sigpropy

Release 1.0.0

Jul 21, 2021

Contents

sigpropy is a Python package for digital signal processing. It includes two main class definitions, *TimeSeries* and *FourierTransform*. These classes include methods to perform common signal processing techniques (e.g., trimming and resampling) and properties to make using them readable and intuitive.

This package and the classes therein are being used in several other Python projects, some of which have been released publicly and others are still in the development stage, 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 1

Contents:

1.1 Installation

`pip install sigpropy` or `pip install sigpropy --upgrade`

`pip` will handle the rest!

1.2 API Reference

1.2.1 TimeSeries

TimeSeries class definition.

class TimeSeries(*amplitude*, *dt*)

Bases: `object`

A class for manipulating time series.

Variables

- ***amplitude*** (*ndarray*) – Denotes the time series amplitude one value per time step. Amplitude can be 1D or 2D, for the 2D case each row is a different time series.
- ***dt*** (*float*) – Time step between samples in seconds.

__init__(*amplitude*, *dt*)

Initialize a *TimeSeries* object.

Parameters

- **amplitude** (*ndarray*) – Amplitude of the time series at each time step.
- **dt** (*float*) – Time step between samples in seconds.

Returns *TimeSeries* – Instantiated with amplitude information.

Raises `TypeError` – If *amplitude* is not castable to *ndarray* or has dimensions greater than 2. Refer to error message(s) for specific details.

amp

amplitude

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

Parameters

- **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 5.

Returns *None* – Filters attribute *amplitude*.

cosine_taper(*width*)
Apply cosine taper to time series.

Parameters **width** (*{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 *amplitude*.

detrend()
Remove linear trend from time series.

Returns *None* – Removes linear trend from attribute *amplitude*.

df

dt

fnyq

classmethod from_dict(*dictionary*)
Create *TimeSeries* object from dictionary representation.

Parameters **dictionary** (*dict*) – Must contain keys "amplitude" and "dt".

Returns *TimeSeries* – Instantiated *TimeSeries* object.

Raises `KeyError` – If any of the required keys (listed above) are missing.

classmethod from_json(*json_str*)
Instantiate *TimeSeries* object form Json string.

Parameters **json_str** (*str*) – Json string with all of the relevant contents of *TimeSeries*. Must contain keys "amplitude" and "dt".

Returns *TimeSeries* – Instantiated *TimeSeries* object.

classmethod from_timeseries(*timeseries*)
Copy constructor for *TimeSeries* object.

Parameters **timeseries** (*TimeSeries*) – *TimeSeries* to be copied.

Returns *TimeSeries* – Copy of the provided *TimeSeries* object.

classmethod from_trace(*trace*)
Initialize a *TimeSeries* object from a trace object.

Parameters **trace** (*Trace*) – Refer to obspy documentation for more information

Returns *TimeSeries* – Initialized with information from *trace*.

fs

join()
Rejoin a split *TimeSeries*.

Returns *None* – Updates the object's internal attributes (e.g., *amplitude*).

n_samples

n_windows

nsamples

nsamples_per_window

nseries

nwindows

split(*windowlength*)
Split record into *n* series of length *windowlength*.

Parameters **windowlength** (*float*) – Duration of desired shorter series 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* – Updates the object's internal attributes (e.g., *amplitude*).

Notes

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

Examples

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

time

to_dict()
Dictionary representation of *TimeSeries*.

Returns *dict* – Containing all of the relevant contents of the *TimeSeries*.

to_json()
Json string representation of *TimeSeries* object.

Returns *str* – Json string with all of the relevant contents of the *TimeSeries*.

trim(*start_time*, *end_time*)
Trim time series in the interval [*start_time*, *end_time*].

Parameters

- **start_time** (*float*) – New time zero in seconds.
- **end_time** (*float*) – New end time in seconds.

Returns *None* – Updates the attributes *amplitude* and *nsamples*.

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.

windowlength

1.2.2 FourierTransform

FourierTransform class definition.

class FourierTransform(*amplitude*, *frequency*, *fnyq=None*, *dtype=<class 'complex'>*)
Bases: `object`

A class for manipulating Fourier transforms.

Variables

- ***frequency*** (*ndarray*) – Frequency vector of the transform in Hz.
- ***amplitude*** (*ndarray*) – The transform's amplitude in the same units as the input. May be 1D or 2D. If 2D each row corresponds to a unique FFT, where each column corresponds to an entry in *frequency*.
- ***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 *frequency[-1]*.

__init__(*amplitude*, *frequency*, *fnyq=None*, *dtype=<class 'complex'>*)
Initialize a *FourierTransform* object.

Parameters

- **amplitude** (*ndarray*) – Fourier transform amplitude.
- **frequency** (*ndarray*) – Linearly spaced frequency vector for Fourier transform.
- **fnyq** (*float, optional*) – Nyquist frequency of Fourier transform, default is *max(frequency)*.

Returns *FourierTransform* – Initialized with *amplitude* and *frequency* information.

amp

amplitude

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

Parameters

- **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.
- ****kwargs** (*dict*) – Additional keyard arguments to fft.

Returns

Tuple – Of the form (frq, fft) where:

frq ndarray of frequencies between zero and the Nyquist frequency (if even) or near the Nyquist (if odd) in Hz.

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

frequency

classmethod from_timeseries(*timeseries*, ***fft_kwargs*)
Create *FourierTransform* from *TimeSeries*.

Parameters

- **timeseries** (*TimeSeries*) – *TimeSeries* object to be transformed.
- ****fft_kwargs** (*dict*) – Custom settings for fft.

Returns *FourierTransform* – Initialized with information from *TimeSeries*.

frq

imag
Imaginary component of complex FFT amplitude.

mag
Magnitude of complex FFT amplitude.

phase
Phase of complex FFT amplitude in radians.

real
Real component of complex FFT amplitude.

resample(*minf*, *maxf*, *nf*, *res_type='log'*, *inplace=False*)
Resample *FourierTransform* over a specified range.

Parameters

- **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 to be returned. By default the resampling is not done inplace (i.e., *inplace=False*).

Returns

None or Tuple –

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

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 *amplitude* is 1D or array if *amplitude* is 2D.

Raises

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

- `NotImplementedError` – If *res_type* is not among those options specified.

smooth_konno_ohmachi (*bandwidth=40.0*)
Apply Konno and Ohmachi smoothing.

Parameters **bandwidth** (*float, optional*) – Width of smoothing window, default is 40.

Returns *None* – Modifies the internal attribute *amplitude* to equal the smoothed value of *mag*.

smooth_konno_ohmachi_fast (*frequencies*, *bandwidth=40*)
Apply fast Konno and Ohmachi smoothing.

Parameters

- **frequencies** (*array-like*) – Frequencies at which the smoothing is performed. If you choose to use all of the frequencies from the FFT for this parameter you should not expect much speedup over *smooth_konno_ohmachi*.
- **bandwidth** (*float, optional*) – Width of smoothing window, default is 40.

Returns *None* – Modifies the internal attribute *amplitude* to equal the smoothed value of *mag*.

1.3 License Information

Copyright (C) 2019-2020 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 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

W