hvsrpy

Release 0.1.0

Joseph P. Vantassel

Nov 25, 2019

CONTENTS:

1 Summary 1

2 License Information 2

3 Sensor3c Class 3

4 Hvsr Class 5

CHAPTER ONE

SUMMARY

hvsrpy is a Python module for 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 University 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, respectively.

CHAPTER

TWO

LICENSE INFORMATION

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.

normalization_factor [float] Maximum value of *ns*, *ew*, and *vt* amplitude used for normalization when plotting.

__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:

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

Returns: A *FourierTransform* object representing the combined horizontal components.

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 `combine_horizontals` for details.

Returns: Initialized *Hvsr* object.

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

property normalization_factor
Return sensor time history normalization factor.

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 component *TimeSeries*.

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 corresponds to each column.

n_windows [int] Number of windows in *Hvsr* object.

valid_window_indices [ndarray] Array of indices indicating valid windows.

rejected_window_indices [ndarray] Array of indices indicating rejected windows.

peak_frq [ndarray] Frequency peaks, one per valid time window.

peak_amp [ndarray] Amplitude(s) which correspond to *peak_frq*.

__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*)

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.

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

Amplitude of the peak of the mean H/V curve.

Args:

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

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

mc_peak_frq(*distribution='log-normal'*)
Frequency of the peak of the mean H/V curve.

Args:

distribution [{'normal', 'log-normal'}, optional] Refer to `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_amp(*distribution='log-normal'*)
Mean amplitude of *f0* of valid time windows.

Args:

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

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

Raises:

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

mean_f0_frq(*distribution='log-normal'*)
Mean *f0* of valid time windows.

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_amp(*n*, *distribution*)
nth sample standard deviation of amplitude of *f0* from time windows.

Args:

n [float] Number of standard deviations away from the mean amplitude of *f0* from valid time windows.

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

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

`nstd_f0_frq`(*n*, *distribution*)
Return nth standard deviation of *f0*.

Args:

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

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`
Valid peak amplitude vector.

`property peak_frq`
Valid peak frequency vector.

`print_stats`(*distribution_f0*)
Print basic statistics of *Hvsr* instance.

`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.

`property rejected_window_indices`
Rejected window indices.

`std_curve`(*distribution='log-normal'*)
Sample standard deviation associated 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_amp`(*distribution='log-normal'*)
Sample standard deviation of amplitude of *f0* of valid time windows.

Args:

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

Returns: Sample standard deviation of the amplitude of f0 according to the distribution specified.

Raises:

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

`std_f0_frq`(*distribution='log-normal'*)
Sample standard deviation of *f0* of valid time windows.

Args:

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

Returns: Sample standard deviation of *f0* 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 `find_peaks` documentation.

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