hvsrpy

Release 0.0.1

Joseph P. Vantassel

Nov 12, 2019

CONTENTS:

CHAPTER

ONE

SUMMARY

hvsrpy is a Python module for performing horizontal-to-vertical spectral ratio processing. hvsrpy was developed by Joseph P. Vantassel with contributions from Dana M. Brannon under the supervision of Professor Brady R. Cox at the University of Texas at Austin. The fully-automated frequency-domain rejection algorithm implemented in *hvsrpy* was developed by Tianjian Cheng under the supervision of Professor Brady R. Cox at the Univesity of Texas at Austin and detailed in Cox et al. (in review).

The module includes two main class definitons *Sensor3c* and *Hvsr*. These classes include various methods for creating and manipulating 3-component sensor and horizontal-to-vertical spectral ratio objects.

CHAPTER

TWO

LICENSE INFORMATION

Copyright (C) 2019 Joseph P. Vantassel (jvantassel@utexas.edu)

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https: //www.gnu.org/licenses/>.

CHAPTER THREE

SENSOR3C CLASS

class Sensor3c(*ns*, *ew*, *vt*)

Class for creating and manipulating 3-component sensor objects.

Attributes:

ns, ew, vt [Timeseries] TimeSeries object for each component.

ns_f, ew_f, vt_f [FourierTransform] FourierTransform object for each component.

__init__(*ns*, *ew*, *vt*)

Initalize a 3-component sensor (Sensor3c) object.

Args:

ns, ew, vt [timeseries] Timeseries object for each component.

Returns: Initialized 3-component sensor (Sensor3c) object.

bandpassfilter(*flow*, *fhigh*, *order*)

Bandpassfilter component TimeSeries.

Refer to *SigProPy* documentation for details.

combine_horizontals(*method='squared-average'*)

Combine two horizontal components (*ns* and *ew*).

Args:

ratio_type [{'squared-averge', 'geometric-mean'}, optional] Defines how the two horizontal components are combined to represent a single horizontal component. By default the 'squared-average' approach is used.

Return: A FourierTransform object representing the combined horizontal component.

cosine_taper(*width*)

Cosine taper component TimeSeries.

Refer to *SigProPy* documentation for details.

detrend()

Detrend component TimeSeries.

Refer to *SigProPy* documentation for details.

classmethod from_mseed(*fname*)

Initialize a 3-component sensor (Sensor3c) object from a .miniseed file.

Args:

fname [str] Name of miniseed file, full path may be used if desired. The file should contain three traces with the appropriate channel names. Refer to the *SEED* Manual here. for specifics.

Returns: Initialized 3-component sensor (Sensor3c) object.

`hv`(*windowlength*, *bp_filter*, *taper_width*, *bandwidth*, *resampling*, *method*)
Prepare time series and fourier transforms then compute H/V.

Args:

windowlength [float] Length of time windows in seconds.

bp_filter [dict] Bandpass filter settings, of the form {'flag':*bool*, 'flow':*float*, 'fhigh':*float*, 'order':*int*}.

taper_width [float] Width of cosine taper.

bandwidth [float] Bandwidth of the Konno and Ohmachi smoothing window.

resampling [dict] Resampling settings, of the form {'minf':*float*, 'maxf':*float*, 'nf':*int*, 'res_type':*str*}.

method [{'squared-averge', 'geometric-mean'}] Refer to method *combine_horizontals* for details.

Returns: Initialized Hvsr object.

Notes: More information for the above arguements can be found in the documenation of *SigProPy*.

`resample`(*fmin*, *fmax*, *fn*, *res_type*, *inplace*)
Resample component FourierTransforms.

Refer to *SigProPy* documentation for details.

`smooth`(*bandwidth*)
Smooth component FourierTransforms.

Refer to *SigProPy* documentation for details.

`split`(*windowlength*)
Split component TimeSeries.

Refer to *SigProPy* documentation for details.

`transform`()
Perform Fourier transform on components.

Returns: *None*, redefines attributes *ew_f*, *ns_f*, and *vt_f* as FourierTransform objects for each component.

CHAPTER FOUR

HVSR CLASS

class Hvsr(*amplitude*, *frequency*, *find_peaks=True*)
Class for creating and manipulating horizontal-to-vertical spectral ratio objects.

Attributes:

amp [ndarray] Array of H/V amplitudes. Each row represents an individual curve and each column a frequency.

frq [ndarray] Vector of frequencies, corresponding to each column.

n_windows [int] Number of windows in Hvsr object.

valid_window_indices [ndarray] Array of indices indicating valid windows.

__init__(*amplitude*, *frequency*, *find_peaks=True*)
Initialize a Hvsr oject from an amplitude and frequency vector.

Args:

amplitude [ndarray] Array of H/V amplitudes. Each row represents an individual curve and each column a frequency.

frequency [ndarray] Vector of frequencies, corresponding to each column.

Returns: Initialized Hvsr object.

static find_peaks(*amp*, ***kwargs*)
Returns the indices of all peaks in *amp*.

Wrapper method for scipy.signal.find_peaks function.

Args:

amp [ndarray] Vector or array of amplitudes. See *amp* attribute for details.

****kwargs** [dict] Refer to *scipy.signal.find_peaks* documentation here.

Returns:

peaks [ndarray or list] *ndarray* or *list* of *ndarrays* (one per window) of peak indices.

properties [dict] Refer to *scipy.signal.find_peaks* documentation.

mc_peak(*distribution='log-normal'*)
Peak of mean H/V curve.

Args:

distribution [{'normal', 'log-normal'}, optional] Refer to method *mean_curve* for details.

Returns: Frequency associated with the peak of the mean H/V curve.

`mean_curve`(*distribution='log-normal'*)
Return mean H/V curve.

Args:

distribution [{'normal', 'log-normal'}, optional] Assumed distribution of mean curve, default is *log-normal*.

Returns: Mean H/V curve as *ndarray* according to the distribution specified.

Raises:

KeyError: If *distribution* does not match the available options.

`mean_f0`(*distribution='log-normal'*)
Return mean value of *f0* of valid timewindows.

Args:

distribution [{'normal', 'log-normal'}] Assumed distribution of *f0*, default is *log-normal*.

Returns: Mean value of *f0* according to the distribution specified.

Raises:

KeyError: If *distribution* does not match the available options.

`nstd_curve`(*n*, *distribution*)
Return nth standard deviation curve.

Args:

n [float] Number of standard deviations away from the mean curve.

distribution [{'log-normal', 'normal'}, optional] Assumed distribution of mean curve, the default is 'log-normal'.

Return: nth standard deviation curve as an *ndarray*.

`nstd_f0`(*n*, *distribution*)
Return nth standard deviation of *f0*.

Args:

n [float] Number of standard deviations away from the mean *f0*.

distribution [{'log-normal', 'normal'}, optional] Assumed distribution of *f0*, the default is 'log-normal'.

Return: nth standard deviation of *f0* as *float*.

`property peak_amp`
Return valid peaks amplitude vector.

`property peak_frq`
Return valid peaks frequency vector.

`reject_windows`(*n=2*, *max_iterations=50*, *distribution_f0='log-normal'*, *distribution_mc='log-normal'*)
Perform rejection of H/V windows using the method proposed by Cox et al. (in review).

Args:

n [float, optional] Number of standard deviations from the mean (default value is 2).

max_iterations [int, optional] Maximum number of rejection iterations (default value is 50).

distribution_f0 [{'log-normal', 'normal'}, optional] Assumed distribution of *f0* from time windows, the default is 'log-normal'.

distribution_mc [{'log-normal', 'normal'}, optional] Assumed distribution of mean curve, the default is 'log-normal'.

Returns:

c_iteration [int] Number of iterations required for convergence.

std_curve (*distribution='log-normal'*)

Sample standard deviation associate with the mean H/V curve.

Args:

distribution [{'normal', 'log-normal'}, optional] Assumed distribution of H/V curve, default is *log-normal*.

Returns: Sample standard deviation of H/V curve as *ndarray* according to the distribution specified.

Raises:

ValueError: If only single time window is defined.

KeyError: If *distribution* does not match the available options.

std_f0 (*distribution='log-normal'*)

Return sample standard deviation of *f0* of valid timewindows.

Args:

distribution [{'normal', 'log-normal'}, optional] Assumed distribution of *f0*, default is *log-normal*.

Returns:

std [float] Sample standard deviation value according to the distribution specified.

Raises:

KeyError: If *distribution* does not match the available options.

to_file_like_geopsy (*fname*, *distribution_f0*, *distribution_mc*)

Save H/V data to file following the Geopsy format.

Args:

fname [str] Name of file to save the results, may be a full or relative path.

distribution_f0 [{'log-normal', 'normal'}, optional] Assumed distribution of *f0* from the time windows, the default is 'log-normal'.

distribution_mc [{'log-normal', 'normal'}, optional] Assumed distribution of mean curve, the default is 'log-normal'.

Returns: *None*, writes file to disk.

update_peaks (***kwargs*)

Update *peaks* attribute with the lowest frequency, highest amplitude peak.

Args:

****kwargs:** Refer to static method *find_peaks* documentation.

Returns: *None*, update *peaks* attribute.