hvsrpy

Release 1.0.0

Joseph P. Vantassel

Feb 14, 2022

CONTENTS

hvsrpy is a Python package for horizontal-to-vertical spectral ratio processing. It includes four main class definitions *Sensor3c*, *Hvsr*, *HvsrRotated*, and *HvsrVault*. These classes include various methods for creating and manipulating 3-component sensor and horizontal-to-vertical spectral ratio objects.

This package and the classes therein are actively being developed, so if you do not see a feature you would like it may very well be under development and released in the near future. To be notified of future releases, you can either `watch` the repository on Github or `Subscribe to releases` on the Python Package Index (PyPI).

CHAPTER

ONE

CONTENTS:

1.1 Installation

`pip install hvsrpy` or `pip install hvsrpy --upgrade`

`pip` will handle the rest!

1.2 API Reference

1.2.1 Sensor3c

Class definition for Sensor3c, a 3-component sensor.

class Sensor3c(*ns*, *ew*, *vt*, *meta=None*)

Bases: `object`

Class for creating and manipulating 3-component sensor objects.

Variables

- **~Sensor3c.ns** (*TimeSeries*) – North-south component, time domain.
- **~Sensor3c.ew** (*TimeSeries*) – East-west component, time domain.
- **~Sensor3c.vt** (*TimeSeries*) – Vertical component, time domain.

__init__(*ns*, *ew*, *vt*, *meta=None*)

Initialize a 3-component sensor (Sensor3c) object.

Parameters

- **ns, ew, vt** (*TimeSeries*) – *TimeSeries* object for each component.
- **meta** (*dict, optional*) – Meta information for object, default is *None*.

Returns *Sensor3c* – Initialized 3-component sensor object.

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

Bandpassfilter components.

Refer to SigProPy documentation for details.

cosine_taper(*width*)

Cosine taper components.

Refer to SigProPy documentation for details.

`detrend()`
Detrend components.

Refer to SigProPy documentation for details.

`classmethod from_dict(dictionary)`
Create *Sensor3c* object from dictionary representation.

Parameters **dictionary** (*dict*) – Must contain keys "ns", "ew", "vt", and may also contain the optional key "meta". "ns", "ew", and "vt" must be dictionary representations of *TimeSeries* objects, see SigProPy documentation for details.

Returns *Sensor3c* – Instantiated *Sensor3c* object.

`classmethod from_json(json_str)`
Create *Sensor3c* object from Json-string representation.

Parameters **json_str** (*str*) – Json-style string, which must contain keys "ns", "ew", and "vt", and may also contain the optional key "meta". "ns", "ew", and "vt" must be Json-style string representations of *TimeSeries* objects, see SigProPy documentation for details.

Returns *Sensor3c* – Instantiated *Sensor3c* object.

`classmethod from_mseed(fname=None, fnames_1c=None)`
Create from .mseed file(s).

Parameters

- **fname** (*str, optional*) – 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, default is *None*.
- **fnames_1c** (*dict, optional*) – Some data acquisition systems supply three separate miniSEED files rather than a single combined file. To use those types of files, simply specify the three files in a *dict* of the form *{'e':'east.mseed', 'n':'north.mseed', 'z':'vertical.mseed'}*, default is *None*.

Returns *Sensor3c* – Initialized 3-component sensor object.

Raises `ValueError` – If both *fname* and *fname_verbose* are *None*.

`hv(windowlength, bp_filter, taper_width, bandwidth, resampling, method, f_low=None, f_high=None, azimuth=None)`
Prepare time series and Fourier transforms then compute H/V.

Parameters

- **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, value between *0.* and *1.*.
- **bandwidth** (*float*) – Bandwidth (b) of the Konno and Ohmachi (1998) smoothing window.
- **resampling** (*dict*) – Resampling settings, of the form *{'minf':float, 'maxf':float, 'nf':int, 'res_type':str}*.
- **method** (*{'squared-average', 'geometric-mean', 'single-azimuth', 'multiple-azimuths'}*) – Refer to `combine_horizontals` for details.
- **f_low, f_high** (*float, optional*) – Upper and lower frequency limits to restrict peak selection, default is *None* meaning search range will not be restricted.

- **azimuth** (*float, optional*) – Refer to `combine_horizontals` for details.

Returns *Hvsr* – Instantiated *Hvsr* object.

property normalization_factor
Time history normalization factor across all components.

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

Refer to SigProPy documentation for details.

to_dict()
Sensor3c object as *dict*.

Returns *dict* – Dictionary representation of a *Sensor3c*.

to_json()
Sensor3c object as JSON-string.

Returns *str* – JSON-string representation of *Sensor3c*.

transform(***kwargs*)
Perform Fourier transform on components.

Returns *dict* – With *FourierTransform*-like objects, one for for each component, indicated by the key 'ew','ns', 'vt'.

1.2.2 Hvsr

Class definition for Hvsr object.

class Hvsr(*amplitude*, *frequency*, *find_peaks=True*, *f_low=None*, *f_high=None*, *meta=None*)
Bases: `object`

Class for creating and manipulating HVSR objects.

Variables

- **~Hvsr.amp** (*ndarray*) – Array of HVSR amplitudes. Each row represents an individual curve/time window and each column a frequency.
- **~Hvsr.frq** (*ndarray*) – Vector of frequencies corresponds to each column.
- **~Hvsr.nseries** (*int*) – Number of windows in *Hvsr* object.
- **~Hvsr.valid_window_indices** (*ndarray*) – Boolean array indicating valid windows.

__init__(*amplitude*, *frequency*, *find_peaks=True*, *f_low=None*, *f_high=None*, *meta=None*)
Create *Hvsr* from iterable of amplitude and frequency.

Parameters

- **amplitude** (*ndarray*) – Array of HVSR amplitudes. Each row represents an individual curve/time window and each column a frequency.
- **frequency** (*ndarray*) – Vector of frequencies, corresponding to each column.
- **find_peaks** (*bool, optional*) – Indicates whether peaks of Hvsr will be found when created, default is *True*.
- **f_low, f_high** (*float, optional*) – Upper and lower frequency limits to restrict peak selection, default is *None* meaning search range will not be restricted.
- **meta** (*dict, optional*) – Meta information about the object, default is *None*.

Returns *Hvsr* – Initialized with *amplitude* and *frequency*.

`static correct_distribution`(*distribution*)

`static find_peaks`(*amp*, *starting_index=0*, ***kwargs*)
Indices of all peaks in *amp*.

Wrapper method for *scipy.signal.find_peaks* function.

Parameters

- **amp** (*ndarray*) – 2D array of amplitudes. See *amp* attribute for details.
- ****kwargs** (*dict*) – Refer to scipy.signal.find_peaks documentation.

Returns

Tuple – Of the form *(peaks, settings)*. Where *peaks* is a *list* of *lists* (one per window) of peak indices, and *settings* is a *dict*, refer to scipy.signal.find_peaks documentation for details.

`mc_peak_amp`(*distribution='lognormal'*)
Amplitude of the peak of the mean HVSR curve.

Parameters **distribution** (*{'normal', 'lognormal'}, optional*) – Refer to `mean_curve` for details.

Returns *float* – Amplitude associated with the peak of the mean HVSR curve.

`mc_peak_frq`(*distribution='lognormal'*)
Frequency of the peak of the mean HVSR curve.

Parameters **distribution** (*{'normal', 'lognormal'}, optional*) – Refer to `mean_curve` for details.

Returns *float* – Frequency associated with the peak of the mean HVSR curve.

`mean_curve`(*distribution='lognormal'*)
Mean HVSR curve.

Parameters **distribution** (*{'normal', 'lognormal'}, optional*) – Assumed distribution of mean curve, default is 'lognormal'.

Returns *ndarray* – Mean HVSR curve according to the distribution specified.

Raises `NotImplementedError` – If *distribution* does not match the available options.

`mean_f0_amp`(*distribution='lognormal'*)
Mean amplitude of *f0* of valid time windows.

Parameters **distribution** (*{'normal', 'lognormal'}*) – Assumed distribution of *f0*, default is 'lognormal'.

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

Raises `NotImplementedError` – If *distribution* does not match the available options.

`mean_f0_frq`(*distribution='lognormal'*)
Mean *f0* of valid time windows.

Parameters **distribution** (*{'normal', 'lognormal'}*) – Assumed distribution of *f0*, default is 'lognormal'.

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

Raises `NotImplementedError` – If *distribution* does not match the available options.

`nstd_curve`(*n*, *distribution='lognormal'*)
nth standard deviation curve.

Parameters

- **n** (*float*) – Number of standard deviations away from the mean curve.
- **distribution** (*{'lognormal', 'normal'}, optional*) – Assumed distribution of mean curve, the default is 'lognormal'.

Returns *ndarray* – nth standard deviation curve.

nstd_f0_amp(*n*, *distribution*)
Value n standard deviations from mean *f0* amplitude.

Parameters

- **n** (*float*) – Number of standard deviations away from the mean amplitude of *f0* from valid time windows.
- **distribution** (*{'lognormal', 'normal'}, optional*) – Assumed distribution of *f0*, the default is 'lognormal'.

Returns *float* – Value n standard deviations from mean *f0* amplitude.

nstd_f0_frq(*n*, *distribution*)
Value n standard deviations from mean *f0*.

Parameters

- **n** (*float*) – Number of standard deviations away from the mean *f0* for the valid time windows.
- **distribution** (*{'lognormal', 'normal'}, optional*) – Assumed distribution of *f0*, the default is 'lognormal'.

Returns *float* – Value n standard deviations from mean *f0*.

property peak_amp
Valid peak amplitude vector.

property peak_frq
Valid peak frequency vector.

print_stats(*distribution_f0*, *places=2*)
Print basic statistics of *Hvsr* instance.

reject_windows(*n=2*, *max_iterations=50*, *distribution_f0='lognormal'*, *distribution_mc='lognormal'*)
Perform rejection of spurious HVSR windows.

Parameters

- **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** (*{'lognormal', 'normal'}, optional*) – Assumed distribution of *f0* from time windows, the default is 'lognormal'.
- **distribution_mc** (*{'lognormal', 'normal'}, optional*) – Assumed distribution of mean curve, the default is 'lognormal'.

Returns *int* – Number of iterations required for convergence.

property rejected_window_indices
Rejected window indices.

std_curve(*distribution='lognormal'*)
Sample standard deviation of the mean HVSR curve.

Parameters **distribution** (*{'normal', 'lognormal'}, optional*) – Assumed distribution of HVSR curve, default is 'lognormal'.

Returns *ndarray* – Sample standard deviation of HVSR curve according to the distribution specified.

Raises

- **ValueError** – If only single time window is defined.
- **NotImplementedError** – If *distribution* does not match the available options.

std_f0_amp(*distribution='lognormal'*)
Sample standard deviation of the amplitude of f0.

Parameters **distribution** (*{'normal', 'lognormal'}, optional*) – Assumed distribution of *f0*, default is 'lognormal'.

Returns *float* – Sample standard deviation of the amplitude of f0 considering only the valid time windows.

Raises **NotImplementedError** – If *distribution* does not match the available options.

std_f0_frq(*distribution='lognormal'*)
Sample standard deviation of *f0* of valid time windows.

Parameters **distribution** (*{'normal', 'lognormal'}, optional*) – Assumed distribution of *f0*, default is 'lognormal'.

Returns *float* – Sample standard deviation of *f0*.

Raises **NotImplementedError** – If *distribution* does not match the available options.

to_file(*fname*, *distribution_f0*, *distribution_mc*, *data_format='hvsrpy'*)
Save HVSR data to summary file.

Parameters

- **fname** (*str*) – Name of file to save the results, may be a full or relative path.
- **distribution_f0** (*{'lognormal', 'normal'}, optional*) – Assumed distribution of *f0* from the time windows, the default is 'lognormal'.
- **distribution_mc** (*{'lognormal', 'normal'}, optional*) – Assumed distribution of mean curve, the default is 'lognormal'.
- **data_format** (*{'hvsrpy', 'geopsy'}, optional*) – Format of output data file, default is 'hvsrpy'.

Returns *None* – Writes file to disk.

update_peaks(***kwargs*)
Update with the lowest frequency, highest amplitude peaks.

Parameters ****kwargs** (*dict*) – Refer to *find_peaks* documentation.

Returns *None* – Updates *peaks* attribute.

1.2.3 HvsrRotated

Class definition for HvsrRotated, a rotated Hvsr measurement.

class HvsrRotated(*hvsr*, *azimuth*, *meta=None*)
Bases: `object`

Class definition for rotated Horizontal-to-Vertical calculations.

Variables

- **~HvsrRotated.hvsrs** (*list*) – Container of *Hvsr* objects, one per azimuth.
- **~HvsrRotated.azimuths** (*ndarray*) – Vector of rotation azimuths correpsonding to *Hvsr* objects.

__init__(*hvsr*, *azimuth*, *meta=None*)
Instantiate a *HvsrRotated* object.

Parameters

- **hvsr** (*Hvsr*) – *Hvsr* object.
- **azimuth** (*float*) – Rotation angle in degrees measured clockwise positive from north (i.e., 0 degrees).
- **meta** (*dict, optional*) – Meta information about the object, default is *None*.

Returns *HvsrRotated* – Instantiated *HvsrRotated* object.

property amp

append(*hvsr*, *azimuth*)
Append *Hvsr* object at a new azimuth.

Parameters

- **hvsr** (*Hvsr*) – *Hvsr* object.
- **az** (*float*) – Rotation angle in degrees measured clockwise from north (i.e., 0 degrees).

Returns *HvsrRotated* – Instantiated *HvsrRotated* object.

property azimuth_count

classmethod from_iter(*hvsrs*, *azimuths*, *meta=None*)
Create HvsrRotated from iterable of Hvsr objects.

property frq

mc_peak_amp(*distribution='lognormal'*)
Amplitude of the peak of the mean HVSR curve.

Parameters **distribution** (*{'normal', 'lognormal'}, optional*) – Refer to `mean_curve` for details.

Returns *float* – Amplitude associated with the peak of the mean HVSR curve.

mc_peak_frq(*distribution='lognormal'*)
Frequency of the peak of the mean HVSR curve.

Parameters **distribution** (*{'normal', 'lognormal'}, optional*) – Refer to `mean_curve` for details.

Returns *float* – Frequency associated with the peak of the mean HVSR curve.

mean_curve(*distribution='lognormal'*)
Mean H/V curve considering all valid windows and azimuths.

Parameters **distribution** (*{'normal', 'lognormal'}, optional*) – Assumed distribution of mean curve, default is 'lognormal'.

Returns *ndarray* – Mean HVSR curve according to the distribution specified.

Raises **NotImplementedError** – If *distribution* does not match the available options.

mean_curves(*distribution='lognormal'*)
Mean curve for each azimuth

Parameters **distribution** (*{'normal', 'lognormal'}, optional*) – Assumed distribution of mean curve, default is 'lognormal'.

Returns *ndarray* – Each row corresponds to an azimuth and each column a frequency.

mean_curves_peak(*distribution='lognormal'*)
Peak from each mean curve, one per azimuth.

mean_f0_amp(*distribution='lognormal'*)
Mean *f0* amplitude from all valid timewindows and azimuths.

Parameters **distribution** (*{'normal', 'lognormal'}, optional*) – Assumed distribution of *f0*, default is 'lognormal'.

Returns *float* – Mean amplitude of *f0* across all valid time windows and azimuths.

Raises **NotImplementedError** – If *distribution* does not match the available options.

mean_f0_frq(*distribution='lognormal'*)
Mean *f0* from all valid timewindows and azimuths.

Parameters **distribution** (*{'normal', 'lognormal'}*) – Assumed distribution of *f0*, default is 'lognormal'.

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

Raises **NotImplementedError** – If *distribution* does not match the available options.

nstd_curve(*n*, *distribution*)
Nth standard deviation on mean curve from all azimuths.

nstd_f0_frq(*n*, *distribution*)
Nth standard deviation on *f0* from all azimuths

property peak_amp
Array of peak amplitudes, one array per azimuth.

property peak_frq
Array of peak frequencies, one array per azimuth.

print_stats(*distribution_f0*, *places=2*)
Print basic statistics of *Hvsr* instance.

reject_windows(***kwargs*)

std_curve(*distribution='lognormal'*)
Sample standard deviation associated with mean HVSR curve.

Parameters **distribution** (*{'normal', 'lognormal'}, optional*) – Assumed distribution of HVSR curve, default is 'lognormal'.

Returns *ndarray* – Sample standard deviation of HVSR curve according to the distribution specified.

Raises

- **ValueError** – If only single time window is defined.
- **NotImplementedError** – If *distribution* does not match the available options.

std_f0_amp(*distribution='lognormal'*)
Sample standard deviation of *f0* amplitude for all valid time windows across all azimuths.

Parameters **distribution** (*{'normal', 'lognormal'}, optional*) – Assumed distribution of *f0*, default is 'lognormal'.

Returns *float* – Sample standard deviation of the amplitude of *f0* according to the distribution specified.

Raises **NotImplementedError** – If *distribution* does not match the available options.

std_f0_frq(*distribution='lognormal'*)
Sample standard deviation of *f0* for all valid time windows across all azimuths.

Parameters **distribution** (*{'normal', 'lognormal'}, optional*) – Assumed distribution of *f0*, default is 'lognormal'.

Returns *float* – Sample standard deviation of *f0*.

Raises **NotImplementedError** – If *distribution* does not match the available options.

to_file(*fname*, *distribution_f0*, *distribution_mc*, *data_format='hvsrpy'*)
Save HVSR data to file.

Parameters

- **fname** (*str*) – Name of file to save the results, may be the full or a relative path.
- **distribution_f0** (*{'lognormal', 'normal'}, optional*) – Assumed distribution of *f0* from the time windows, the default is 'lognormal'.
- **distribution_mc** (*{'lognormal', 'normal'}, optional*) – Assumed distribution of mean curve, the default is 'lognormal'.
- **data_format** (*{'hvsrpy'}, optional*) – Format of output data file, default is 'hvsrpy'.

Returns *None* – Writes file to disk.

1.2.4 HvsrSpatial

HvsrVault class definition.

class HvsrVault(*coordinates*, *means*, *stddevs=None*)
Bases: `object`

A container for Hvsr objects.

Variables

- **~HvsrVault.coordinates** (*ndarray*) – Relative x and y coordinates of the sensors, where each row of the *ndarray* in an x, y pair.
- **~HvsrVault.means** (*ndarray*) – Mean f0 value for each location, meaning is determined by *distribution* keyword argument.
- **~HvsrVault.stddevs** (*ndarray, optional*) – Standard deviation for each location, meaning is determined by *distribution* keyword argument, default is *None* indicating no uncertainty is defined.

__init__(*coordinates*, *means*, *stddevs=None*)
Create a container for *Hvsr* statistics.

Parameters

- **coordinates** (*ndarray*) – Relative x and y coordinates of the sensors, where each row of the *ndarray* in an x, y pair.
- **means** (*ndarray*) – Mean f0 value for each location, meaning is determined by *distribution* keyword argument.
- **stddevs** (*ndarray, optional*) – Standard deviation for each location, meaning is determined by *distribution* keyword argument, default is *None* indicating no uncertainty is defined.
- **distribution** (*{'normal', 'lognormal'}, optional*) – Distribution to which the mean and stddev for each point corresponds, default is 'lognormal'.

`bounded_voronoi`(*boundary*)

Vertices of bounded Voronoi region.

Parameters **boundary** (*ndarray*) – x, y coordinates defining the spatial boundary. Must be of shape *(N, 2)*.

Returns *tuple* – Of the form *(new_vertices, indices)* where *new_vertices* defines the vertices of each region and *indices* indicates how these vertices relate to the master statistics.

`spatial_weights`(*boundary*, *dc_method='voronoi'*)

Calculate the weights for each Voronoi region.

Parameters

- **boundary** (*ndarray*) – x, y coordinates defining the spatial boundary. Must be of shape *(N, 2)*.
- **dc_method** (*{"voronoi"}, optional*) – Declustering method, default is 'voronoi'.

Returns *tuple* – Of the form *(weights, indices)* where *weights* are the statistical weights and *indicates* the bounding box of each cell.

`montecarlo_f0`(*mean*, *stddev*, *weights*, *dist_generators='lognormal'*, *dist_spatial='lognormal'*, *nrealizations=1000*, *generator='PCG64'*)

MonteCarlo simulation for spatial distribution of f0.

Parameters

- **mean, stddev** (*ndarray*) – Mean and standard deviation of each generating point. Meaning of these parameters is dictated by *dist_generators*.
- **weights** (*ndarray*) – Weights for each generating point.
- **dist_generators** (*{'lognormal', 'normal'}, optional*) – Assumed distribution of each generating point, default is *lognormal*.

if dist is	mean must be	stddev must be
normal	μ	σ
lognormal	λ	ζ

- **dist_spatial** (*{'lognormal', 'normal'}, optional*) – Assumed distribution of spatial statistics on f0, default is *lognormal*.
- **generator** (*{'PCG64', 'MT19937'}, optional*) – Bit generator, default is *PCG64*.

Returns *tuple* – Of the form *(f0_mean, f0_stddev, f0_realizations)*.

1.3 CLI Reference

1.3.1 hvsrpy

Command line interface to hvsrpy.

```
hvsrpy [OPTIONS] [FILE_NAMES]...
```

Options

--config <config>
Path to configuration file.

--windowlength <windowlength>
Length of time windows, default is 60 seconds.

--filter_bool <filter_bool>
Controls whether a bandpass filter is applied, default is False.

--filter_flow <filter_flow>
Low frequency limit of bandpass filter in Hz, default is 0.1.

--filter_fhigh <filter_fhigh>
High frequency limit of bandpass filter in Hz, default is 30.

--filter_order <filter_order>
Filter order, default is 5 (i.e., 5th order filter).

--width <width>
Length of cosine taper, default is 0.1 (5% on each side) of time window.

--bandwidth <bandwidth>
Bandwidth coefficient for Konno & Ohmachi (1998) smoothing, default is 40.

--resample_fmin <resample_fmin>
Minimum frequency in Hz to consider when resampling, defaults is 0.2.

--resample_fmax <resample_fmax>
Maximum frequency in Hz to consider when resampling, defaults is 20.

--resample_fnum <resample_fnum>
Number of samples in resampled curve, default is 128.

--resample_type <resample_type>
Type of resampling, default is 'log'.

Options log | linear

--peak_f_lower <peak_f_lower>
Lower frequency limit of peak selection, defaults to entire range.

--peak_f_upper <peak_f_upper>
Upper frequency limit of peak selection, defaults to entire range.

--method <method>
Method for combining the horizontal components, default is 'geometric-mean'.

Options squared-average | geometric-mean | single-azimuth | multiple-azimuths

--azimuth <azimuth>
Azimuth to orient horizontal components when method is 'single-azimuth', default is 0.

--azimuthal_interval <azimuthal_interval>
Interval in degrees between azimuths when method is 'multiple-azimuths', default is 15.

--rejection_bool <rejection_bool>
Determines whether the rejection is performed, default is True.

--n <n>
Number of standard deviations to consider when performing the rejection, default is 2.

--max_iterations <max_iterations>
Number of permitted iterations to convergence, default is 50.

--distribution_f0 <distribution_f0>
Distribution assumed to describe the fundamental site frequency, default is 'lognormal'.

Options lognormal | normal

--distribution_mc <distribution_mc>
Distribution assumed to describe the median curve, default is 'lognormal'.

Options lognormal | normal

--no_time
Flag to suppress HVSR compute time.

--no_figure
Flag to prevent figure creation.

--ymin <ymin>
Manually set the lower y limit of the HVSR figure.

--ymax <ymax>
Manually set the upper y limit of the HVSR figure.

--summary_type <summary_type>
Summary file format to save, default is 'hvsrpy'.

Options none | hvsrpy | geopsy

--nproc <nproc>
Number of subprocesses to launch, default is number of cpus minus 1.

Arguments

FILE_NAMES
Optional argument(s)

1.4 License Information

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

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX

Symbols

A

B

C

D

F

U