lats) vector.
to_funwave(filename[, clip])	Write spectra in Funwave format.
to_json(filename[, mode, date_format])	Write spectra in json format.
to_netcdf(filename[, specname, ncformat, ...])	Write spectra in netCDF format using wavespectra conventions.
to_octopus(filename[, site_id, fcut, ...])	Save spectra in Octopus format.
to_orcaflex(model[, minEnergy])	Writes the spectrum to an Orcaflex model
to_swan(filename[, append, id, ntime])	Write spectra in SWAN ASCII format.
to_ww3(filename[, ncformat, compress])	Save spectra in native WW3 netCDF format.

* The two accessors are attached to the respective xarray objects via the *spec* namespace.

2.7.2 SpecArray

All methods in *SpecArray* accessor are also available from *SpecDataset*.

Integrated spectral parameters

SpecArray.hs	Spectral significant wave height Hm0.
SpecArray.hmax	Maximum wave height Hmax.
SpecArray.tp	Peak wave period Tp.
SpecArray.tm01	Mean absolute wave period Tm01.
SpecArray.tm02	Mean absolute wave period Tm02.
SpecArray.dpm	Peak wave direction Dpm.
SpecArray.dp	Peak wave direction Dp.
SpecArray.dm	Mean wave direction from the 1st spectral moment Dm.
SpecArray.dspr	Directional wave spreading Dspr.
SpecArray.swe	Spectral width parameter by Cartwright and Longuet-Higgins (1956).
SpecArray.sw	Spectral width parameter by Longuet-Higgins (1975).

wavespectra.SpecArray.hs

SpecArray.**hs**(*tail=True*)

Spectral significant wave height Hm0.

Args:

- tail (bool): if True fit high-frequency tail before integrating spectra.

wavespectra.SpecArray.hmax

SpecArray.**hmax**()

Maximum wave height Hmax.

hmax is the most probably value of the maximum individual wave height for each sea state. Note that maximum wave height can be higher (but not by much since the probability density function is rather narrow).

Reference:

- Holthuijsen LH (2005). Waves in oceanic and coastal waters (page 82).

wavespectra.SpecArray.tp

SpecArray.**tp**(*smooth=True*)

Peak wave period Tp.

Args:

- smooth (bool): True for the smooth wave period, False for the discrete period corresponding to the maxima in the frequency spectra.

wavespectra.SpecArray.tm01

SpecArray.**tm01**()

Mean absolute wave period Tm01.

True average period from the 1st spectral moment.

wavespectra.SpecArray.tm02

SpecArray.**tm02**()

Mean absolute wave period Tm02.

Average period of zero up-crossings (Zhang, 2011).

wavespectra.SpecArray.dpm

SpecArray.**dpm**()

Peak wave direction Dpm.

Note From WW3 Manual:

- peak wave direction, defined like the mean direction, using the frequency/wavenumber bin containing of the spectrum F(k) that contains the peak frequency only.

wavespectra.SpecArray.dp

SpecArray.**dp**()

Peak wave direction Dp.

Defined as the direction where the energy density of the frequency-integrated spectrum is maximum.

wavespectra.SpecArray.dm

SpecArray.**dm**()

Mean wave direction from the 1st spectral moment Dm.

wavespectra.SpecArray.dspr

SpecArray.**dspr**()

Directional wave spreading Dspr.

The one-sided directional width of the spectrum.

wavespectra.SpecArray.swe

SpecArray.**swe**()

Spectral width parameter by Cartwright and Longuet-Higgins (1956).

Represents the range of frequencies where the dominant energy exists.

Reference:

- Cartwright and Longuet-Higgins (1956). The statistical distribution of maxima of a random function. Proc. R. Soc. A237, 212-232.

wavespectra.SpecArray.sw

SpecArray.**sw**(*mask=nan*)

Spectral width parameter by Longuet-Higgins (1975).

Represents energy distribution over entire frequency range.

Args:

- mask (float): value for missing data in output

Reference:

- Longuet-Higgins (1975). On the joint distribution of the periods and amplitudes of sea waves. JGR, 80, 2688-2694.

Spectral partitioning

SpecArray.split	Split spectra over freq and/or dir dims.
SpecArray.partition	Partition wave spectra using Hanson's watershed algorithm.

wavespectra.SpecArray.split

SpecArray.**split**(*fmin=None*, *fmax=None*, *dmin=None*, *dmax=None*, *rechunk=True*)

Split spectra over freq and/or dir dims.

Args:

- fmin (float): lowest frequency to split spectra, by default the lowest.
- fmax (float): highest frequency to split spectra, by default the highest.
- dmin (float): lowest direction to split spectra at, by default min(dir).
- dmax (float): highest direction to split spectra at, by default max(dir).
- rechunk (bool): Rechunk split dims so there is one single chunk.

Note:

- Spectra are interpolated at *fmin* / *fmax* if they are not in self.freq.
- Recommended rechunk==True so ufuncs with freq/dir as core dims will work.

wavespectra.SpecArray.partition

SpecArray.**partition**(*wsp_darr*, *wdir_darr*, *dep_darr*, *swells=3*, *agefac=1.7*, *wscut=0.3333*)

Partition wave spectra using Hanson's watershed algorithm.

This method is not lazy, make sure array will fit into memory.

Args:

- wsp_darr (DataArray): wind speed (m/s).
- wdir_darr (DataArray): Wind direction (degree).
- dep_darr (DataArray): Water depth (m).
- swells (int): Number of swell partitions to compute.
- agefac (float): Age factor.
- wscut (float): Wind speed cutoff.

Returns:

- part_spec (SpecArray): partitioned spectra with one extra dimension representig partition number.

Note:

- Input DataArrays must have same non-spectral dims as SpecArray.

References:

- Hanson, Jeffrey L., et al. "Pacific hindcast performance of three numerical wave models." JTECH 26.8 (2009): 1614-1633.

Other methods

SpecArray.momf	Calculate given frequency moment.
SpecArray.momd	Calculate given directional moment.
SpecArray.wavelen	Wavelength L from frequency coords.
SpecArray.celerity	Wave celerity C from frequency coords.
SpecArray.stats	Calculate multiple spectral stats into a Dataset.
SpecArray.plot	Plot spectra in polar axis.
SpecArray.scale_by_hs	Scale spectra using expression based on Significant Wave Height hs.
SpecArray.oned	Returns the one-dimensional frequency spectra.
SpecArray.to_energy	Convert from energy density (m2/Hz/degree) into wave energy spectra (m2).
SpecArray.interp	Interpolate onto new spectral basis.
SpecArray.interp_like	Interpolate onto coordinates from other spectra.

wavespectra.SpecArray.momf

SpecArray.**momf**(*mom=0*)

Calculate given frequency moment.

wavespectra.SpecArray.momd

SpecArray.**momd**(*mom=0*, *theta=90.0*)

Calculate given directional moment.

wavespectra.SpecArray.wavelen

SpecArray.**wavelen**(*depth=None*)

Wavelength L from frequency coords.

Args:

- depth (float): Water depth, use deep water approximation by default.

Returns;

- L: ndarray of same shape as freq with wavelength for each frequency.

wavespectra.SpecArray.celerity

SpecArray.**celerity**(*depth=None*)

Wave celerity C from frequency coords.

Args:

- depth (float): Water depth, use deep water approximation by default.

Returns;

- C: ndarray of same shape as freq with wave celerity for each frequency.

wavespectra.SpecArray.stats

SpecArray.**stats**(*stats*, *fmin=None*, *fmax=None*, *dmin=None*, *dmax=None*, *names=None*)

Calculate multiple spectral stats into a Dataset.

Args:

- stats (list): strings specifying stats to be calculated. (dict): keys are stats names, vals are dicts with kwargs to use with corresponding method.
- fmin (float): lower frequencies for splitting spectra before calculating stats.
- fmax (float): upper frequencies for splitting spectra before calculating stats.
- dmin (float): lower directions for splitting spectra before calculating stats.
- dmax (float): upper directions for splitting spectra before calculating stats.
- names (list): strings to rename each stat in output Dataset.

Returns:

- Dataset with all spectral statistics specified.

Note:

- All stats names must correspond to methods implemented in this class.
- If names is provided, its length must correspond to the length of stats.

wavespectra.SpecArray.plot

`SpecArray.plot`(*kind='contourf'*, *normalised=True*, *logradius=True*, *as_period=False*, *rmin=None*, *rmax=None*, *show_theta_labels=True*, *show_radii_labels=True*, *radii_ticks=None*, *radii_labels_angle=22.5*, *radii_labels_size=8*, *cbar_ticks=[0.01, 0.1, 1.0]*, *cmap='RdBu_r'*, *extend='neither'*, *efth_min=0.001*, ***kwargs*)

Plot spectra in polar axis.

Args:

- kind (str): Plot kind, one of (*contourf*, *contour*, *pcolormesh*).
- normalised (bool): Show efth normalised between 0 and 1.
- logradius (bool): Set log radii.
- as_period (bool): Set radii as wave period instead of frequency.
- rmin (float): Minimum value to clip the radius axis.
- rmax (float): Maximum value to clip the radius axis.
- show_theta_labels (bool): Show direction tick labels.
- show_radii_labels (bool): Show radii tick labels.
- radii_ticks (array): Tick values for radii.
- radii_labels_angle (float): Polar angle at which radii labels are positioned.
- radii_labels_size (float): Fontsize for radii labels.
- cbar_ticks (array): Tick values for colorbar.
- cmap (str, obj): Colormap to use.
- efth_min (float): Clip energy density below this value.
- kwargs: All extra kwargs are passed to the plotting method defined by *kind*.

Returns:

- pobj: The xarray object returned by calling *da.plot.{kind}(**kwargs)*.

Note:

- If normalised==True, contourf uses a logarithmic colour scale by default.
- Plot and axes can be redefined from the returned xarray object.
- Xarray uses the *sharex*, *sharey* args to control which panels receive axis labels. In order to set labels for all panels, set these to *False*.
- Masking of low values can be done in contourf by setting *efth_min* larger than the lowest contour level along with *extend* set to "neither" or "min".

wavespectra.SpecArray.scale_by_hs

SpecArray.**scale_by_hs**(*expr*, *hs_min=-inf*, *hs_max=inf*, *tp_min=-inf*, *tp_max=inf*, *dpm_min=-inf*, *dpm_max=inf*)

Scale spectra using expression based on Significant Wave Height hs.

Args:

- expr (str): expression to apply, e.g. '0.13*hs + 0.02'.
- **hs_min, hs_max, tp_min, tp_max, dpm_min, dpm_max (float): Ranges of hs,** tp and dpm over which the scaling defined by *expr* is applied.

wavespectra.SpecArray.oned

SpecArray.**oned**(*skipna=True*)

Returns the one-dimensional frequency spectra.

Direction dimension is dropped after integrating.

Args:

- skipna (bool): choose it to skip nans when integrating spectra. This is the default behaviour for sum() in DataArray. Notice it converts masks, where the entire array is nan, into zero.

wavespectra.SpecArray.to_energy

SpecArray.**to_energy**(*standard_name='sea_surface_wave_directional_energy_spectra'*)

Convert from energy density (m2/Hz/degree) into wave energy spectra (m2).

wavespectra.SpecArray.interp

SpecArray.**interp**(*freq=None*, *dir=None*, *maintain_m0=True*)

Interpolate onto new spectral basis.

Args:

- freq (DataArray, 1darray): Frequencies of interpolated spectra (Hz).
- dir (DataArray, 1darray): Directions of interpolated spectra (deg).
- maintain_m0 (bool): Ensure variance is conserved in interpolated spectra.

Returns:

- dsi (DataArray): Regridded spectra.

Note:

- All freq below lowest freq are interpolated assuming $E_d(f = 0) = 0$.
- $Ed(f)$ is set to zero for new freq above the highest freq in dset.
- Only the 'linear' method is currently supported.

wavespectra.SpecArray.interp_like

SpecArray.**interp_like**(*other*, *maintain_m0=True*)

Interpolate onto coordinates from other spectra.

Args:

- other (Dataset, DataArray): Spectra defining new spectral basis.
- maintain_m0 (bool): Ensure variance is conserved in interpolated spectra.

Returns:

- dsi (DataArray): Regridded spectra.

2.7.3 SpecDataset

SpecDataset.sel	Select stations near or at locations defined by (lons, lats) vector.

wavespectra.SpecDataset.sel

SpecDataset.**sel**(*lons*, *lats*, *method='idw'*, *tolerance=2.0*, *dset_lons=None*, *dset_lats=None*, ***kwargs*)

Select stations near or at locations defined by (lons, lats) vector.

Args:

- lons (list): Longitude values of locations to select.
- lats (list): Latitude values of locations to select.
- **method (str): Method to use for inexact matches:**
  - idw: Inverse distance weighting selection.
  - nearest: Nearest site selection.
  - bbox: Sites inside bbox [min(lons), min(lats)], [max(lons), max(lats)].
  - None: Only exact matches.
- tolerance (float): Maximum distance between locations and original stations for inexact matches.
- dset_lons (array): Longitude of stations in dset, not required but could help improove speed.
- dset_lats (array): Latitude of stations in dset, not required but could help improove speed.
- kwargs: Extra keywargs to pass to the respective sel function (i.e., *sel_nearest*, *sel_idw*).

Return:

- dset (SpecDataset): Stations Dataset selected at locations defined by zip(lons, lats).

Note:

- *tolerance* behaves differently with methods 'idw' and 'nearest'. In 'idw' sites with no neighbours within *tolerance* are masked whereas in 'nearest' an exception is raised.
- *dset_lons*, *dset_lats* are not required but can improve performance when *dset* is chunked with site=1 (expensive to access site coords) and improve precision if projected coors are provided at high latitudes.

Output methods described in the *Output functions* section: *to_swan* *to_netcdf* *to_octopus* *to_ww3* *to_json*

2.7.4 Input functions

NetCDF-based input functions

read_ww3	Read Spectra from WAVEWATCHIII native netCDF format.
read_ncswan	Read Spectra from SWAN native netCDF format.
read_wwm	Read Spectra from WWMII native netCDF format.
read_netcdf	Read Spectra from generic netCDF format.
read_era5	Read Spectra from ECMWF ERA5 netCDF format.

* These functions also support Zarr files

Other input functions

read_swan	Read Spectra from SWAN ASCII file.
read_triaxys	Read spectra from TRIAXYS wave buoy ASCII files.
read_spotter	Read Spectra from Spotter file.
read_octopus	Read spectra from Octopus file format.
read_dataset	Format and attach SpecArray accessor to an existing xarray dataset.
read_ndbc	Read Spectra from NDBC netCDF format.
read_json	Read Spectra from json.

Convenience SWAN ASCII input functions

input.swan.read_swans	Read multiple SWAN ASCII files into single Dataset.
input.swan.read_hotswan	Read partial SWAN hotfiles into single gridded hotfile Dataset.
input.swan.read_swanow	Read SWAN nowcast from fileglob, keep overlapping dates from most recent files.

2.7.5 Output functions

SpecDataset.to_swan	Write spectra in SWAN ASCII format.
SpecDataset.to_netcdf	Write spectra in netCDF format using wavespectra conventions.
SpecDataset.to_octopus	Save spectra in Octopus format.
SpecDataset.to_ww3	Save spectra in native WW3 netCDF format.
SpecDataset.to_json	Write spectra in json format.

2.7.6 Spectral reconstruction

Spectral reconstruction functionality is under development. There are functions available to fit parametric spectrum shapes from wave partitions but the construct api is not properly established.

construct.jonswap	Constructs JONSWAP spectra.
construct.ochihubble	Construct OCHIHUBBLE spectra.
construct.helpers.spread	Generic spreading function.
construct.helpers.arrange_inputs	Check all inputs are same shape and add frequency and direction dims.
construct.helpers.make_dataset	Package spectral matrix to xarray.
construct.helpers.check_coordinates	Check coordinates are consistent with parameter.

wavespectra.construct.jonswap

wavespectra.construct.**jonswap**(*tp*, *dp*, *alpha*, *gamma=3.3*, *dspr=20*, *freqs=array([0.04, 0.06, 0.08, 0.1, 0.12, 0.14, 0.16, 0.18, 0.2, 0.22, 0.24, 0.26, 0.28, 0.3, 0.32, 0.34, 0.36, 0.38, 0.4, 0.42, 0.44, 0.46, 0.48, 0.5, 0.52, 0.54, 0.56, 0.58, 0.6, 0.62, 0.64, 0.66, 0.68, 0.7, 0.72, 0.74, 0.76, 0.78, 0.8, 0.82, 0.84, 0.86, 0.88, 0.9, 0.92, 0.94, 0.96, 0.98])*, *dirs=array([0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350])*, *coordinates=[]*, *sumpart=True*)

Constructs JONSWAP spectra.

wavespectra.construct.ochihubble

wavespectra.construct.**ochihubble**(*hs*, *tp*, *L*, *dp*, *dspr*, *freqs=array([0.02, 0.04, 0.06, 0.08, 0.1, 0.12, 0.14, 0.16, 0.18, 0.2, 0.22, 0.24, 0.26, 0.28, 0.3, 0.32, 0.34, 0.36, 0.38, 0.4, 0.42, 0.44, 0.46, 0.48, 0.5, 0.52, 0.54, 0.56, 0.58, 0.6, 0.62, 0.64, 0.66, 0.68, 0.7, 0.72, 0.74, 0.76, 0.78, 0.8, 0.82, 0.84, 0.86, 0.88, 0.9, 0.92, 0.94, 0.96, 0.98])*, *dirs=array([0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350])*, *coordinates=[('part', [0, 1])]*, *sumpart=True*)

Construct OCHIHUBBLE spectra.

wavespectra.construct.helpers.spread

wavespectra.construct.helpers.**spread**(*dp_matrix*, *dspr_matrix*, *dirs*)

Generic spreading function.

Args:
dp_matrix: dspr_matrix: dirs:

Returns:
G1:

Note:
Function defined such that int{G1 dtheta}=1*

wavespectra.construct.helpers.arrange_inputs

wavespectra.construct.helpers.**arrange_inputs**(**args*)

Check all inputs are same shape and add frequency and direction dims.

wavespectra.construct.helpers.make_dataset

wavespectra.construct.helpers.**make_dataset**(*spec*, *freqs*, *dirs*, *coordinates=[]*)

Package spectral matrix to xarray.

Args:
spec: freqs: dirs: coordinates:

Returns:
dset: SpecDset object

wavespectra.construct.helpers.check_coordinates

wavespectra.construct.helpers.**check_coordinates**(*param*, *coordinates*)

Check coordinates are consistent with parameter.

Args:
param: coordinates:

2.7.7 Internal core functions and objects

watershed module

core.watershed.partition	Watershed partitioning.
core.watershed.nppart	Watershed partition on a numpy array.
core.watershed.hs	Significant wave height Hmo.
core.watershed.frequency_resolution	Frequency resolution array.
core.watershed.inflection	Points of inflection in smoothed frequency spectra.

wavespectra.core.watershed.partition

wavespectra.core.watershed.**partition**(*dset*, *wspd='wspd'*, *wdir='wdir'*, *dpt='dpt'*, *swells=3*, *agefac=1.7*, *wscut=0.3333*)

Watershed partitioning.

Args:

- dset (xr.DataArray, xr.Dataset): Spectra array or dataset in wavespectra convention.
- wspd (xr.DataArray, str): Wind speed DataArray or variable name in dset.
- wdir (xr.DataArray, str): Wind direction DataArray or variable name in dset.
- dpt (xr.DataArray, str): Depth DataArray or the variable name in dset.
- swells (int): Number of swell partitions to compute.
- agefac (float): Age factor.

- wscut (float): Wind speed cutoff.

Returns:

- dspart (xr.Dataset): Partitioned spectra dataset with extra dimension.

References:

- **Hanson, Jeffrey L., et al. "Pacific hindcast performance of three** numerical wave models." JTECH 26.8 (2009): 1614-1633.

wavespectra.core.watershed.nppart

`wavespectra.core.watershed.`**`nppart`**(*spectrum*, *freq*, *dir*, *wspd*, *wdir*, *dpt*, *swells=3*, *agefac=1.7*, *wscut=0.3333*)

Watershed partition on a numpy array.

Args:

- spectrum (2darray): Wave spectrum array with shape (nf, nd).
- freq (1darray): Wave frequency array with shape (nf).
- dir (1darray): Wave direction array with shape (nd).
- wspd (float): Wind speed.
- wdir (float): Wind direction.
- dpt (float): Water depth.
- swells (int): Number of swell partitions to compute.
- agefac (float): Age factor.
- wscut (float): Wind speed cutoff.

Returns:

- specpart (3darray): Wave spectrum partitions with shape (np, nf, nd).

wavespectra.core.watershed.hs

`wavespectra.core.watershed.`**`hs`**(*spectrum*, *freq*, *dir*, *tail=True*)

Significant wave height Hmo.

Args:

- spectrum (2darray): wave spectrum array.
- freq (1darray): wave frequency array.
- dir (1darray): wave direction array.
- tail (bool): if True fit high-frequency tail before integrating spectra.

wavespectra.core.watershed.frequency_resolution

wavespectra.core.watershed.**frequency_resolution**(*freq*)

Frequency resolution array.

Args:

- freq (1darray): Frequencies to calculate resolutions from with shape (nf).

Returns:

- df (1darray): Resolution for pairs of adjacent frequencies with shape (nf-1).

wavespectra.core.watershed.inflection

wavespectra.core.watershed.**inflection**(*spectrum*, *freq*, *dfres=0.01*, *fmin=0.05*)

Points of inflection in smoothed frequency spectra.

Args:

- fdspec (ndarray): freq-dir 2D specarray.
- dfres (float): used to determine length of smoothing window.
- fmin (float): minimum frequency for looking for minima/maxima.

attributes module

core.attributes.set_spec_attributes	Standarise CF attributes in specarray variables
core.attributes.attrs	

wavespectra.core.attributes.set_spec_attributes

wavespectra.core.attributes.**set_spec_attributes**(*dset*)

Standarise CF attributes in specarray variables

wavespectra.core.attributes.attrs

wavespectra.core.attributes.attrs = {'ATTRS': {'cycle': {'standard_name': 'forecast_reference_time'}, 'dir': {'standard_name': 'sea_surface_wave_from_direction', 'units': 'degree'}, 'dm': {'standard_name': 'sea_surface_wave_mean_from_direction', 'units': 'degree'}, 'dp': {'standard_name': 'sea_surface_wave_from_direction_at_variance_spectral_density_maximum', 'units': 'degree'}, 'dpm': {'standard_name': 'sea_surface_wave_from_direction_at_variance_spectral_density_maximum', 'units': 'degree'}, 'dpt': {'standard_name': 'sea_floor_depth_below_sea_surface', 'units': 'm'}, 'dspr': {'standard_name': 'sea_surface_wave_directional_spread', 'units': 'degree'}, 'efth': {'standard_name': 'sea_surface_wave_directional_variance_spectral_density', 'units': 'm2 s degree-1'}, 'freq': {'standard_name': 'sea_surface_wave_frequency', 'units': 'Hz'}, 'hmax': {'standard_name': 'sea_surface_wave_maximum_height', 'units': 'm'}, 'hs': {'_ipython_canary_method_should_not_exist_': {}, 'standard_name': 'sea_surface_wave_significant_height', 'units': 'm'}, 'lat': {'standard_name': 'latitude', 'units': 'degrees_north'}, 'lon': {'standard_name': 'longitude', 'units': 'degrees_east'}, 'part': {'standard_name': 'spectral_partition_number', 'units': ''}, 'site': {'standard_name': 'site', 'units': ''}, 'sw': {'standard_name': 'sea_surface_wave_spectral_width', 'units': ''}, 'swe': {'standard_name': 'sea_surface_wave_spectral_width', 'units': ''}, 'time': {'standard_name': 'time'}, 'tm01': {'standard_name': 'sea_surface_wave_mean_period_from_variance_spectral_density_first_frequency_moment', 'units': 's'}, 'tm02': {'standard_name': 'sea_surface_wave_mean_period_from_variance_spectral_density_second_frequency_moment', 'units': 's'}, 'tp': {'standard_name': 'sea_surface_wave_period_at_variance_spectral_density_maximum', 'units': 's'}, 'wdir': {'standard_name': 'wind_from_direction_at_10m_above_ground_level', 'units': 'degree'}, 'wspd': {'standard_name': 'wind_speed_at_10m_above_ground_level', 'units': 'm s-1'}}, 'CYCLENAME': 'cycle', 'DEPNAME': 'dpt', 'DIRNAME': 'dir', 'FREQNAME': 'freq', 'LATNAME': 'lat', 'LONNAME': 'lon', 'PARTNAME': 'part', 'SITENAME': 'site', 'SPECNAME': 'efth', 'TIMENAME': 'time', 'WDIRNAME': 'wdir', 'WSPDNAME': 'wspd', '__qualname__': {}, '__sphinx_mock__': {}}

npstats

core.npstats.hs	Significant wave height Hmo.
core.npstats.dpm_gufunc	dpm_gufunc(x1, x2, x3, /, out=None, *, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj, axes, axis])
core.npstats.dp_gufunc	dp_gufunc(x1, x2, /, out=None, *, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj, axes, axis])
core.npstats.tps_gufunc	tps_gufunc(x1, x2, x3, /, out=None, *, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj, axes, axis])
core.npstats.tp_gufunc	tp_gufunc(x1, x2, x3, /, out=None, *, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj, axes, axis])

wavespectra.core.npstats.hs

wavespectra.core.npstats.**hs**(*spectrum*, *freq*, *dir*, *tail=True*)

Significant wave height Hmo.

Args:

- spectrum (2darray): wave spectrum array.
- freq (1darray): wave frequency array.
- dir (1darray): wave direction array.
- tail (bool): if True fit high-frequency tail before integrating spectra.

wavespectra.core.npstats.dpm_gufunc

wavespectra.core.npstats.**dpm_gufunc = <ufunc 'dpm_gufunc'>**

dpm_gufunc(x1, x2, x3, /, out=None, *, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj, axes, axis])

Mean direction at the peak wave period Dpm.

Args:

- ipeak (int): Index of the maximum energy density in the frequency spectrum E(f).
- momsin (1darray): Sin component of the 1st directional moment.
- momcos (1darray): Cos component of the 1st directional moment.

Returns:

- dpm (float): Mean direction at the frequency peak of the spectrum.

wavespectra.core.npstats.dp_gufunc

wavespectra.core.npstats.**dp_gufunc = <ufunc 'dp_gufunc'>**

dp_gufunc(x1, x2, /, out=None, *, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj, axes, axis])

Peak wave direction Dp.

Args:

- ipeak (int): Index of the maximum energy density in the frequency spectrum E(f).
- dir (1darray): Wave direction array.

Returns:

- dp (float): Direction of the maximum energy density in the frequency-integrated spectrum.

wavespectra.core.npstats.tps_gufunc

wavespectra.core.npstats.**tps_gufunc = <ufunc 'tps_gufunc'>**

tps_gufunc(x1, x2, x3, /, out=None, *, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj, axes, axis])

Smooth peak wave period Tp.

Args:

- ipeak (int): Index of the maximum energy density in frequency spectrum E(f).
- spectrum (1darray): Direction-integrated wave spectrum array E(f).
- freq (1darray): Wave frequency array.

Returns:

- tp (float): Period of the maximum energy density in the smooth spectrum.

Note:

- The smooth peak period is the peak of a parabolic fit around the spectral peak. It is the period commonly defined in SWAN and WW3 model output.

wavespectra.core.npstats.tp_gufunc

wavespectra.core.npstats.**tp_gufunc = <ufunc 'tp_gufunc'>**

tp_gufunc(x1, x2, x3, /, out=None, *, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj, axes, axis])

Peak wave period Tp.

Args:

- ipeak (int): Index of the maximum energy density in frequency spectrum E(f).
- spectrum (1darray): Frequency wave spectrum array E(f).
- freq (1darray): Wave frequency array.

Returns:

- tp (float): Period of the maximum energy density in the frequency spectrum.

Note:

- Arg spectrum is only defined so the signature is consistent with tps function.

xrstats

core.xrstats.peak_wave_direction	Peak wave direction Dp.
core.xrstats.mean_direction_at_peak_wave_pe	Mean direction at the peak wave period Dpm.
core.xrstats.peak_wave_period	Smooth Peak wave period Tp.

wavespectra.core.xrstats.peak_wave_direction

wavespectra.core.xrstats.**peak_wave_direction**(*dset*)

Peak wave direction Dp.

Args:

- dset (xr.DataArray, xr.Dataset): Spectra array or dataset in wavespectra convention.

Returns:

- dp (xr.DataArray): Peak wave direction data array.

wavespectra.core.xrstats.mean_direction_at_peak_wave_period

wavespectra.core.xrstats.**mean_direction_at_peak_wave_period**(*dset*)

Mean direction at the peak wave period Dpm.

Args:

- dset (xr.DataArray, xr.Dataset): Spectra array or dataset in wavespectra convention.

Returns:

- dpm (xr.DataArray): Mean direction at the peak wave period data array.

Note from WW3 Manual:

- Peak wave direction, defined like the mean direction, using the freq/wavenum bin containing of the spectrum F(k) that contains the peak frequency only.

wavespectra.core.xrstats.peak_wave_period

wavespectra.core.xrstats.**peak_wave_period**(*dset*, *smooth=True*)

Smooth Peak wave period Tp.

Args:

- dset (xr.DataArray, xr.Dataset): Spectra array or dataset in wavespectra convention.
- smooth (bool): Choose between the smooth (tps) or the raw (tp) peak wave period type.

Returns:

- tp (xr.DataArray): Peak wave period data array.

select module

core.select.sel_nearest	Select sites from nearest distance.
core.select.sel_bbox	Select sites within bbox.
core.select.sel_idw	Select sites from inverse distance weighting.
core.select.Coordinates.distance	Distance between each station in (dset_lons, dset_lats) and site (lon, lat).
core.select.Coordinates.nearer	Nearer stations in (dset_lons, dset_lats) to site (lon, lat).
core.select.Coordinates.nearest	Nearest station in (dset_lons, dset_lats) to site (lon, lat).

wavespectra.core.select.sel_nearest

wavespectra.core.select.**sel_nearest**(*dset*, *lons*, *lats*, *tolerance=2.0*, *unique=False*, *exact=False*, *dset_lons=None*, *dset_lats=None*)

Select sites from nearest distance.

Args:

dset (Dataset): Stations SpecDataset to select from. lons (array): Longitude of sites to interpolate spectra at. lats (array): Latitude of sites to interpolate spectra at. tolerance (float): Maximum distance to use site for interpolation. unique (bool): Only returns unique sites in case of repeated inexact matches. exact (bool): Require exact matches. dset_lons (array): Longitude of stations in dset. dset_lats (array): Latitude of stations in dset.

Returns:

Selected SpecDataset at locations defined by (lons, lats).

Note:

Args *dset_lons*, *dset_lats* are not required but can improve performance when
dset is chunked with site=1 (expensive to access station coordinates) and improve precision if projected coordinates are provided at high latitudes.

wavespectra.core.select.sel_bbox

wavespectra.core.select.**sel_bbox**(*dset*, *lons*, *lats*, *tolerance=0.0*, *dset_lons=None*, *dset_lats=None*)

Select sites within bbox.

Args:

dset (Dataset): Stations SpecDataset to select from. lons (array): Longitude of sites to interpolate spectra at. lats (array): Latitude of sites to interpolate spectra at. tolerance (float): Extend bbox extents by. dset_lons (array): Longitude of stations in dset. dset_lats (array): Latitude of stations in dset.

Returns:

Selected SpecDataset within bbox defined by:
lower-left=[min(lons), min(lats)], upper-right=[max(lons), max(lats)].

Note:

Args *dset_lons*, *dset_lats* are not required but can improve performance when
dset is chunked with site=1 (expensive to access station coordinates) and improve precision if projected coordinates are provided at high latitudes.

wavespectra.core.select.sel_idw

wavespectra.core.select.**sel_idw**(*dset*, *lons*, *lats*, *tolerance=2.0*, *max_sites=4*, *dset_lons=None*, *dset_lats=None*)

Select sites from inverse distance weighting.

Args:

dset (Dataset): Stations SpecDataset to interpolate from. lons (array): Longitude of sites to interpolate spectra at. lats (array): Latitude of sites to interpolate spectra at. tolerance (float): Maximum distance to use site for interpolation. max_sites (int): Maximum number of neighbour sites to use for interpolation. dset_lons (array): Longitude of stations in dset. dset_lats (array): Latitude of stations in dset.

Returns:

Selected SpecDataset at locations defined by (lons, lats).

Note:

Args *dset_lons*, *dset_lats* are not required but can improve performance when
dset is chunked with site=1 (expensive to access station coordinates) and improve precision if projected coordinates are provided at high latitudes.

wavespectra.core.select.Coordinates.distance

`Coordinates.distance(lon, lat)`

Distance between each station in (dset_lons, dset_lats) and site (lon, lat).

Args:
lon (float): Longitude to locate from lons. lat (float): Latitude to locate from lats.

Returns:
List of distances between each station and site.

wavespectra.core.select.Coordinates.nearer

`Coordinates.nearer(lon, lat, tolerance=inf, max_sites=None)`

Nearer stations in (dset_lons, dset_lats) to site (lon, lat).

Args:
lon (float): Longitude of of station to locate from lons. lat (float): Latitude of of station to locate from lats. tolerance (float): Maximum distance for scanning neighbours. max_sites (int): Maximum number of neighbours.

Returns:

Indices and distances of up to *max_sites* neighbour stations not farther from
tolerance, ordered from closer to farthest station.

wavespectra.core.select.Coordinates.nearest

`Coordinates.nearest(lon, lat)`

Nearest station in (dset_lons, dset_lats) to site (lon, lat).

Args:
lon (float): Longitude to locate from lons. lat (float): Latitude to locate from lats.

Returns:
Index and distance of closest station.

utils module

core.utils.wavelen	Wavelength L.
core.utils.wavenuma	Chen and Thomson wavenumber approximation.
core.utils.celerity	Wave celerity C.
core.utils.dnum_to_datetime	Convert from numeric date to datetime.
core.utils.to_nautical	Convert from cartesian to nautical angle.
core.utils.unique_indices	Remove duplicate indices from dataset.
core.utils.unique_times	Remove duplicate times from dataset.
core.utils.to_datetime	Convert Datetime64 date to datatime.
core.utils.spddir_to_uv	Converts (spd, dir) to (u, v).
core.utils.uv_to_spddir	Converts (u, v) to (spd, dir).
core.utils.interp_spec	Interpolate onto new spectral basis.
core.utils.flatten_list	Flatten list of lists
core.utils.regrid_spec	Regrid spectra onto new spectral basis.

wavespectra.core.utils.wavelen

wavespectra.core.utils.**wavelen**(*freq*, *depth=None*)

Wavelength L.

Args:

- freq (ndarray): Frequencies (Hz) for calculating L.
- depth (float): Water depth, use deep water approximation by default.

Returns;

- L: ndarray of same shape as freq with wavelength for each frequency.

wavespectra.core.utils.wavenuma

wavespectra.core.utils.**wavenuma**(*ang_freq*, *water_depth*)

Chen and Thomson wavenumber approximation.

wavespectra.core.utils.celerity

wavespectra.core.utils.**celerity**(*freq*, *depth=None*)

Wave celerity C.

Args:

- freq (ndarray): Frequencies (Hz) for calculating C.
- depth (float): Water depth, use deep water approximation by default.

Returns;

- C: ndarray of same shape as freq with wave celerity for each frequency.

wavespectra.core.utils.dnum_to_datetime

wavespectra.core.utils.**dnum_to_datetime**(*dnum*)

Convert from numeric date to datetime.

wavespectra.core.utils.to_nautical

wavespectra.core.utils.**to_nautical**(*ang*)

Convert from cartesian to nautical angle.

wavespectra.core.utils.unique_indices

wavespectra.core.utils.**unique_indices**(*ds*, *dim='time'*)

Remove duplicate indices from dataset.

Args:

- ds (Dataset, DataArray): Dataset to remove duplicate indices from.
- dim (str): Dimension to remove duplicate indices from.

Returns:

dsout (Dataset, DataArray): Dataset with duplicate indices along dim removed.

wavespectra.core.utils.unique_times

wavespectra.core.utils.**unique_times**(*ds*)

Remove duplicate times from dataset.

wavespectra.core.utils.to_datetime

wavespectra.core.utils.**to_datetime**(*np64*)

Convert Datetime64 date to datatime.

wavespectra.core.utils.spddir_to_uv

wavespectra.core.utils.**spddir_to_uv**(*spd*, *direc*, *coming_from=False*)

Converts (spd, dir) to (u, v).

Args:

spd (array): magnitudes to convert. direc (array): directions to convert (degree). coming_from (bool): True if directions in coming-from convention,

False if in going-to.

Returns:

u (array): eastward wind component. v (array): northward wind component.

wavespectra.core.utils.uv_to_spddir

wavespectra.core.utils.**uv_to_spddir**(*u*, *v*, *coming_from=False*)

Converts (u, v) to (spd, dir).

Args:

u (array): eastward wind component. v (array): northward wind component. coming_from (bool): True for output directions in coming-from convention,

False for going-to.

Returns:

mag (array): magnitudes. direc (array): directions (degree).

wavespectra.core.utils.interp_spec

wavespectra.core.utils.**interp_spec**(*inspec*, *infreq*, *indir*, *outfreq=None*, *outdir=None*, *method='linear'*)

Interpolate onto new spectral basis.

Args:

inspec (2D ndarray): input spectrum E(infreq,indir) to be interpolated. infreq (1D ndarray): frequencies of input spectrum. indir (1D ndarray): directions of input spectrum. outfreq (1D ndarray): frequencies of output interpolated spectrum, same as

infreq by default.

outdir (1D ndarray): directions of output interpolated spectrum, same as
infreq by default.

method: {'linear', 'nearest', 'cubic'}, method of interpolation to use with
griddata.

Returns:

outspec (2D ndarray): interpolated ouput spectrum E(outfreq,outdir).

Note:

If either outfreq or outdir is None or False this coordinate is not interpolated Choose indir=None if spectrum is 1D.

TODO: Deprecate in favour of new regrid_spec function.

wavespectra.core.utils.flatten_list

wavespectra.core.utils.**flatten_list**(*l*, *a*)

Flatten list of lists

wavespectra.core.utils.regrid_spec

wavespectra.core.utils.**regrid_spec**(*dset*, *freq=None*, *dir=None*, *maintain_m0=True*)

Regrid spectra onto new spectral basis.

Args:

- dset (Dataset, DataArray): Spectra to interpolate.
- freq (DataArray, 1darray): Frequencies of interpolated spectra (Hz).
- dir (DataArray, 1darray): Directions of interpolated spectra (deg).
- maintain_m0 (bool): Ensure variance is conserved in interpolated spectra.

Returns:

- dsi (Dataset, DataArray): Regridded spectra.

Note:

- All freq below lowest freq are interpolated assuming $E_d(f = 0) = 0$.
- $Ed(f)$ is set to zero for new freq above the highest freq in dset.
- Only the 'linear' method is currently supported.
- Duplicate wrapped directions (e.g., 0 and 360) are removed when regridding directions because indices must be unique to intepolate.

swan module

core.swan.read_tab	Read swan table file.
core.swan.SwanSpecFile	Read spectra in SWAN ASCII format.
core.swan._dateparse	Date parsing to read SWAN tab files.

wavespectra.core.swan.read_tab

wavespectra.core.swan.**read_tab**(*filename*, *toff=0*)

Read swan table file.

Args:

filename (str): name of SWAN tab file to read toff (float): timezone offset in hours

Returns:

Pandas DataFrame object

wavespectra.core.swan.SwanSpecFile

class wavespectra.core.swan.**SwanSpecFile**(*filename*, *freqs=None*, *dirs=None*, *x=None*, *y=None*, *time=False*, *id='Swan Spectrum'*, *dirorder=False*, *append=False*, *tabfile=None*)

Read spectra in SWAN ASCII format.

__init__(*filename*, *freqs=None*, *dirs=None*, *x=None*, *y=None*, *time=False*, *id='Swan Spectrum'*, *dirorder=False*, *append=False*, *tabfile=None*)

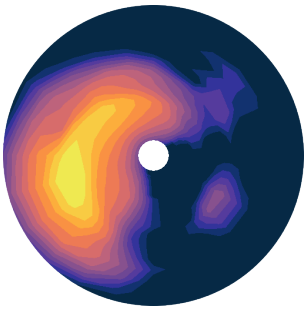
Methods

`__init__`(filename[, freqs, dirs, x, y, ...])	
`close`()	Close file handle.
`read`()	Read single timestep from current position in file.
`readall`()	Read the entire file.
`write_header`([time, str1, str2, timecode, ...])	Write header to file.
`write_spectra`(arr[, time])	Write spectra from single timestamp.

wavespectra.core.swan._dateparse

wavespectra.core.swan.**_dateparse**(*x*)

Date parsing to read SWAN tab files.



2.8 Credits

2.8.1 Development Lead

- Rafael Guedes <r.guedes@oceanum.science>

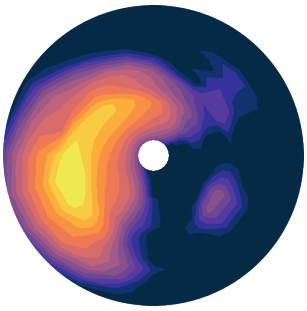
2.8.2 Developers

- Tom Durrant
- David Johnson

2.8.3 Contributors

- Carlos A. Michelén Ströfer
- Henrique Rapizo
- John Harrington
- Jorge Perez
- Paul Branson
- Ruben de Bruin

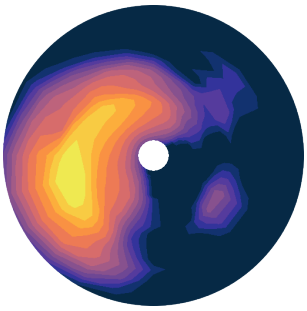
- Ryan Coe
- Spicer Bak



2.9 Support

Wavespectra is an open source project. The easiest way to get help with the project is to open an issue on Github.

Any contributions are welcome. Please feel free to fork this project and create pull requests, or contact Rafael Guedes with any specific queries (r.guedes@oceanum.science).



2.10 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

2.10.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/wavespectra/wavespectra/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

wavespectra could always use more documentation, whether as part of the official wavespectra docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/wavespectra/wavespectra/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

2.10.2 Get Started!

Ready to contribute? Here's how to set up *wavespectra* for local development.

1. Fork the *wavespectra* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/wavespectra.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv wavespectra
$ cd wavespectra/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 wavespectra tests
$ python setup.py test or py.test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

2.10.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.7, and for PyPy. Check https://travis-ci.org/wavespectra/wavespectra/pull_requests and make sure that the tests pass for all supported Python versions.

2.10.4 Tips

To run a subset of tests:

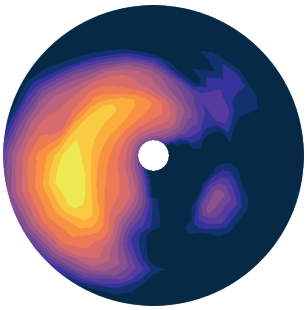
```
$ py.test tests.test_wavespectra
```

2.10.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.



2.11 History

Wavespectra is an open source project that was started at MetOcean Solutions and open sourced in April 2018. In July 2019 it was moved into the wavespectra github open source organisation and transitioned into a fully community developed project. This changelog covers the release history since v3.0 when wavespectra was open-sourced.

2.11.1 Releases

3.14.0 (2023-07-03)

Internal Changes

- Redefine packaging via pyproject.toml to conform to PEP517/518 (PR77).
- All packaging metadata removed from setup.py and moved to pyproject.toml. The setup.py file is now only used to build the Fortran module.
- Removed the MANIFEST.in file, package data now handled in pyproject.toml.
- Removed the requirements folder, requirements now handled in pyproject.toml.
- Removed some packaging attributes from wavespectra.__init__.py, now handled in pyproject.toml.
- New docs theme: sphinx_rtd_theme by pydata_sphinx_theme, fixes issue with rtd not working with sphinx>=7.0.
- Add readthedocs config.

3.13.0 (2023-01-09)

New Features

- Support for CSV Spotter files in read_spotter by by ryancoe (PR77).
- New reader *read_ndbc* for NDBC netcdf datasets (PR80).

Bug Fixes

- Fix bug in 2D spectra construction in read_ndbc_ascii due to wrong scaling (GH70).
- Ensure directions are continuous when reading from Funwave file with split directions.

Internal Changes

- New github action to test and publish package on new releases.

Deprecation

- Replace previous NDBC ASCII reader *read_ndbc* by *read_ndbc_ascii*.

3.12.1 (2022-10-27)

Internal Changes

- Fix numpy pre-install requirement by cmichelenstrofer (PR75).

3.12.0 (2022-08-19)

New Features

- Improve installation section in the docs to mention pre-install requirements of numpy and Fortran compiler.

Bug Fixes

- Fix bug caused by fixed numpy version (PR72).

Internal Changes

- Import Fortran partition code inside function so the library can still be imported if the module does not build.
- Remove Hypothesis from requirements.

3.11.0 (2022-05-04)

New Features

- New reader for Octopus file format by RubendeBruin (PR65).

Bug Fixes

- Fix bug in direction calculation caused by changes in xr ufuncs (PR59).
- Fix nrecs in test octopus file.
- Fix to zarr testing by RubendeBruin (PR55).

Internal Changes

- Only interpolate with inverse distance weighting if 2 or more neighbour sites are found within tolerance (PR62).
- Allow pathlib objects in read_swan (PR64).
- Increase float precision in Octopus writer.
- Make zarr, fsspec and gcsfs extra dependencies instead of default.
- Remove *get_mapper* call from zarr opener.

3.10.0 (2021-08-21)

New Features

- New option in *read_triaxys* to allow providing the magnitic declination to correct.
- New spectral regridding capability by RubendeBruin. The function is wrapped in *SpecArray.interp* and *SpecArray.interp_by* which mimic the behaviour in the respective counterparts from xarray.
- Replace plot api by a simple wrapper around xarray plotting capability. The new wrapper no longer duplicate internal functions from xarray and should better integrate any upstream changes. The new api also handles logarithmic axes and masking in a more natural way (PR48).
- New Orcaflex export function by RubendeBruin (PR37).
- New *wavespectra.core.utils.unique_indices* function (unique_times will be deprecated in future releases.

Bug Fixes

- Fix plot bug with the new plot api (GH44).
- Fix bug in *scale_by_hs* when run on dask datasets.

Internal Changes

- Fixed sphinx-gallery dependency by by RubendeBruin (PR41).
- Add new funwave functiont to docs.
- Update authors list.
- Allow pathlib objects in read_triaxys.

Deprecation

- Calling the plot kind as a method from *SpecArray.plot*, e.g. *SpecArray.plot.contourf* is deprecated with the new plotting api. Now *kind* needs to be provided as an argument.
- Arguments *show_radius_label* and *show_direction_label* are deprecated from *SpecArray.plot*. Labels are no longer drawn as they fall on top of ticks. In order to show it the axes properties now must be manually defined from the axis.
- Argument *as_log10* from the old plot api to plot the log10(efth) is deprecated in the new api. Similar result can be achieved in the new api by manually converting efth before plotting.

- Remove deprecated methods *_strictly_increasing* and *_collapse_array*.

3.9.0 (2021-05-29)

New Features

- Funwave spectra reader read_funwave (PR36).
- Funwave spectra writer to_funwave (PR36).

3.8.1 (2021-04-06)

Bug Fixes

- Add numba to setup.py, not installed properly from requirements/default.txt for some reason.

3.8.0 (2021-03-30)

New Features

- Watershed partitioning now supports dask (PR27).
- Spectral splitting now supports dask.
- The following spectral parameters now support dask (PR11):
  - tp
  - dp
  - dpm
  - dspr
- Wavespectra conda recipe by RubendeBruin.

Internal Changes

- Core watershed partitioning code organised into watershed module.
- *max_swells* replaced by *swells* in watershed partition to return fixed number of swells.
- Renamed module *wavespectra.core.misc* by *wavespectra.core.utils*.
- Removed deprecated method *_same_dims*, *_inflection* and *_product* from *SpecArray*.
- Get rid of simpy dependency.
- New daskable stats defined as ufuncs using numba.
- SpecArray attributes redefined as property methods.

Bug Fixes

deprecation

- Drop support for python < 3.7
- Dropped args *hs_min* and *nearest* in *SpecArray.partition*.

3.7.2 (2021-01-12)

New Features

- Handle ndbc spectra files with no minutes column (PR25).
- Writers to_swan and to_octopus now deal with extra non-supported dimensions.

Internal Changes

- Stop fixing pandas and xarray versions.
- Remove attrdict dependency.
- Define *_FillValue* in to_netcdf.

Bug Fixes

- Fix bug in sel with *"nearest"* option.
- Ensure last time chunk is written in to_swan when the dataset time size is not divisible by ntime (GH20).

3.7.1 (2020-08-26)

Internal Changes

- Optimise to_swan (over 100x improvements when writing very large spectra).
- Optimise to_octopus (over 10x improvements when writing very large spectra).
- Allow loading time chunks when writing swan and octopus files.

3.7.0 (2020-07-16)

New Features

- New json reader and writer (PR21).

Internal Changes

- Raise exception when trying to compute directional methods on 1d, frequency spectra.

3.6.5 (2020-07-10)

Bug Fixes

- Fix bug in sel methods.

3.6.4 (2020-06-29)

Bug Fixes

- Ensure yml config is shipped with distribution.

3.6.3 (2020-06-28)

Internal Changes

- Increase time resolution in netcdf outptu from to_netcdf.

3.6.2 (2020-06-28)

Internal Changes

- Make netcdf packing work for datasets in zarr format.

3.6.1 (2020-06-28)

Internal Changes

- Packing output netcdf files as int32 dtype by default.

3.6.0 (2020-06-27)

New Features

- New method to construct spectra from NDBC buoy data (PR17).
- New method to output spectra in native WW3 format.

Bug Fixes

- Fix bug with selecting circular longitudes in different conventions (GH20).
- Ensure directions in coming-from convention in read_era5 (PR18).
- Fix radian convertions in read_era5 (PR19).
- Fix coordinate values assignment errors with xarray>=0.15.1 (GH16).
- Ensure coordinates attributes are kept with certain readers.

deprecation

- Deprecated legacy *read_ww3_msl* reader.
- Deprecated *read_dictionary* in favour of using xarray's to_dict and from_dict methods.

Internal Changes

- Remove curly brackets from units.
- Remove original variable attributes from files hidden with underscores (*_units* and *_variable_name*).
- Remove xarray version limitation to <0.15.0.

3.5.3 (2020-04-14)

Fix xarray version until breaking changes with 0.15.1 are taken care of.

Bug Fixes

- Avoid index duplication when merging datasets in to_octopus function.

Internal Changes

- Fix xarray at 0.15.0 for now as 0.15.1 introduces many breaking changes.

3.5.2 (2020-03-09)

New Features

- New method read_era5 to read spectra in ERA5 format by John Harrington.
- New method read_wavespectra to read files already in wavespectra convention.

3.5.1 (2019-12-12)

Bug Fixes

- Import accessors within try block in __init__.py so install won't break.

Internal Changes

- Implemented coveralls.
- Added some more tests.

3.5.0 (2019-12-09)

The first PyPI release from new wavespectra **github organisation.**

Breaking Changes

- Drop support for Python 2.
- Drop support for Python < 3.6.

New Features

- Add method in SpecDataset accessor to plot polar wave spectra, api borrowed from xarray.
- New *sel* method in SpecDataset accessor to select sites using different methods.
- Support for zarr wave spectra datasets from either local or remote sources.
- New *read_spotter* function to read spectra from Spotter file format, currently only reading as 1D.
- Add *read_dataset* function to convert existing dataset from unknown file into SpecDataset.
- Python Notebooks split into a new notebooks repository within the wavespectra organisation.
- New branch pure-python with fortran watershed algorithm replaced by python. This code is ~3x slower than the fortran one but it is easier to install particularly if the system does not have fortran compiler. We will make an effort to keep this branch in sync with Master.
- Redefined autodocs.

Bug Fixes

- Consolidate history to link to github commits from all contributors.
- Fix error in *partition* with dask array not supportting item assignment.
- Fix docs building, currently working from *pure-python* branch due to gfortran dependency.

Internal Changes

- Decouple file reading from accessor definition in input functions so existing datasets can be converted.
- Compute method *_twod* lazily.
- Replace drop calls to fix deprecation warnings.
- Consolidate changelog in history file.
- Building with travis and tox.
- Adopt black code formatting.
- Set up flake8.

3.4.0 (2019-03-28)

The last PyPI release from old metocean github organisation.

New Features

- Add support to Python 3.

3.3.1 (2019-03-19)

New Features

- Support SWAN Cartesian locations.
- Support energy unit in SWAN ASCII spectra.

3.3.0 (2019-02-21)

New Features

- Add *dircap_270* option in *read_swan*.

Bug Fixes

- Ensure lazy computations in *swe* method.

Internal Changes

- Remove *inplace* calls that will deprecate in xarray.

3.2.5 (2019-01-25)

Bug Fixes

- Ensure datasets are loaded lazily in *read_swan* and *read_wwm*.

3.2.4 (2019-01-23)

Bug Fixes

- Fix tp-smooth bug caused by float32 dtype.

3.2.3 (2019-01-08)

New Features

- Function *read_triaxys* to read spectra from TRIAXYS file format.

Bug Fixes

- Fix bug with frequency and energy units in *read_wwm*.

3.2.2 (2018-12-04)

Bug Fixes

- Ensure dataset from swan netcdf has site coordinate.

3.2.1 (2018-11-14)

New Features

- Function *read_wwm* to read spectra from WWM model format.

Bug Fixes

- Convert direction to degree in *read_ncswan*.

3.2.0 (2018-11-04)

New Features

- Function *read_ncswan* to read spectra from SWAN netcdf model format.

Bug Fixes

- Ensure lazy computation in *uv_to_spddir*.

Internal changes

- Unify library PyPI release versions.

3.1.4 (2018-08-29)

Bug Fixes

- Fix bug in *read_swans* when handling swan bnd files with *ntimes* argument.

3.1.3 (2018-07-27)

Changes

- Use 10m convention in default wind standard names.

3.1.2 (2018-07-05)

Changes

- Adjust default standard name for *dm*.

Bug Fixes

- Fix renaming option in *stats* method.

3.1.1 (2018-05-17)

Bug Fixes

New Features

- Allow choosing maximum number of partitions in *partition* method.

3.1.0 (2018-05-09)

New Features

- Function to read spectra in cf-json formatting.

Bug Fixes

- Fix but in *read_swan* when files have no timestamp.

3.0.2 (2018-05-03)

Bug Fixes

- Ensure data is not loaded into memory in *read_ww3*.

3.0.1 (2018-04-28)

New Features

- Sphinx autodoc.
- Method *read_dictionary* to define SpecDataset from python dictionary.
- Set pytest as the testing framework and add several new testings.
- Add notebooks.

Bug Fixes

- Get rid of left over *freq* coordinate in *hs* method.
- Fix calculation in *_peak* method.
- Stop misleading warning in *tp* method.
- Fix to *hs* method.

Internal Changes

- Replace obsolete sort method by xarray's sortby.
- Falster calculation in *tp*.
- Improvements to SpecDataset wrapper.

3.0 (2018-03-05)

This major release marks the migration from the predecessor *pyspectra* library, as well as the open-sourcing of wavespectra and first PyPI release.

New Features

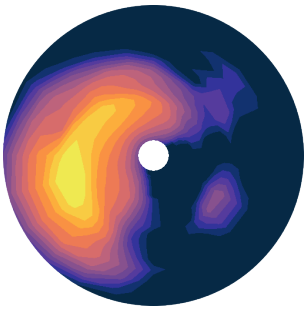
- Library restructured with plugins input / output modules .
- New *_peak* method to return the true peak instead of the maxima.
- Making reading functions available at module level.

Bug Fixes

- Ensure slicing won't break due to precision (xarray bug).

Internal Changes

- Rename package.



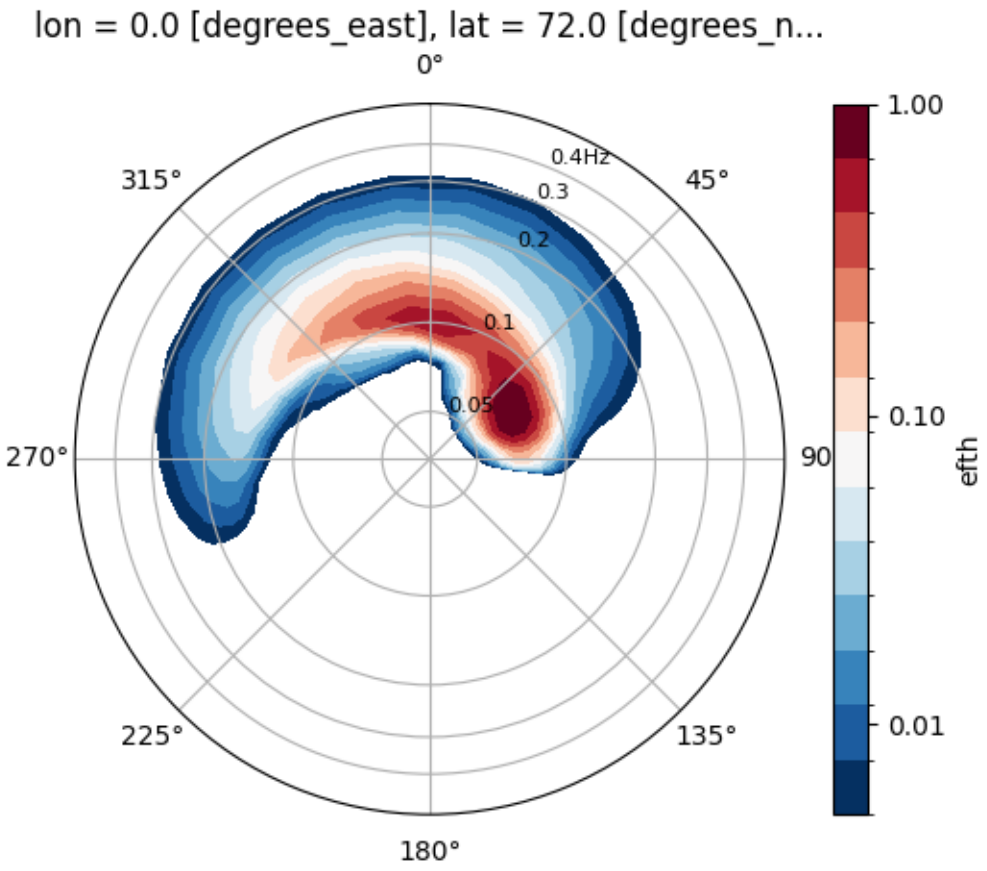
2.12 Gallery

orphan

2.12.1 Gallery

Default arguments

Plot of wave spectrum with default options



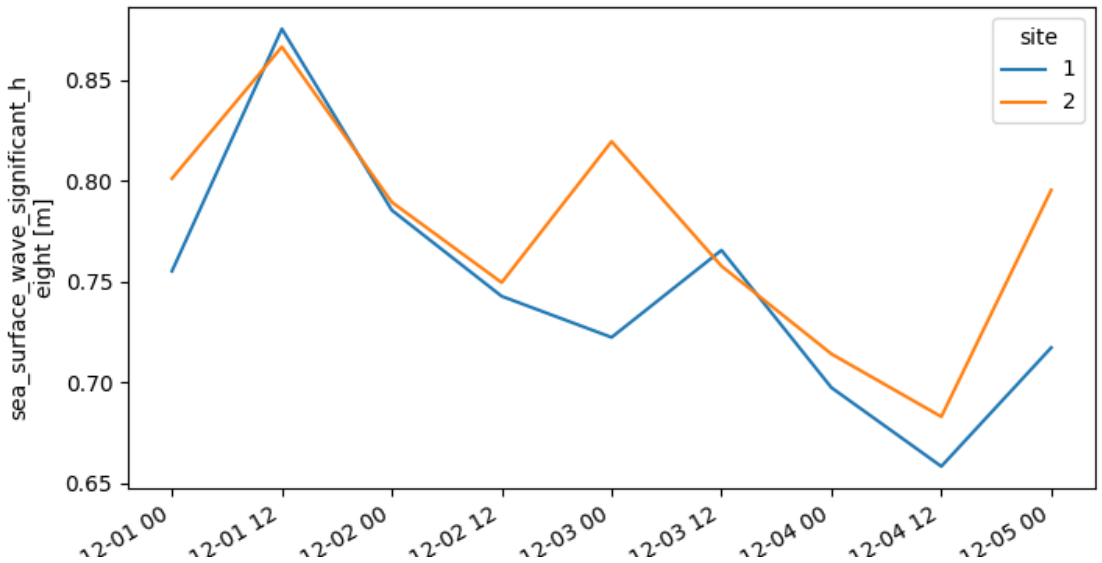

```
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/wavespectra/input/netcdf.py:50: UserWarning: rename 'time' to 'time'␣
↪does not create an index anymore. Try using swap_dims instead or use set_index after␣
↪rename to create an indexed coordinate.
  dset = dset.rename({k: v for k, v in coord_map.items() if k in dset})
```

```
import matplotlib.pyplot as plt
from wavespectra import read_era5


dset = read_era5("../_static/era5file.nc")
ds = dset.isel(lat=0, lon=0, time=0)
p = ds.spec.plot()
```

Total running time of the script: (0 minutes 0.837 seconds)

Calculate and plot Hs

Plots Hs calculated from spectra dataset



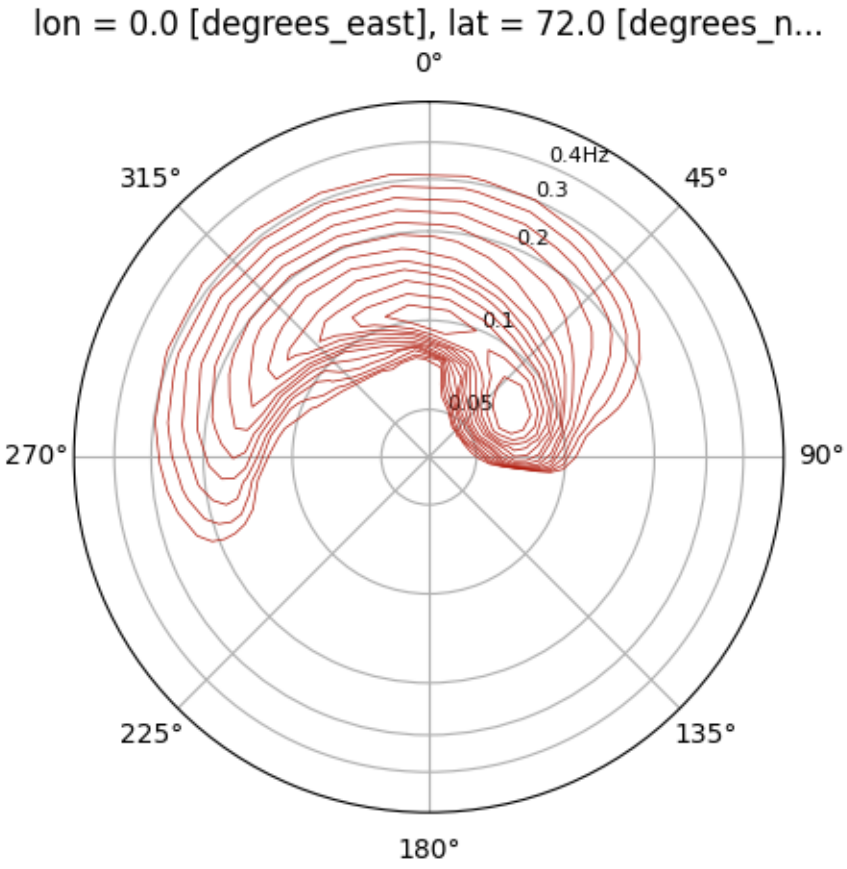

```
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/wavespectra/input/ww3.py:63: UserWarning: rename 'time' to 'time' does␣
↪not create an index anymore. Try using swap_dims instead or use set_index after rename␣
↪to create an indexed coordinate.
  dset = dset.rename(MAPPING)
```

```
import matplotlib.pyplot as plt
from wavespectra import read_ww3


dset = read_ww3("../_static/ww3file.nc")

fig = plt.figure(figsize=(8, 4))
hs = dset.spec.hs()
p = hs.plot.line(x="time")
```

Total running time of the script: (0 minutes 0.484 seconds)

Spectrum as contour

Contour type plot of wave spectrum



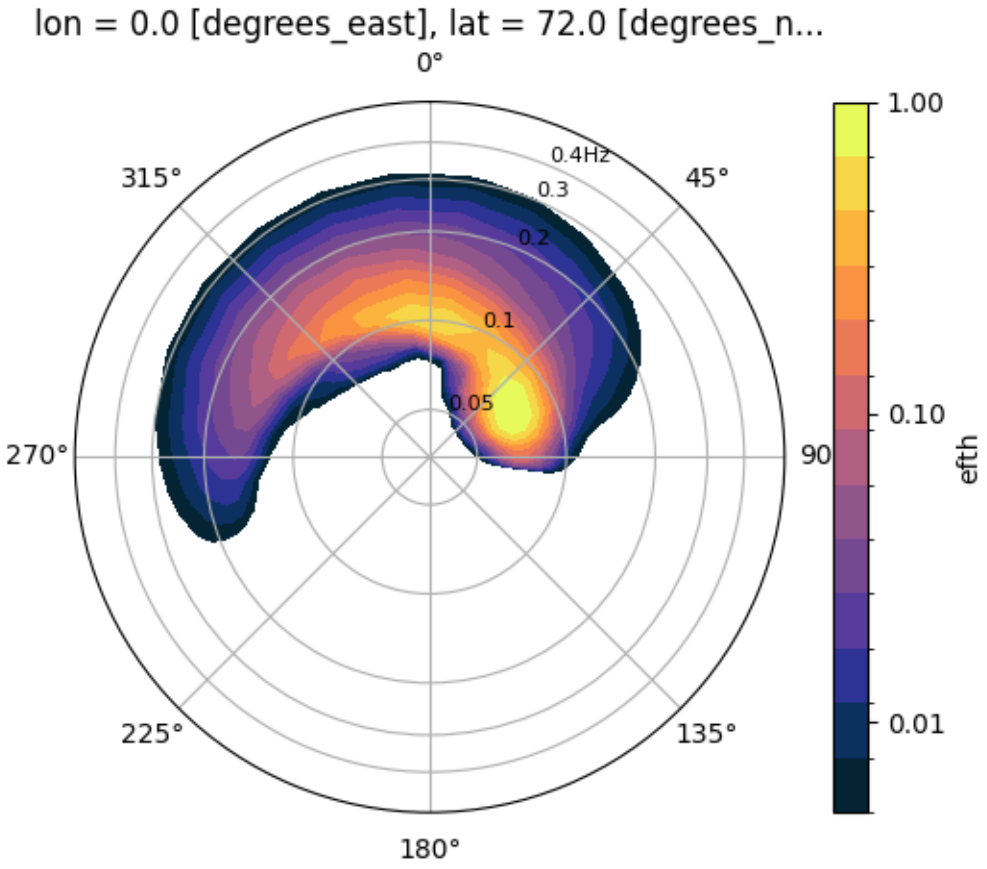

```
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/wavespectra/input/netcdf.py:50: UserWarning: rename 'time' to 'time'␣
↪does not create an index anymore. Try using swap_dims instead or use set_index after␣
↪rename to create an indexed coordinate.
  dset = dset.rename({k: v for k, v in coord_map.items() if k in dset})
```

```
import matplotlib.pyplot as plt
from wavespectra import read_era5


dset = read_era5("../_static/era5file.nc")
ds = dset.isel(lat=0, lon=0, time=0)
p = ds.spec.plot(kind="contour", colors="#af1607", linewidths=0.5)
```

Total running time of the script: (0 minutes 0.585 seconds)

Spectrum as contourf

Contourf type plot of wave spectrum



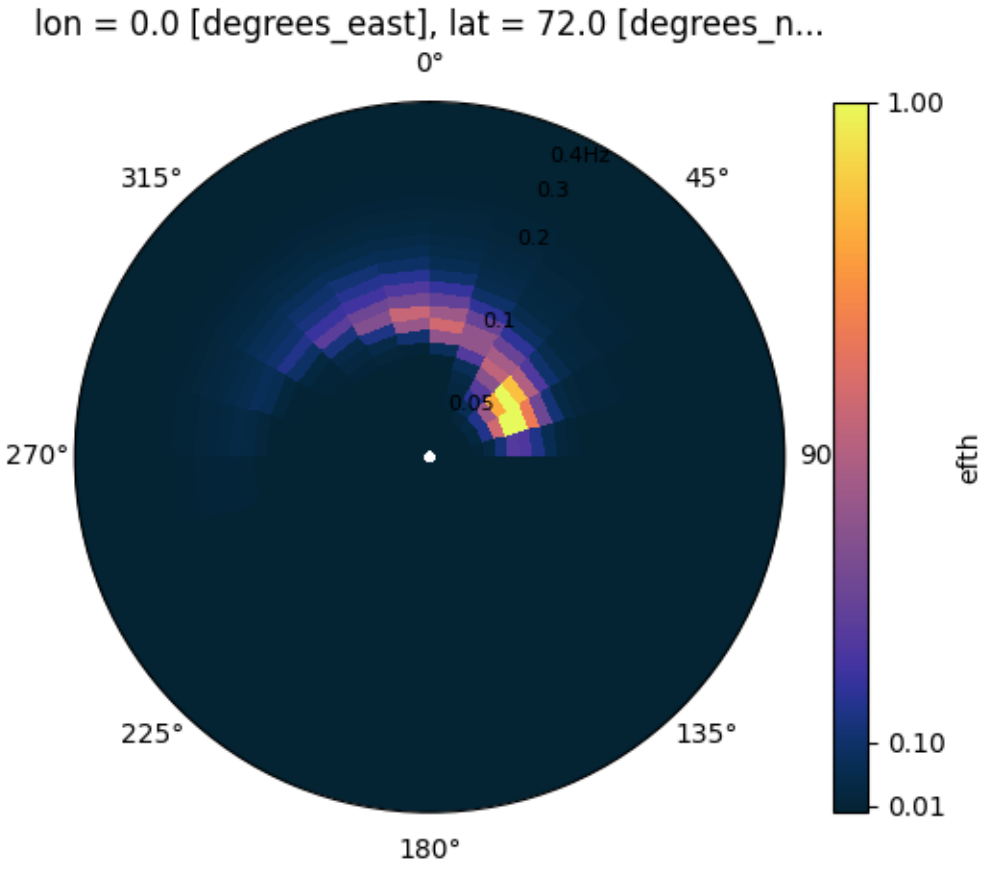

```
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/wavespectra/input/netcdf.py:50: UserWarning: rename 'time' to 'time'␣
↪does not create an index anymore. Try using swap_dims instead or use set_index after␣
↪rename to create an indexed coordinate.
  dset = dset.rename({k: v for k, v in coord_map.items() if k in dset})
```

```
import matplotlib.pyplot as plt
import cmocean
from wavespectra import read_era5


dset = read_era5("../_static/era5file.nc")
ds = dset.isel(lat=0, lon=0, time=0)
p = ds.spec.plot(kind="contourf", cmap=cmocean.cm.thermal)
```

Total running time of the script: (0 minutes 0.665 seconds)

Spectrum as pcolormesh

Pcolor type plot of wave spectrum



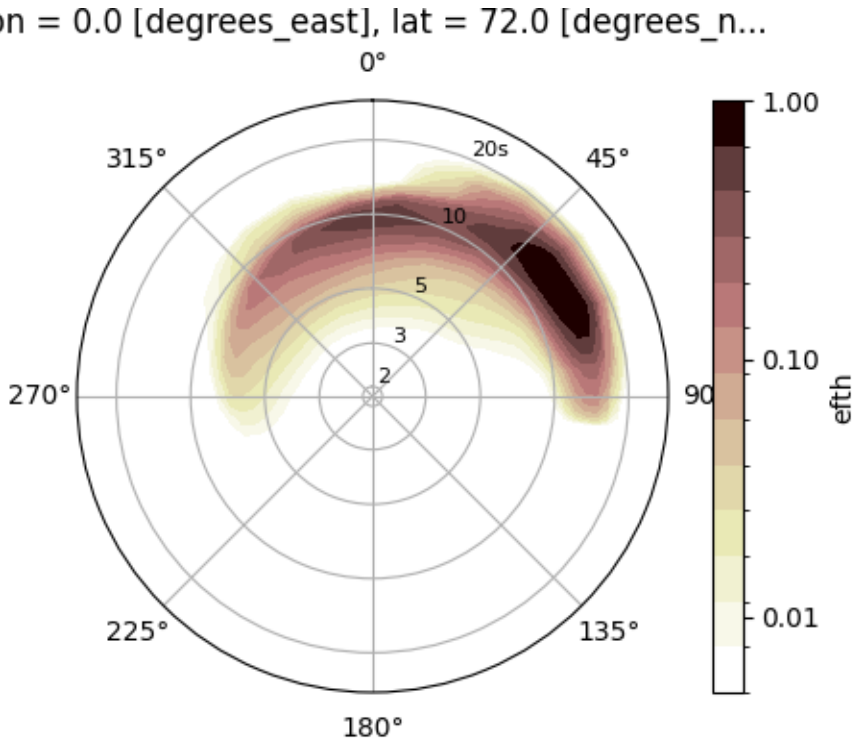

```
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/wavespectra/input/netcdf.py:50: UserWarning: rename 'time' to 'time'␣
↪does not create an index anymore. Try using swap_dims instead or use set_index after␣
↪rename to create an indexed coordinate.
  dset = dset.rename({k: v for k, v in coord_map.items() if k in dset})
```

```
import matplotlib.pyplot as plt
import cmocean
from wavespectra import read_era5


dset = read_era5("../_static/era5file.nc")
ds = dset.isel(lat=0, lon=0, time=0)
p = ds.spec.plot(kind="pcolormesh", cmap=cmocean.cm.thermal)
```

Total running time of the script: (0 minutes 0.495 seconds)

Wave period spectrum

Plot of a wave spectrum in period space



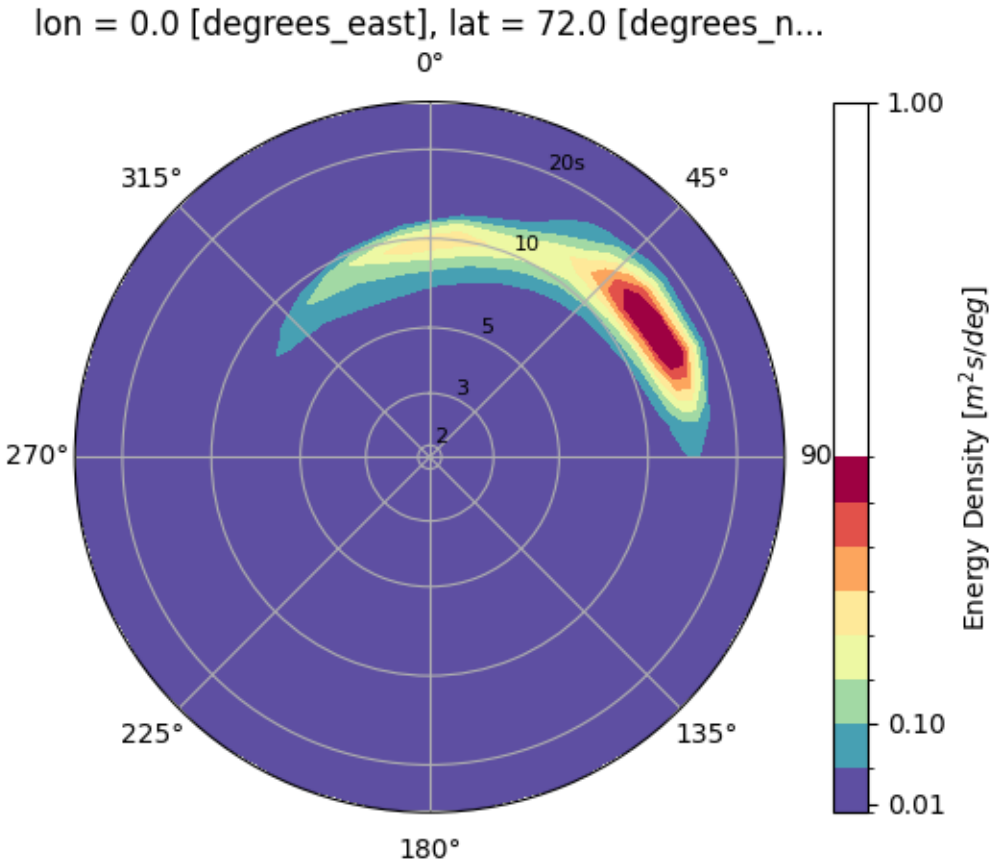

```
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/wavespectra/input/netcdf.py:50: UserWarning: rename 'time' to 'time'␣
↪does not create an index anymore. Try using swap_dims instead or use set_index after␣
↪rename to create an indexed coordinate.
  dset = dset.rename({k: v for k, v in coord_map.items() if k in dset})
```

```
import matplotlib.pyplot as plt
import cmocean
from wavespectra import read_era5


dset = read_era5("../_static/era5file.nc")
ds = dset.isel(lat=0, lon=0, time=0)
fig = plt.figure(figsize=(6, 4))
p = ds.spec.plot(as_period=True, cmap="pink_r")
```

Total running time of the script: (0 minutes 0.642 seconds)

Energy density values

Show actual energy density rather than normalised values




```
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/wavespectra/input/netcdf.py:50: UserWarning: rename 'time' to 'time'␣
↪does not create an index anymore. Try using swap_dims instead or use set_index after␣
↪rename to create an indexed coordinate.
  dset = dset.rename({k: v for k, v in coord_map.items() if k in dset})
```

```
import matplotlib.pyplot as plt
from wavespectra import read_era5


dset = read_era5("../_static/era5file.nc")
ds = dset.isel(lat=0, lon=0, time=0)
p = ds.spec.plot(
    normalised=False,
    as_period=True,
```

(continues on next page)

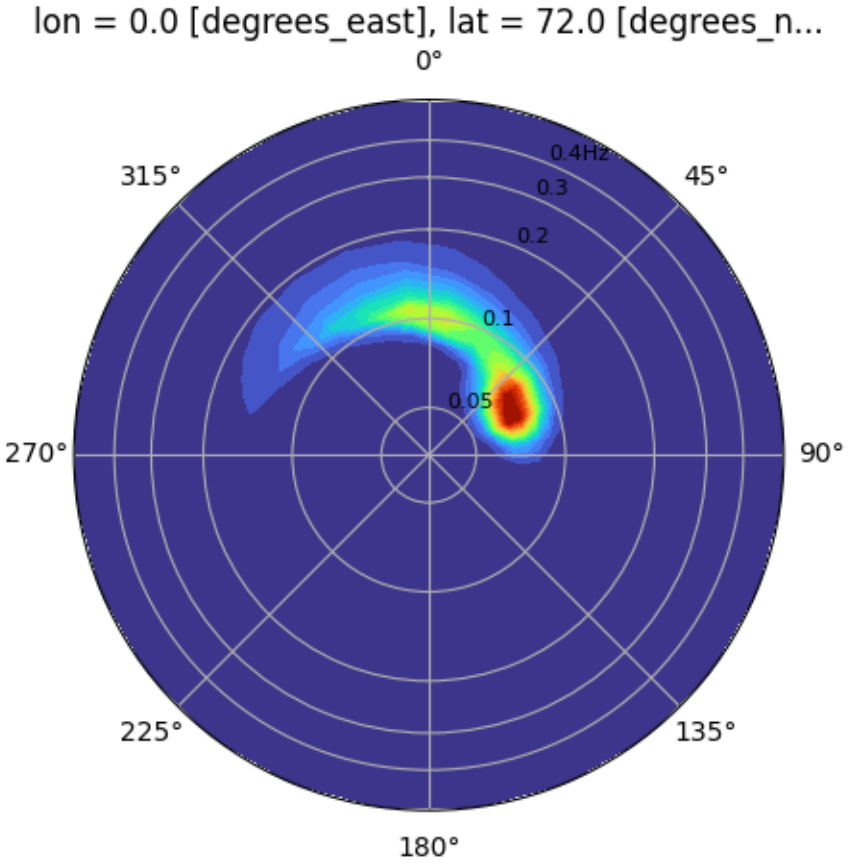
(continued from previous page)

```
    cmap="Spectral_r",
    levels=10,
)
```

Total running time of the script: (0 minutes 0.718 seconds)

Plotting parameters from xarray

Wavespectra allows passing some parameters from the functions wrapped from xarray such as contourf (excluding some that are manipulated in wavespectra such as *ax*, *x* and others):



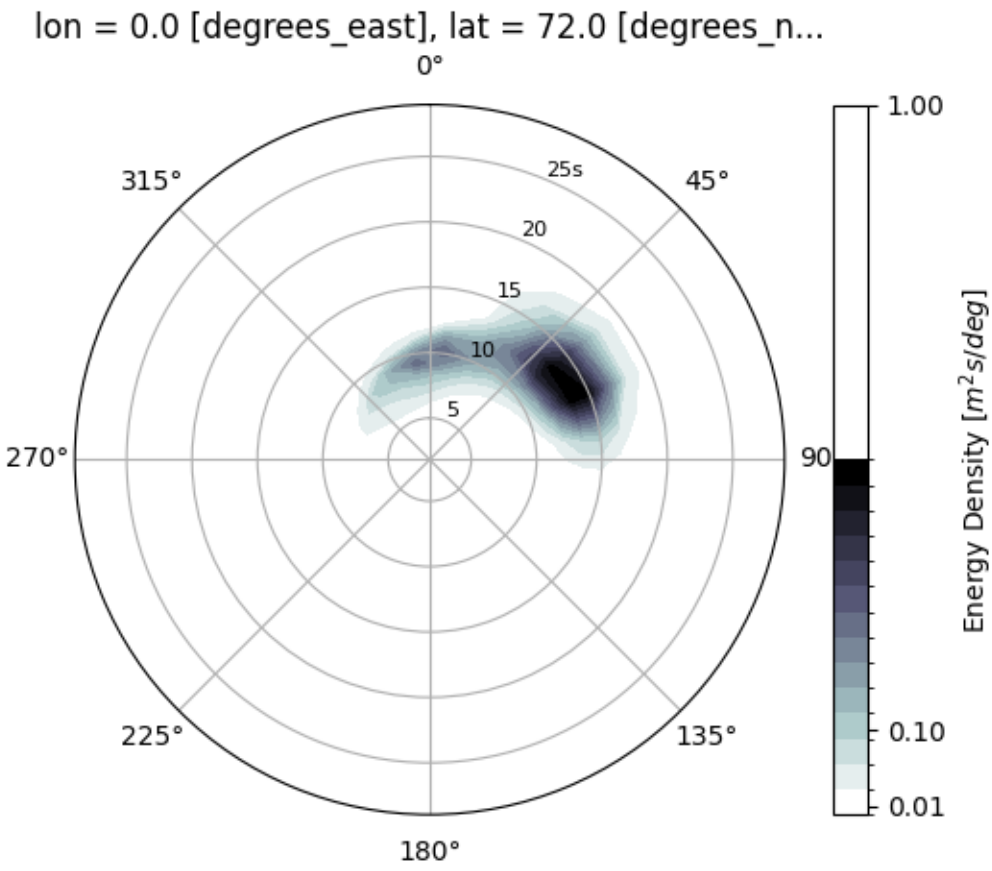

```
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/wavespectra/input/netcdf.py:50: UserWarning: rename 'time' to 'time'␣
↪does not create an index anymore. Try using swap_dims instead or use set_index after␣
↪rename to create an indexed coordinate.
  dset = dset.rename({k: v for k, v in coord_map.items() if k in dset})
```

```
import matplotlib.pyplot as plt
from wavespectra import read_era5


dset = read_era5("../_static/era5file.nc")
ds = dset.isel(lat=0, lon=0, time=0)
p = ds.spec.plot(
    kind="contourf",
    cmap="turbo",
    add_colorbar=False,
    extend="both",
    levels=25
    )
```

Total running time of the script: (0 minutes 0.578 seconds)

Linear radii

Radii are shown on a logarithmic scale by default but can also be shown on a linear scale




```
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/wavespectra/input/netcdf.py:50: UserWarning: rename 'time' to 'time'␣
```

(continues on next page)

(continued from previous page)

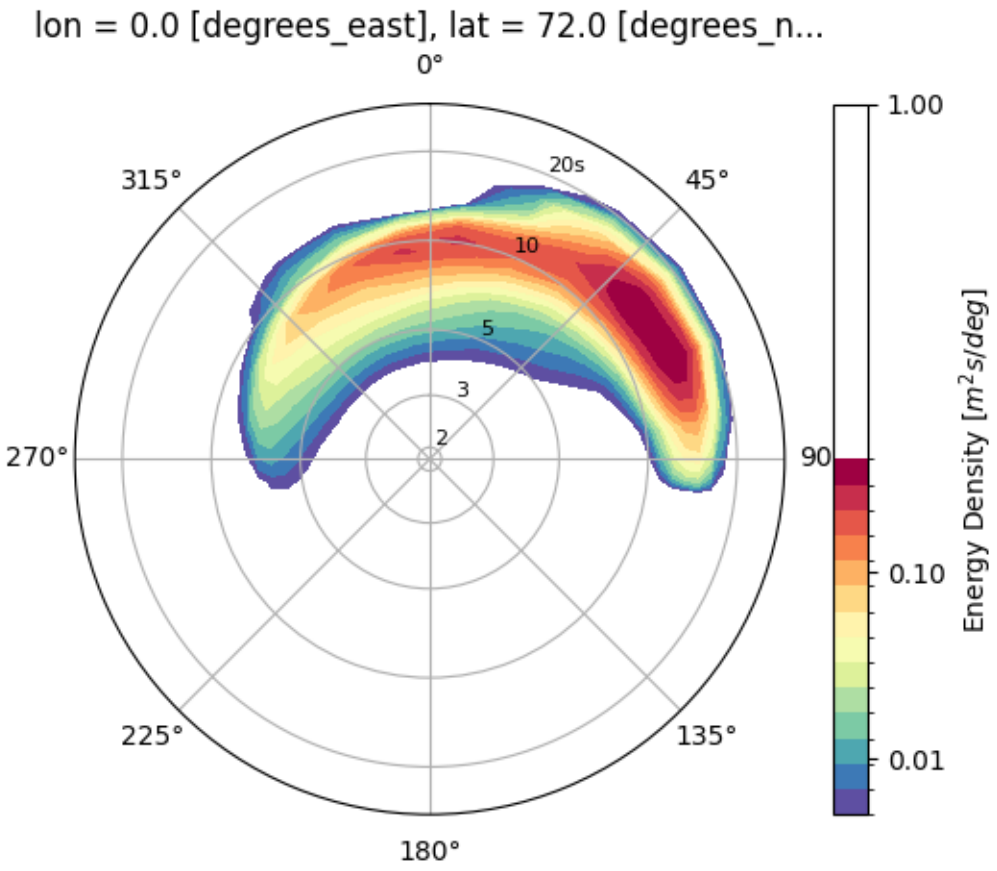
```
↪does not create an index anymore. Try using swap_dims instead or use set_index after␣
↪rename to create an indexed coordinate.
  dset = dset.rename({k: v for k, v in coord_map.items() if k in dset})
```

```
import matplotlib.pyplot as plt
from wavespectra import read_era5


dset = read_era5("../_static/era5file.nc")
ds = dset.isel(lat=0, lon=0, time=0)
p = ds.spec.plot(
        as_period=True,
        normalised=False,
        levels=15,
        cmap="bone_r",
        logradius=False,
        radii_ticks=[5, 10, 15, 20, 25],
)
```

Total running time of the script: (0 minutes 0.491 seconds)

Logarithmic contours

Logarithmic contour levels are only default for normalised spectra but they can be still manually specified




```
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/wavespectra/input/netcdf.py:50: UserWarning: rename 'time' to 'time'␣
↪does not create an index anymore. Try using swap_dims instead or use set_index after␣
↪rename to create an indexed coordinate.
  dset = dset.rename({k: v for k, v in coord_map.items() if k in dset})
```

```
import numpy as np
import matplotlib.pyplot as plt
from wavespectra import read_era5


dset = read_era5("../_static/era5file.nc")
ds = dset.isel(lat=0, lon=0, time=0)
p = ds.spec.plot(
    normalised=False,
    as_period=True,
    cmap="Spectral_r",
    levels=np.logspace(np.log10(0.005), np.log10(0.4), 15),
    cbar_ticks=[0.01, 0.1, 1],
```

(continues on next page)

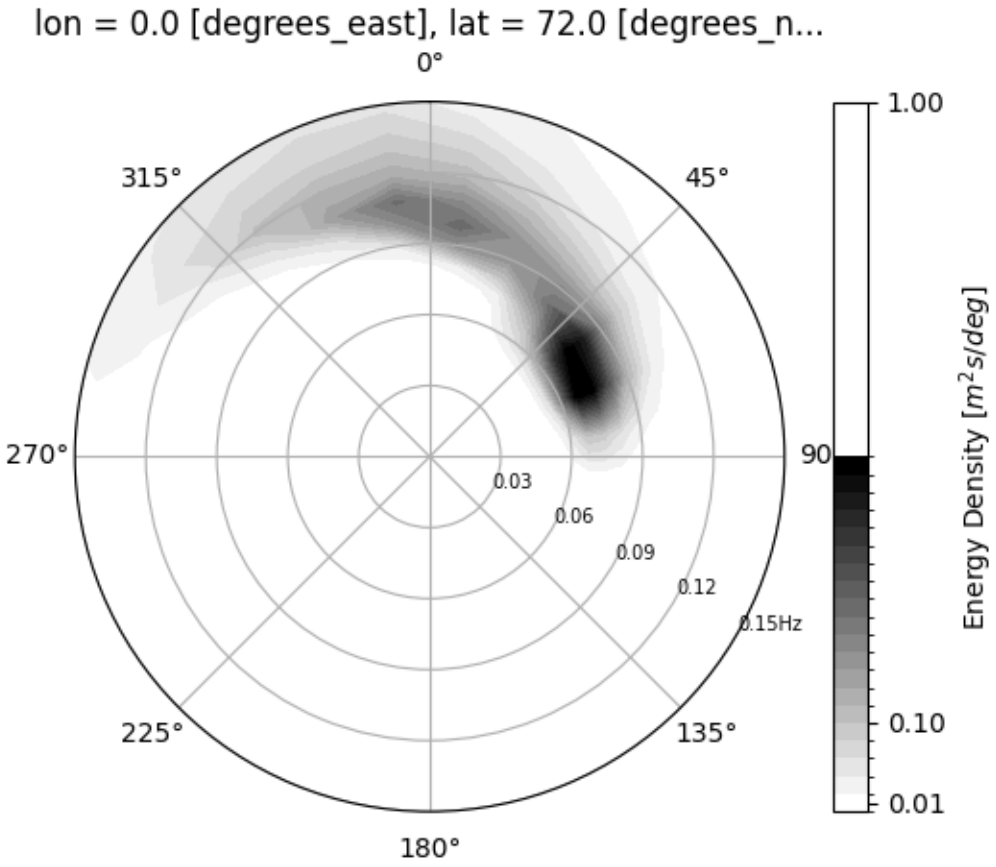
(continued from previous page)

```
)
```

Total running time of the script: (0 minutes 0.526 seconds)

Control radius extent

Radius extent can be defined from *rmin* and *rmax* parameters



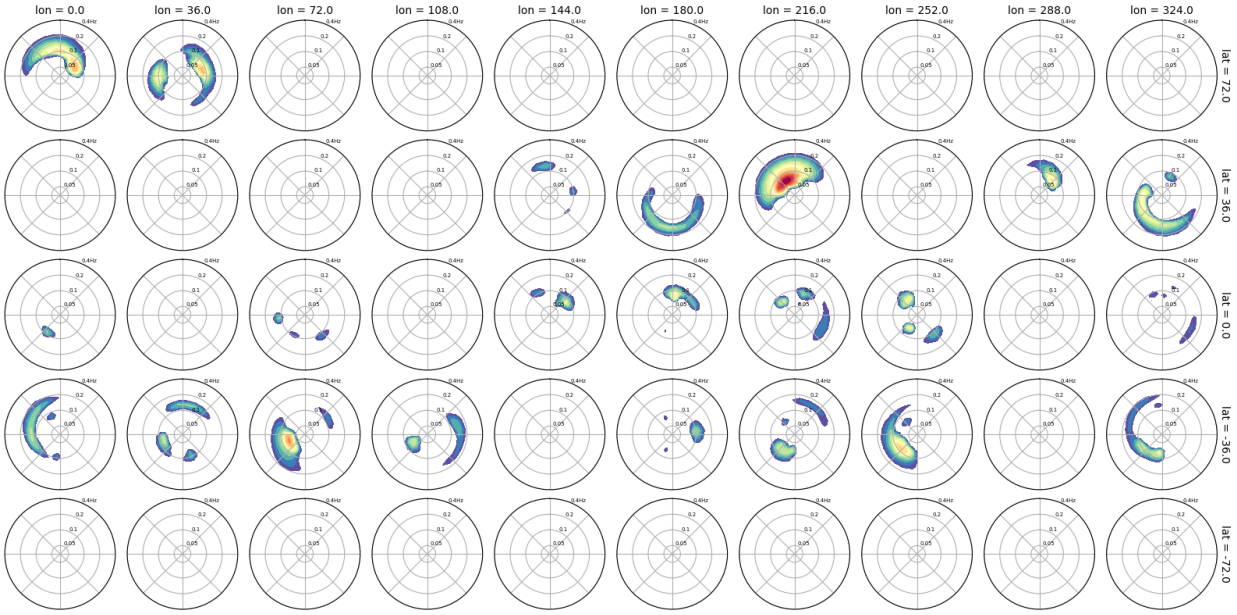

```
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/wavespectra/input/netcdf.py:50: UserWarning: rename 'time' to 'time'␣
↪does not create an index anymore. Try using swap_dims instead or use set_index after␣
↪rename to create an indexed coordinate.
  dset = dset.rename({k: v for k, v in coord_map.items() if k in dset})
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/xarray/plot/dataarray_plot.py:2141: UserWarning: The following kwargs␣
↪were not used by contour: 'radii_labels'
  primitive = ax.contourf(x, y, z, **kwargs)
```

```
import matplotlib.pyplot as plt
from wavespectra import read_era5


dset = read_era5("../_static/era5file.nc")
ds = dset.isel(lat=0, lon=0, time=0)
p = ds.spec.plot(
    rmin=0,
    rmax=0.15,
    logradius=False,
    normalised=False,
    levels=25,
    cmap="gray_r",
    radii_ticks=[0.03, 0.06, 0.09, 0.12, 0.15],
    radii_labels=["0.05", "0.1", "0.15Hz"],
    radii_labels_angle=120,
    radii_labels_size=7
)
```

Total running time of the script: (0 minutes 0.350 seconds)

Faceting capability

Faceting capability from xarray is supported




```
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/wavespectra/input/netcdf.py:50: UserWarning: rename 'time' to 'time'␣
↪does not create an index anymore. Try using swap_dims instead or use set_index after␣
↪rename to create an indexed coordinate.
  dset = dset.rename({k: v for k, v in coord_map.items() if k in dset})
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/checkouts/v3.14.0/docs/
```

(continues on next page)

(continued from previous page)

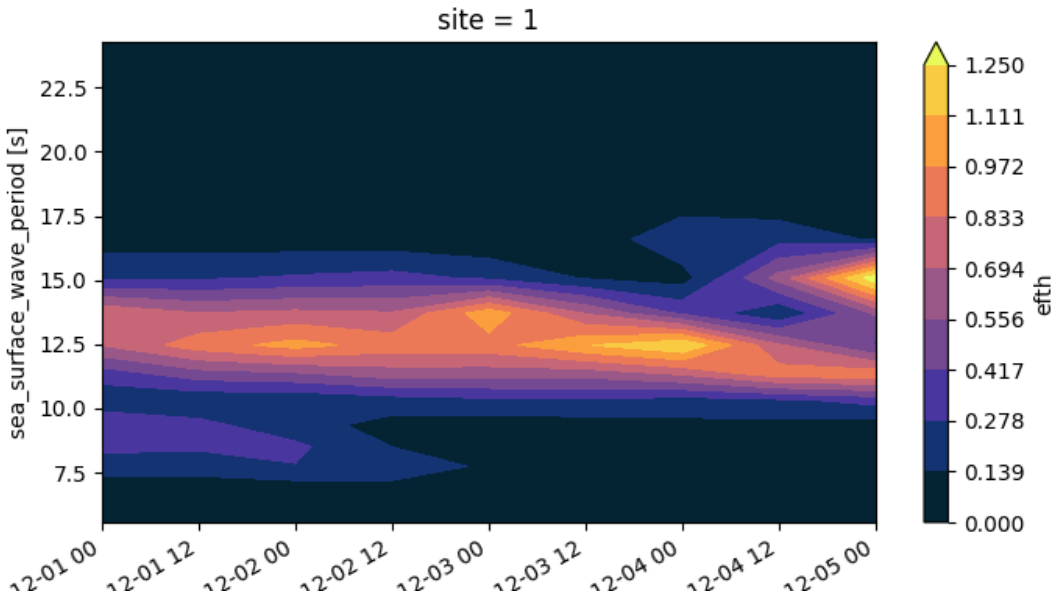
```
↪gallery/plot_faceting.py:26: UserWarning: The figure layout has changed to tight
  plt.tight_layout()
```

```
import numpy as np
import matplotlib.pyplot as plt
from wavespectra import read_era5


dset = read_era5("../_static/era5file.nc").isel(time=0)
p = dset.spec.plot(
        col="lon",
        row="lat",
        figsize=(16,8),
        add_colorbar=False,
        show_theta_labels=False,
        show_radii_labels=True,
        radii_ticks=[0.05, 0.1, 0.2, 0.4],
        rmax=0.4,
        radii_labels_size=5,
        cmap="Spectral_r",
)
plt.tight_layout()
```

Total running time of the script: (0 minutes 24.581 seconds)

Hovmoller diagram of spectra timeseries

Integrate spectra and plot as Hovmoller diagram



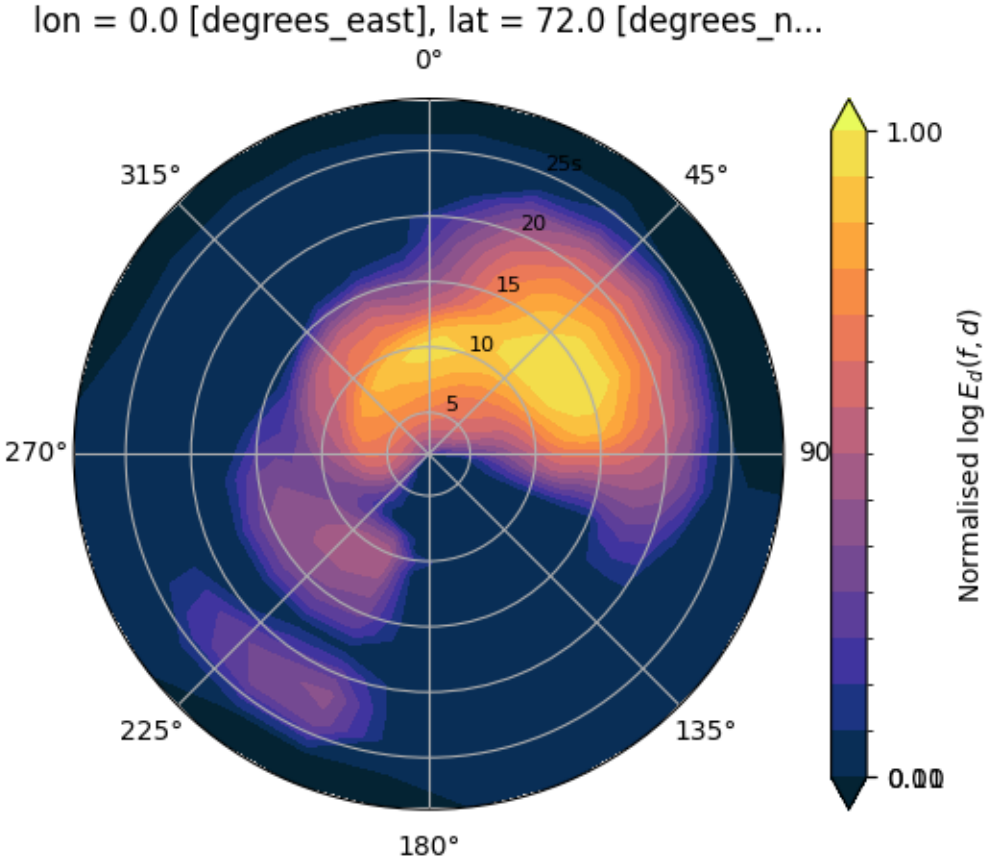

```
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/wavespectra/input/ww3.py:63: UserWarning: rename 'time' to 'time' does␣
↪not create an index anymore. Try using swap_dims instead or use set_index after rename␣
↪to create an indexed coordinate.
  dset = dset.rename(MAPPING)
```

```
import matplotlib.pyplot as plt
import cmocean
from wavespectra import read_ww3


fig = plt.figure(figsize=(8, 4))

dset = read_ww3("../_static/ww3file.nc")
ds = dset.isel(site=0).spec.split(fmax=0.18)
ds = ds.spec.oned().rename({"freq": "period"})
ds = ds.assign_coords({"period": 1 / ds.period})
ds.period.attrs.update({"standard_name": "sea_surface_wave_period", "units": "s"})
p = ds.plot.contourf(
    x="time", y="period", vmax=1.25, cmap=cmocean.cm.thermal, levels=10
)
```

Total running time of the script: (0 minutes 0.334 seconds)

Logarithmic energy density

The *as_log10* option to plot the $\log E_d(f,d)$ has been deprecated but similar result can be achieved by calculating the $\log E_d(f,d)$ beforehand:




```
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/wavespectra/input/netcdf.py:50: UserWarning: rename 'time' to 'time'␣
↪does not create an index anymore. Try using swap_dims instead or use set_index after␣
↪rename to create an indexed coordinate.
  dset = dset.rename({k: v for k, v in coord_map.items() if k in dset})
```

```
import numpy as np
import matplotlib.pyplot as plt
import cmocean
from wavespectra import read_era5


dset = read_era5("../_static/era5file.nc")
ds = dset.isel(lat=0, lon=0, time=0)
```

(continues on next page)

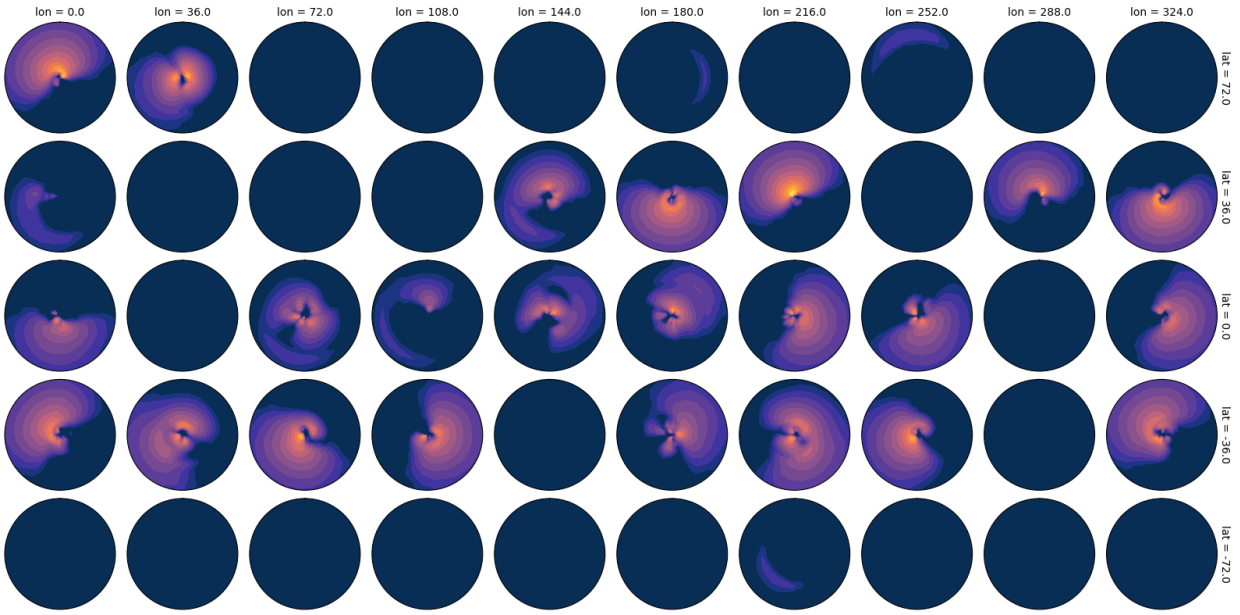
(continued from previous page)

```
ds = ds.where(ds>0, 1e-5) # Avoid infinity values
ds = np.log10(ds)
p = ds.spec.plot(
        as_period=True,
        logradius=False,
        cbar_kwargs={"label": "Normalised $\log{E_{d}(f,d)}$"},
        vmin=0.39,
        levels=15,
        extend="both",
        cmap=cmocean.cm.thermal,
)
```

Total running time of the script: (0 minutes 0.680 seconds)

Faceting with clean axes

Removing axes could help visualising patterns in multiple plots



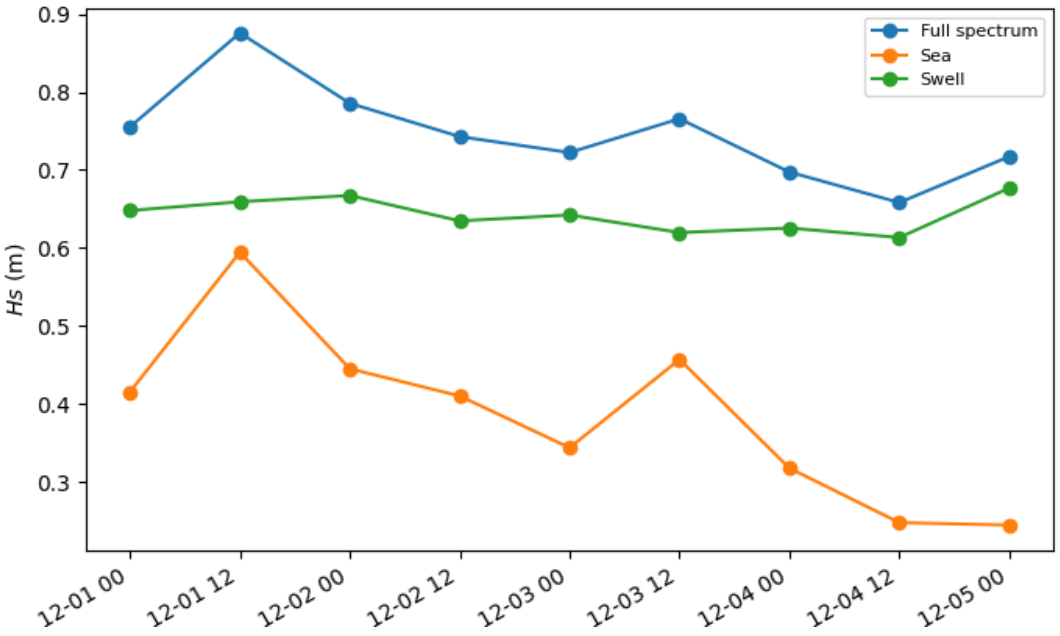

```
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/wavespectra/input/netcdf.py:50: UserWarning: rename 'time' to 'time'␣
↪does not create an index anymore. Try using swap_dims instead or use set_index after␣
↪rename to create an indexed coordinate.
  dset = dset.rename({k: v for k, v in coord_map.items() if k in dset})
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/checkouts/v3.14.0/docs/
↪gallery/plot_clean_axis.py:29: UserWarning: The figure layout has changed to tight
  plt.tight_layout()
```

```
import numpy as np
import matplotlib.pyplot as plt
import cmocean
from wavespectra import read_era5


dset = read_era5("../_static/era5file.nc").isel(time=0)
dset1 = dset.where(dset>0, 1e-5)
dset1 = np.log10(dset1)
p = dset1.spec.plot(
        clean_axis=True,
        col="lon",
        row="lat",
        figsize=(16,8),
        logradius=False,
        vmin=0.39,
        levels=15,
        extend="both",
        cmap=cmocean.cm.thermal,
        add_colorbar=False,
)
plt.tight_layout()
```

Total running time of the script: (0 minutes 27.452 seconds)

Frequency-split parameters

Split spectra and plot parameters



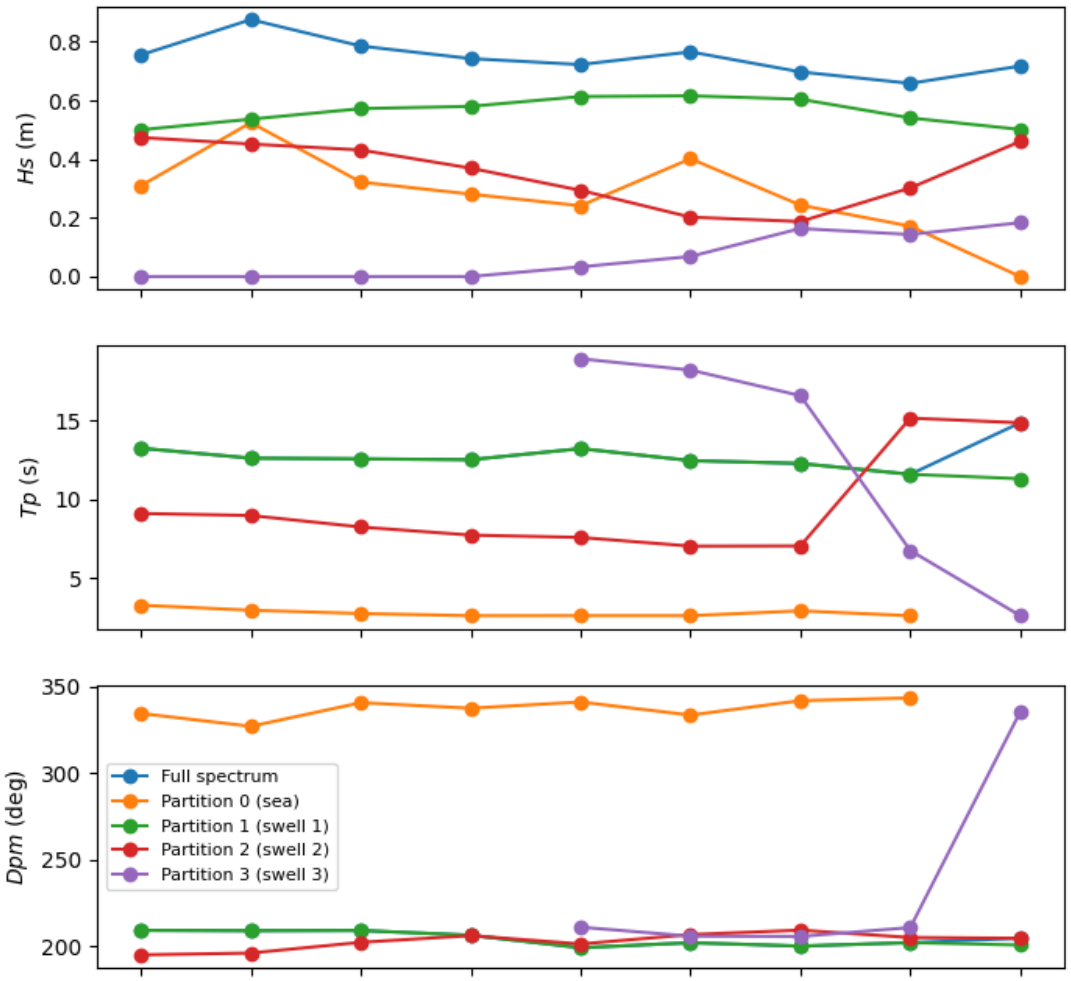
```
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/wavespectra/input/ww3.py:63: UserWarning: rename 'time' to 'time' does␣
↪not create an index anymore. Try using swap_dims instead or use set_index after rename␣
↪to create an indexed coordinate.
  dset = dset.rename(MAPPING)

Text(0.5, -8.964633430759923, '')
```

```
import matplotlib.pyplot as plt
from wavespectra import read_ww3


dset = read_ww3("../_static/ww3file.nc")

fcut = 1 / 8
sea = dset.spec.split(fmin=fcut)
swell = dset.spec.split(fmax=fcut)

plt.figure(figsize=(8, 4.5))
p1 = dset.spec.hs().isel(site=0).plot(label="Full spectrum", marker="o")
p2 = sea.spec.hs().isel(site=0).plot(label="Sea", marker="o")
p3 = swell.spec.hs().isel(site=0).plot(label="Swell", marker="o")
l = plt.legend(loc=0, fontsize=8)
plt.title("")
plt.ylabel("$Hs$ (m)")
plt.xlabel("")
```

Total running time of the script: (0 minutes 0.333 seconds)

Watershed parameters

Partition spectra and plot parameters




```
/home/docs/checkouts/readthedocs.org/user_builds/wavespectra/envs/v3.14.0/lib/python3.10/
↪site-packages/wavespectra/input/ww3.py:63: UserWarning: rename 'time' to 'time' does␣
↪not create an index anymore. Try using swap_dims instead or use set_index after rename␣
↪to create an indexed coordinate.
  dset = dset.rename(MAPPING)
```

```
import matplotlib.pyplot as plt
from wavespectra import read_ww3
```

(continues on next page)

(continued from previous page)

```
dset = read_ww3("../_static/ww3file.nc")

dspart = dset.spec.partition(dset.wspd, dset.wdir, dset.dpt)
pstats = dspart.spec.stats(["hs", "tp", "dpm"])

fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(8, 8))

# Hs
p1 = dset.spec.hs().isel(site=0).plot(ax=ax1, label="Full spectrum", marker="o")
p2 = pstats.hs.isel(part=0, site=0).plot(ax=ax1, label="Partition 0 (sea)", marker="o")
p3 = pstats.hs.isel(part=1, site=0).plot(
    ax=ax1, label="Partition 1 (swell 1)", marker="o"
)
p4 = pstats.hs.isel(part=2, site=0).plot(
    ax=ax1, label="Partition 2 (swell 2)", marker="o"
)
p5 = pstats.hs.isel(part=3, site=0).plot(
    ax=ax1, label="Partition 3 (swell 3)", marker="o"
)
ax1.set_ylabel("$Hs$ (m)")

# Tp
p5 = dset.spec.tp().isel(site=0).plot(ax=ax2, label="Full spectrum", marker="o")
p6 = pstats.tp.isel(part=0, site=0).plot(ax=ax2, label="Partition 0 (sea)", marker="o")
p7 = pstats.tp.isel(part=1, site=0).plot(
    ax=ax2, label="Partition 1 (swell 1)", marker="o"
)
p8 = pstats.tp.isel(part=2, site=0).plot(
    ax=ax2, label="Partition 2 (swell 2)", marker="o"
)
p9 = pstats.tp.isel(part=3, site=0).plot(
    ax=ax2, label="Partition 3 (swell 3)", marker="o"
)
ax2.set_ylabel("$Tp$ (s)")

# Dpm
p10 = dset.spec.dpm().isel(site=0).plot(ax=ax3, label="Full spectrum", marker="o")
p11 = pstats.dpm.isel(part=0, site=0).plot(
    ax=ax3, label="Partition 0 (sea)", marker="o"
)
p12 = pstats.dpm.isel(part=1, site=0).plot(
    ax=ax3, label="Partition 1 (swell 1)", marker="o"
)
p13 = pstats.dpm.isel(part=2, site=0).plot(
    ax=ax3, label="Partition 2 (swell 2)", marker="o"
)
p14 = pstats.dpm.isel(part=3, site=0).plot(
    ax=ax3, label="Partition 3 (swell 3)", marker="o"
)
ax3.set_ylabel("$Dpm$ (deg)")
```

(continues on next page)

(continued from previous page)

```
l = plt.legend(loc=0, fontsize=8)

for ax in [ax1, ax2, ax3]:
    ax.set_title("")
    ax.set_xlabel("")
    ax.set_xticklabels([])
```

Total running time of the script: (0 minutes 1.010 seconds)

CHAPTER

THREE

HISTORY

Wavespectra started as an internal tool developed at Metocean Solutions by Rafael Guedes, Tom Durrant and David Johnson. It was released as open source in April 2018 and received contribution from Jorge Perez. The project was transitioned into a fully community developed project in July 2019, hosted at the wavespectra github organisation.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

INDEX

Symbols

A

C

D

F

H

I

J

M

N

O

P

R

S

T

U

W