SigProPy

Release 0.0.1

Joseph P. Vantassel

Nov 12, 2019

CONTENTS:

CHAPTER ONE

SUMMARY

SigProPy is Python module for digital signal processing. The module includes two main class definitons *TimeSeries* and *FourierTransform*. These classes include various methods for creating and manipulating time series and Fourier transforms.

CHAPTER

TWO

LICENSE INFORMATION

CHAPTER

THREE

TIMESERIES CLASS

class TimeSeries(*amplitude*, *dt*, *n_stacks=1*, *delay=0*)
A class for manipulating time series.

Attributes:

amp [ndarray] Denotes the time series amplitude. If *amp* is 1D each sample corresponds to a single time step. If *amp* is 2D each row corresponds to a particular section of the time record (i.e., time window) and each column corresponds to a single time step.

dt [float] Denotes the time step between samples in seconds.

n_windows [int] Number of time windows that the time series has been split into (i.e., number of rows of amp if 2D).

n_samples [int] Number of samples in time series (i.e., *len(amp)* if *amp* is 1D or number of columns if *amp* is 2D).

fs [float] Sampling frequency in Hz equal to *1/dt*.

fnyq [float] Nyquist frequency in Hz equal to *fs/2*.

__init__(*amplitude*, *dt*, *n_stacks=1*, *delay=0*)
Initialize a TimeSeries object.

Args:

amplitude [ndarray] Amplitude of the time series at each time step. Refer to attribute definition for details.

dt [float] Time step between samples in seconds.

n_stacks [int, optional] Number of stacks used to produce the amplitude (the default value is 1, denoting a single unstacked recording).

delay [float {<=0.}, optional] Indicates the pre-event delay in seconds.

Returns: Intialized TimeSeries object.

Raises:

ValueError: If *delay* is greater than 0.

bandpassfilter(*flow*, *fhigh*, *order=5*)
Apply bandpass Butterworth filter to time series.

Args:

flow [float] Low-cut frequency (content below *flow* is filtered).

fhigh [float] High-cut frequency (content above *fhigh* is filtered).

order [int, optional] Filter order (default is 5th).

Returns: *None*, instead filters attribute *amp*.

cosine_taper(*width*)

Apply cosine taper to time series.

Args:

width [float {0.-1.}] Amount of the time series to be tapered. 0. is equal to a rectangular and 1. a Hann window.

Returns: *None*, applies cosine taper to attribute *amp*.

detrend()

Remove linear trend from time series.

Returns: *None*, remove linear trend from attribute *amp*.

classmethod from_trace(*trace*, *n_stacks=1*, *delay=0*)

Initialize a TimeSeries object from a trace object.

This method is a more general method than *from_trace_seg2*, as it does not attempt to extract any metadata from the Trace object.

Args:

trace [Trace] Refer to obspy documentation for more information (https://github.com/obspy/obspy/wiki).

n_stacks [int, optional] Number of stacks the time series represents, (default is 1, signifying a single unstacked time record).

delay [float {<=0.}, optional] Denotes the pre-event delay, (default is zero, signifying no pre-event recording is included).

Returns: Initialized TimeSeries object.

split(*windowlength*)

Split time series into windows of duration *windowlength*.

Args:

windowlength [float] Duration of desired window length in seconds. If *windowlength* is not an integer multiple of *dt*, the window length is rounded to up to the next integer multiple of *dt*.

Returns: *None*, reshapes attribute *amp* into a 2D array where each row is a different consecutive time window and each column denotes a time step.

Note: The last sample of each window is repeated as the first sample of the following time window to ensure a logical number of windows. Without this, a 10 minute record could not be broken into 10 1-minute records.

Example:

```
>>> import sigpropy as sp
>>> import numpy as np
>>> amp = np.array([0,1,2,3,4,5,6,7,8,9])
>>> tseries = sp.TimeSeries(amp, dt=1)
>>> tseries.split(2)
>>> tseries.amp
array([[0, 1, 2],
    [2, 3, 4],
```

(continues on next page)

(continued from previous page)

```
    [4, 5, 6],
    [6, 7, 8]])
```

property time
Return time vector for TimeSeries object.

trim(*start_time*, *end_time*)
Trim excess from time series in the half-open interval [start_time, end_time).

Args:

start_time [float] New time zero in seconds.

end_time [float] New end time in seconds. Note that the interval is half-open.

Returns: *None*,updates the attributes: *n_samples*, *delay*, and *df*.

Raises:

IndexError: If the *start_time* and *end_time* is illogical. For example, *start_time* is before the start of the *delay* or after *end_time*, or the *end_time* is after the end of the record.

zero_pad(*df*)
Append zeros to the end of the TimeSeries object to achieve a desired frequency step.

Args:

df [float] Desired frequency step in Hz. Must be positive.

Returns: *None*, modifies attributes: *amp*, *n_samples*, and *multiple*.

Raises:

TypeError: If *df* is not a float.

ValueError: If *df* is not positive.

CHAPTER

FOUR

FOURIERTRANSFORM CLASS

class FourierTransform(*amplitude*, *frq*, *fnyq=None*)

A class for manipulating Fourier transforms.

Attributes:

frq [ndarray] Frequency vector of the transform in Hz.

amp [ndarray] The transform's amplitude is in the same units as the input. May be 1D or 2D. If 2D each row corresponds to a unique FFT, where each column correpsonds to an entry in *frq*.

fnyq [float] The Nyquist frequency associated with the time series used to generate the Fourier transform. Note this may or may not be equal to *frq[-1]*.

__init__(*amplitude*, *frq*, *fnyq=None*)

Initialize a FourierTransform object.

Args:

amplitude [ndarray] Fourier transform amplitude. Refer to attribute *amp* for more details.

frq [ndarray] Linearly spaced frequency vector for Fourier transform.

fnyq [float, optional] Nyquist frequency of Fourier Transform (by default the maximum value of *frq* vector is used).

Returns: An initialized FourierTransform object.

static fft(*amplitude*, *dt*)

Compute the fast-Fourier transform (FFT) of a time series.

Args:

amplitude [ndarray] Denotes the time series amplitude. If *amplitude* is 1D each sample corresponds to a single time step. If *amplitude* is 2D each row corresponds to a particular section of the time record (i.e., time window) and each column corresponds to a single time step.

dt [float] Denotes the time step between samples in seconds.

Returns:

Tuple of the form (frq, fft) where:

frq [ndarray] Positve frequency vector between zero and the Nyquist frequency (if even) or near the Nyquist (if odd) in Hz.

fft [ndarray] Complex amplitudes for the frequencies between zero and the Nyquist (if even) or near the Nyquist (if odd) with units of the input ampltiude. If *amplitude* is a 2D array *fft* will also be a 2D array where each row is the FFT of each row of *amplitude*.

classmethod from_timeseries(*timeseries*)

Create a FourierTransform object from a TimeSeries object.

Args:

timeseries [TimeSeries] TimeSeries object to be transformed.

Returns: An initialized FourierTransform object.

property imag

Imaginary component of complex FFT amplitude.

property mag

Magnitude of complex FFT amplitude.

property phase

Phase of complex FFT amplitude in radians.

property real

Real component of complex FFT amplitude.

resample(*minf*, *maxf*, *nf*, *res_type='log'*, *inplace=False*)

Resample FourierTransform over a specified range.

Args:

minf [float] Minimum value of resample.

maxf [float] Maximum value of resample.

nf [int] Number of resamples.

res_type [{"log", "linear"}, optional] Type of resampling, default value is *log*.

inplace [bool, optional] Determines whether resampling is done in place or if a copy is returned be returned. By default the resampling is not done inplace (i.e., *inplace=False*).

Returns:

If *inplace=True* *None*, method edits the internal attribute *amp*.

If *inplace=False* A tuple of the form (*frequency*, *amplitude*) where *frequency* is the resampled frequency vector and *amplitude* is the resampled amplitude vector if *amp* is 1D or array if *amp* is 2D.

Raises:

ValueError: If *maxf*, *minf*, or *nf* are illogical.

NotImplementedError: If *res_type* is not amoung those options specified.

smooth_konno_ohmachi(*bandwidth=40.0*)

Apply Konno and Ohmachi smoothing.

Args:

bandwidth [float, optional] Width of smoothing window, by default this is set to 40.

Returns: *None*, modifies the internal attribute *amp* to equal the smoothed value of *mag*.