Advices for calculating spectra for seismic analysis

DISCLAIMER

This document is not intended as a user manual but to just give few advices about which functions can be used to calculate spectra in SciPy and ObsPy for the relevant analyses for site characterization studies. For the details of these functions, please refer to the appropriate documentation available at the SciPy and ObsPy project webpages.

Introduction

Spectral analysis of seismic data is at the basis of seismic characterization of the Einstein Telescope candidate sites. Softwares and programming languages offer a pletora of functions to calculate spectra. The scope of this brief report is to give an overview of the functions and procedures that are used in seismic characterization analysis for the Sos Enattos site and to show which are best suited for some specific analyses.

Spectra in SciPy and ObsPy

SciPy and ObsPy are the two modules used to calculate spectra in the seismic characterization analysis for Sos Enattos using respectively the functions `scipy.fft.fft` and `obspy.PPSD`. Each function offers a different approach.

ObsPy's PSD is calculated using the procedure proposed by McNamara & Buland [2004]. The fft is calculated through the Welch method included in SciPy and then the frequency range of the spectra is divided into octaves in which an average is computed between the lower and upper limits of the octave. This averaging creates heavily smoothed PSDs. The advantage of this function is that it requires very little coding and plotting, manipulating and saving the spectra is straightforward. Natively, `obspy.PPSD` is conceived to plot PPSD (spectral histograms) but the spectra of each time segment and other relevant quantities can be easily extracted by the user. Also, extracting the time series of the spectra at given frequencies is fast and easy. The drawback is that the smoothing erases relevant information such as transient peaks and allows only to have an idea about general noise levels and data quality issues. Furthermore the method is closed and not very flexible in the choice of parameters.

On the other hand, SciPy functions are the cornerstones of spectra calculation in Python. The most basic function available is `scipy.fft.fft` and 99% of Python functions that calculate spectra (e.g. `np.fft, scipy.signal.welch,` `scipy.signal.spectrogram`), use this function. The former function is a basic function that outputs the complex spectra of an input array, whereas the others give the opportunity to window and taper the spectra taking these parameters as input. Therefore, windowing, tapering and other manipulation must be coded explicitly by the user when using `scipy.fft.fft`. It should be noted, though, that `scipy.fft.fft` is one of the few, if not the only, functions that outputs complex spectra, a feature that turns out to be useful in some cases.

Obviously, the use of SciPy's functions requires more coding that ObsPy, since some features have to be recreated by the user, e.g. calculation of the spectrograms, retrieval of the times of segments, plotting of histograms, reconstruction of time series.

When to use ObsPy

Although limited by smoothing, ObsPy PSD is useful to understand general noise levels, data quality and the evolution of microseisms in parallel with the evolution of noise. In fact, this function is used to show PPSD plots, evolution of day and night noise levels, spectral ratios and correlation with sea waves in which it turns out to be extremely useful.

In fact, time series of the spectra and the time series of sea wave height are in a similar format and it is easy to find coincidences in time since, in both cases, the time stamps are saved for each sample of the time series.

Summary

- Seasonality of microseismic noise;
- visualization of PPSD to understand noise levels;
- evolution of noise levels;
- extraction of time series;
- correlation with sea wave height.

How to

- Calculate spectra with default parameters (3600 s segments with 50% overlap):

  `ppsd_obj = PPSD(trace.stats, metadata = inventory)`
- Get the period binning: `ppsd_obj._period_binning[0]`
- Get the GPS times of each segment of the spectra (in datetime object format):

  `ppsd_obj._times_processed[segment_index]`
- Get the spectra segments: `ppsd_obj._binned_psds[segment_index]`
- Extract the time series at a given period: `time_series,_,_,_ =`

  `ppsd_obj.extract_psd_values(period=T)`

When to use SciPy

Aside from the evaluation of general noise levels, SciPy's functions turn out to be useful for the rest of the analyses. For example, the algorithm for the calculation of the polarization in the direction of microseisms (Tanimoto et al. 2006) requires the use of complex spectra for which the use of `scipy.fft.fft` is unavoidable. Therefore, the user must window and taper the data according to his/her preferences on his/her own.

The same applies for the calculation of H/V ratios. The use of ObsPy's PPSD will provide less robust results because of its heavy smoothing. Therefore, the user must once again rely on SciPy. Nevertheless, in this case the choice of the function to use is left to the user. In fact, the H/V ratio procedure requires the use of real spectra. This leaves some freedom in the choice of the functions, since the user can choose `scipy.fft.fft` and build his/her own procedure, to have more control on the calculation of the spectra, or can use `scipy.signal.welch` or `scipy.signal.spectrogram` to simplify things.

It is worth pointing out that smoothing in general is not a deprecated procedure in data analysis. Smoothing can be used to reduce the "noise" of spectra when this can affect the analysis. For example, in the calculation of the directionality of microseisms, one has to get the location of the microseismic peak. If the spectra are raw, this may not be precise because spikes will affect the automatic finding of the broad microseismic peak. Nevertheless a light gaussian smoothing will solve this issue and will also provide better visualization.

Summary

- Directionality of microseisms;
- calculation of H/V ratio;
- identification of noise peaks and noise transients;
- visualization of non smoothed PPSD.

How to

- Get two-sided complex spectra: `TSCP = scipy.fft.fft(raw_trace);`
- Get one side of the spectra: `OSS = TSCP[:len(TSCP)//2 + 1];`
- Get real power spectra: `PSD = OSS * numpy.conjugate(OSS);`
- Get real amplitude `ASD = numpy.sqrt(PSD);`
- Get one sided frequencies of the spectra: `frequency = scipy.fft.fftfreq(len(TSCP), 1/f_samp)[:len(TSCP)//2+1];`
- Get phase of the spectra: `phase = numpy.arctan2(numpy.imag(TSCP[:len(TSCP)//2 + 1]),numpy.real(TSCP[:len(TSCP)//2 + 1]));`

- Calculate the spectra using Scipy's spectrogram function on `T` long segments overlapped by 50%: `frequency, time, specgram = scipy.signal.spectrogram(raw_trace, fs=f_samp, window='hann', nperseg=f_samp*T, noverlap=f_samp*T/2);`
    - `Frequency`: array of sample frequencies;
    - `Time`: Array of segment times (in seconds considering the first segment as the time 0);
    - `Specgram`: spectrogram of the data, contains the single segment spectra.
- Get real spectra using the Welch method on `T` long segments overlapped by 50%: `frequency, spectra = scipy.signal.welch(raw_trace, window='hann', nperseg=f_samp*T, noverlap=f_samp*T/2);`
    - `Frequency`: array of sample frequencies;
    - `spectra`: power spectrum of `raw_trace.`