SigProPy

Release 0.0.1

Joseph P. Vantassel

Nov 08, 2019

CONTENTS:

SigProPy is a digital signal processing module for python. The module includes two main classes a TimeSeries and FourierTransform class. These classes include various methods for creating and manipulating time series and fourier transforms.

CHAPTER

ONE

TIMESERIES CLASS

This file contains the class TimeSeries for creating and manipulating time series objects.

class TimeSeries (*amplitude*, *dt*, *nstacks=1*, *delay=0*)

A class for editing and manipulating time series.

Attributes:

amp: np.array 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*.

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

bandpassfilter (*flow*, *fhigh*, *order=5*)

Bandpass filter TimeSeries.

Args:

flow: float Low cut-off frequency for filter (content below flow is filtered).

fhigh: float High cut-off frequency for filter (content above fhigh is filtered).

order: int Filter order

Returns: Returns *None*, perform filter operation on attribute *amp*.

static check_input (*name*, *values*)

Perform simple checks on values of parameter *name*.

Specifically:

1. Check *values* is *ndarray*, *list*, or *tuple*.
2. If *list* or *tuple* convert to a *ndarray*.
3. Check *values* is one-dimensional *ndarray*.

Args:

name [str] *name* of parameter to be check. Only used to raise easily understood exceptions.

values [any] value of parameter to be checked.

Returns: *values* as an *ndarray*.

Raises:

TypeError If entries do not comply with checks 1. and 2. listed above.

cosine_taper (*width*)
Apply cosine taper to TimeSeries.

Args:

width [float {0.-1.}] Amount of the TimeSeries to be tapered. 0 represents a rectangular window and 1 a Hann window.

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

classmethod from_trace (*trace*, *nstacks=1*, *delay=0*)
Initialize a TimeSeries object from a trace object.

This method is a more general method than from_trace_seg2, which does not attempt to extract anything from the trace object except for its data and sampling step.

Args: trace: Trace object.

nstacks: Integer representing the number of stacks. Default value is one (i.e., only a single recording was made).

delay: Number that is less than or equal to zero, denoting the pre-trigger delay. Default value is zero (i.e., no pre-trigger delay was recorded).

Returns: Initialized TimeSeries object.

Raises: This method raises no exceptions.

split (*windowlength*)
Split TimeSeries into windows of length windowlength.

Args:

windowlength [int] Integer defining length of window in seconds. If *windowlength* is not integer multiple of dt, the window length is rounded to up to the next integer multiple of dt.

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

property time
Return time vector for TimeSeries object.

trim(*start_time*, *end_time*)
Trim excess from TimeSeries object 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: Returns *None*, instead 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 before the *start_time* or after the end of the record.

zero_pad(*df=0.2*)
Append zeros to the end of the TimeSeries object to achieve a desired frequency step.

Args:

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

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

Raises:

TypeError If *df* is not a float.

ValueError If *df* is not positive.

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

Attributes:

amp: np.array 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*.

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

bandpassfilter(*flow*, *fhigh*, *order=5*)
Bandpass filter TimeSeries.

Args:

flow: float Low cut-off frequency for filter (content below flow is filtered).

fhigh: float High cut-off frequency for filter (content above fhigh is filtered).

order: int Filter order

Returns: Returns *None*, perform filter operation on attribute *amp*.

static check_input(*name*, *values*)
Perform simple checks on values of parameter *name*.

Specifically:

1. Check *values* is *ndarray*, *list*, or *tuple*.
2. If *list* or *tuple* convert to a *ndarray*.
3. Check *values* is one-dimensional *ndarray*.

Args:

name [str] *name* of parameter to be check. Only used to raise easily understood exceptions.

values [any] value of parameter to be checked.

Returns: *values* as an *ndarray*.

Raises:

TypeError If entries do not comply with checks 1. and 2. listed above.

cosine_taper(*width*)

Apply cosine taper to TimeSeries.

Args:

width [float {0.-1.}] Amount of the TimeSeries to be tapered. 0 represents a rectangular window and 1 a Hann window.

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

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

Initialize a TimeSeries object from a trace object.

This method is a more general method than from_trace_seg2, which does not attempt to extract anything from the trace object except for its data and sampling step.

Args: trace: Trace object.

nstacks: Integer representing the number of stacks. Default value is one (i.e., only a single recording was made).

delay: Number that is less than or equal to zero, denoting the pre-trigger delay. Default value is zero (i.e., no pre-trigger delay was recorded).

Returns: Initialized TimeSeries object.

Raises: This method raises no exceptions.

split(*windowlength*)

Split TimeSeries into windows of length windowlength.

Args:

windowlength [int] Integer defining length of window in seconds. If *windowlength* is not integer multiple of dt, the window length is rounded to up to the next integer multiple of dt.

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

property time
Return time vector for TimeSeries object.

trim(*start_time*, *end_time*)
Trim excess from TimeSeries object 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: Returns *None*, instead 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 before the *start_time* or after the end of the record.

zero_pad(*df=0.2*)
Append zeros to the end of the TimeSeries object to achieve a desired frequency step.

Args:

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

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

Raises:

TypeError If *df* is not a float.

ValueError If *df* is not positive.

CHAPTER

TWO

FOURIERTRANSFORM CLASS

This file contains the class FourierTransform for creating and working with fourier transform objects.

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

A class for editing and manipulating fourier transforms.

Attributes:

frq [np.array] Frequency vector of the transform in Hertz.

amp [np.array] The transform's amplitude in the same units as the input.

static fft(*amplitude*, *dt*)

Compute the fast-fourier transform of a time series.

Args:

amplitude [np.array] Time series amplitudes (one per time step). Can be a 2D array where each row is a valid time series.

dt [float] Indicating the time step in seconds.

Returns: Tuple of the form (frq, fft) where: frq is an np.array containing the positve frequency vector

between zero and the nyquist frequency (if even) or near the nyquist (if odd) in Hertz.

fft is an np.array of complex amplitudes for the frequencies between zero and the nyquist 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*)

Compute the Fast Fourier Transform from a timeseries.

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 or resample.

nf [int] Number of resamples.

res_type [{"log", "linear"}] Type of resampling.

inplace [bool] Determines whether resampling is done in place or if a copy should be returned.

Returns: If inplace=True, None.

If inplace=False, a tuple of the form (frequency,) where each parameter is a list.

Raises:

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

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

class FourierTransform(*amplitude*, *frq*, *fnyq=None*)
A class for editing and manipulating fourier transforms.

Attributes:

frq [np.array] Frequency vector of the transform in Hertz.

amp [np.array] The transform's amplitude in the same units as the input.

static fft(*amplitude*, *dt*)
Compute the fast-fourier transform of a time series.

Args:

amplitude [np.array] Time series amplitudes (one per time step). Can be a 2D array where each row is a valid time series.

dt [float] Indicating the time step in seconds.

Returns: Tuple of the form (frq, fft) where: frq is an np.array containing the positve frequency vector

between zero and the nyquist frequency (if even) or near the nyquist (if odd) in Hertz.

fft is an np.array of complex amplitudes for the frequencies between zero and the nyquist 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*)
Compute the Fast Fourier Transform from a timeseries.

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 or resample.

nf [int] Number of resamples.

res_type [{"log", "linear"}] Type of resampling.

inplace [bool] Determines whether resampling is done in place or if a copy should be returned.

Returns: If inplace=True, None.

If inplace=False, a tuple of the form (frequency,) where each parameter is a list.

Raises:

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

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

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search