Zaghloul_Habiba_Lab4

April 10, 2023

1 Cross-correlation (8 marks, 4/4)

It has been shown that noise records at two seismic stations, when cross-correlated and stacked, are very closely associated with the Green's function between these two seismic stations (i.e. given a delta force at one station, the displacement recorded at the other station). On Quercus, two files are given: one is a vertical component seismogram at PHL (Park Hill) and the other is a vertical component seismogram at MLAC (Mammoth Lakes). Both are records for one day (24 hours) on February 1, 2003. Sampling rate is $dt = 1.0$ seconds.

1. Take the cross-correlation (using FFT) of these seismograms using the record at PHL as $x(t)$ and that at MLAC as $y(t)$ and plot $C_{xy}(\tau)$. Zoom in your plot between [-300; 300] seconds.
2. Bit conversion is often used for the cross-correlation Greens function approach. It simply changes any positive numbers to 1 and negative numbers to -1 (numpy `sign()` function). Apply this to the data at PHL and MLAC and compute their cross-correlation. Compare the results against those from the previous step. Does bit-conversion work in this case to preserve the phase information of the true cross-correlation? (Note the amplitude of bit-converted cross-correlation is of no meaning).

   Hint: for discrete cross-correlation using fft:

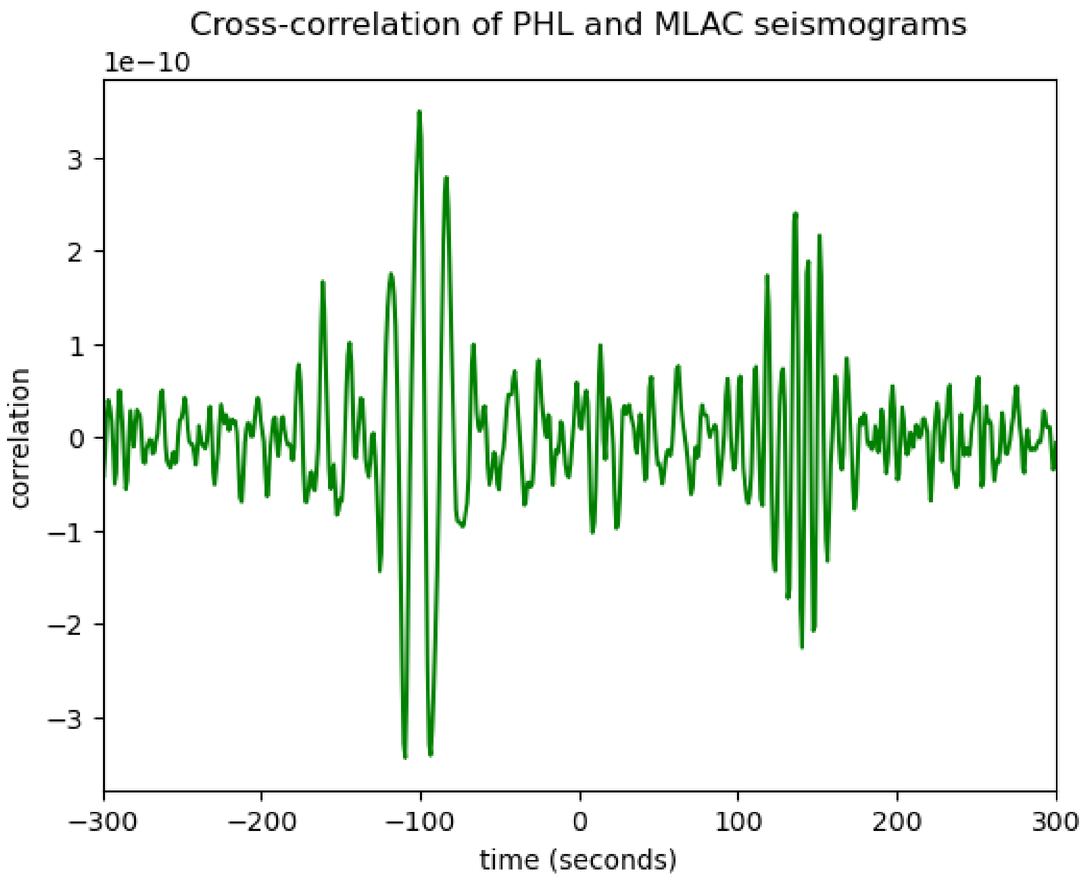
   - suppose `x = [x0 x1 ... x(N-1)]; y = [y0 y1 ... y(N-1)]`
   - note conv(x,y)=v=[v0 v1 … v(2N-2)], and notice the relationship between convolution and cross-correlation
   - also note how the fourier transform of the cross-correlation is related to $X(w)^*Y(w)$
   - if we first pad the end of each x and y array with N-1 zeros, convolution theorem suggests that we can interpret the inverse transform of $X^*(w)Y(w)$ as the result of a convolution (length 2N-1), w, interpreted as:
     w = ifft(…) = [w[lag = 0] w[lag = 1 dt] … w[lag = (N-1) dt] w[lag = -(N-1) dt] … w[lag = -1 dt]]
   - we then must apply fftshift to center the time shift/lag axis at 0
   - the point of this fft approach is that it's much faster than directly convolving; you can check against np.correlate(x, y, mode='same')
   - "zoom your plot to [-300, 300]" is in reference to the time lag axis in the above process

   Hint: you can load the ascii file into Python by `genfromtxt` function:
   tmp = np.genfromtxt('MLAC_data.txt')
   mlac = tmp.flatten()

[2]:
```python
#collaborators: none
#part 1
import numpy as np
import matplotlib.pyplot as plt
mlac = np.genfromtxt('MLAC_data.txt').flatten()
phl = np.genfromtxt('PHL_data.txt').flatten()

mlac_fft = np.fft.fft(mlac)
phl_fft = np.conj(np.fft.fft(phl))
corr = np.fft.fftshift(np.fft.ifft(phl_fft*mlac_fft))
N = len(corr)
tau = np.linspace(-N/2,N/2,N)

plt.plot(tau,corr, 'green')
plt.title('Cross-correlation of PHL and MLAC seismograms')
plt.xlabel('time (seconds)')
plt.ylabel('correlation')
plt.xlim(-300,300)
```

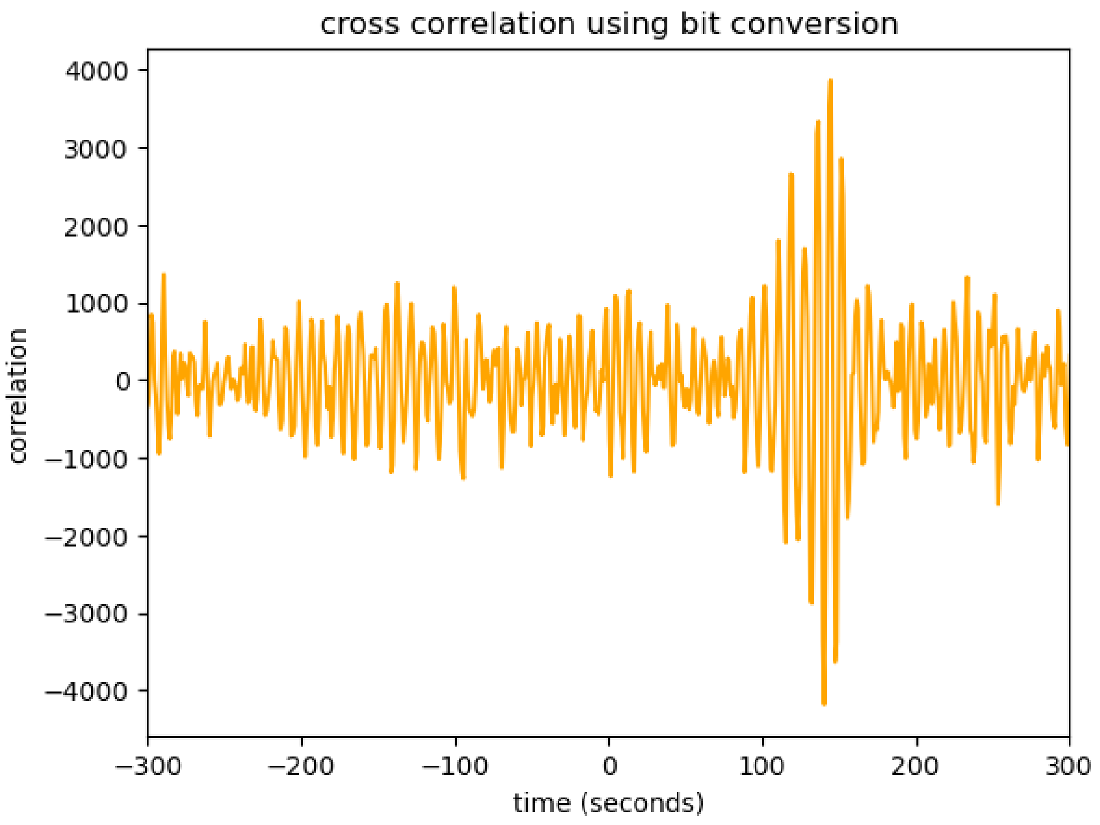
```
/opt/conda/lib/python3.10/site-packages/matplotlib/cbook/__init__.py:1369:
ComplexWarning: Casting complex values to real discards the imaginary part
  return np.asarray(x, float)
```

[2]:
```
(-300.0, 300.0)
```



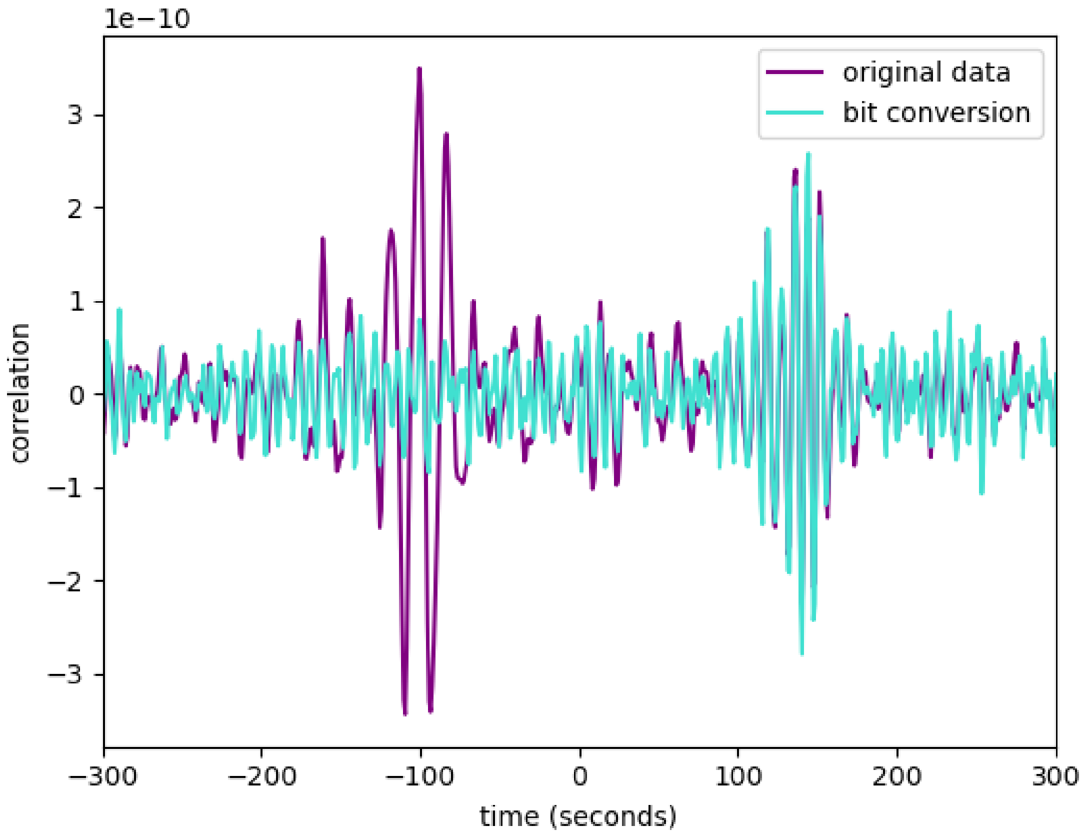
[3]:
```
#part 2
phl_bit = np.sign(phl)
mlac_bit = np.sign(mlac)
phlbit_fft = np.conj(np.fft.fft(phl_bit))
mlacbit_fft = np.fft.fft(mlac_bit)

corr_bit = np.fft.fftshift(np.fft.ifft(phlbit_fft*mlacbit_fft))
plt.plot(tau,corr_bit, 'orange')
plt.xlim(-300,300)
plt.xlabel('time (seconds)')
plt.ylabel('correlation')
plt.title('cross correlation using bit conversion')
```

[3]:
```
Text(0.5, 1.0, 'cross correlation using bit conversion')
```



```
[4]: #comparing part 1 with part 2
     plt.plot(tau,corr, 'purple', label = 'original data')
     plt.plot(tau,corr_bit / 15e12, 'turquoise', label = 'bit conversion')
     plt.xlim(-300,300)
     plt.legend()
     plt.xlabel('time (seconds)')
     plt.ylabel('correlation')
```

```
[4]: Text(0, 0.5, 'correlation')
```



The bit conversion preserves the phase information of the true cross correlation almost all the time, since the amplitude of the bit conversion doesn't give us any useful information. In the beginning (of our plot at -300s), the peaks do not align as well as the peaks after approximately 0 seconds. From 0-300s we can confidently say that the bit conversion fully preserves the phase information.

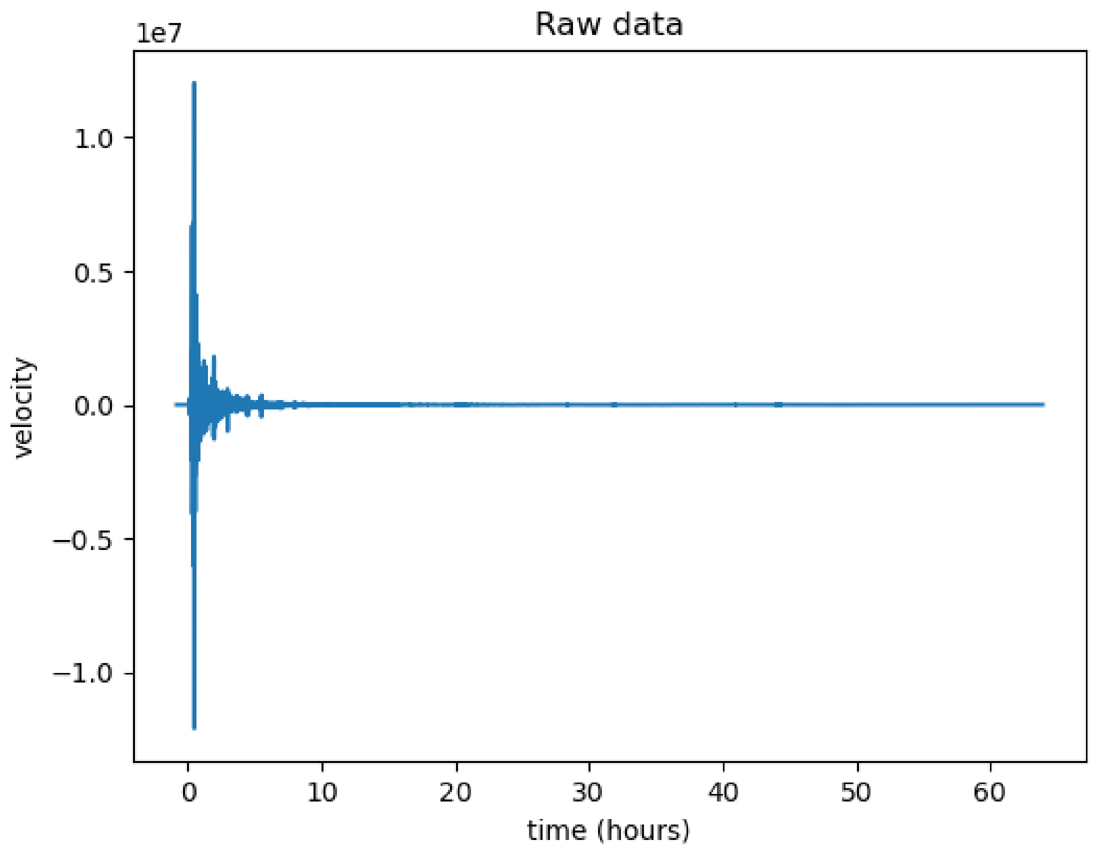
2 Normal Modes from Long Period Seismometer (7 marks, 1/2/2/1/1)

Background: The solid earth 'rings' like a bell after being struck by a great earthquake. These are the normal modes associated with internal structures (density and elastic moduli) of the Earth, and the excitation amplitudes of these modes are determined by the earthquake source mechanism. The frequencies of these normal modes of oscillation are very low, usually between 0.1 milliHertz (mHz) and 10 mHz. It is hard to see them above 10 mHz because these higher frequency modes attenuate quickly, or the frequency spacings are too small to be identified individually. Because the Earth is a complex structure, with twisting, breathing, and more complex spatial structure in its modes, the modal frequencies are not simple multiples of a single fundamental as is the case for a guitar string. They are labelled with a notation (e.g. like ${}_0S_2$ for spheroidal modes or like ${}_1T_8$ for toroidal modes) based on the spherical harmonic spatial distribution that the mode corresponds to, in the same way that the electron wavefunctions for the Hydrogen atom are labelled. Geophysicists measure these frequencies because they can be used to invert for models of the Earth's internal seismic velocity and density structures. With very high-resolution data, one can even see splitting of these resonances due to the Earth's rotation and elliptical shape, in a matter analogous to the Zeeman splitting of atomic spectral lines. You can also optically detect similar phenomenon ('helioseismology') going on in the sun, from which one can also test models of the sun's interior. (More descriptions can be found on any introductory solid-earth geophysics book). Here we examine three days of very long period seismic data recorded on the horizontal direction at station NWAO (Narrogin, Western Australia) after the devastating $M_w = 9.0$, Mar 11th, 2011 Honshu, Japan earthquake. Data `nwao.vh1` (VH channel, $dt = 10$ sec) is given as an ascii file with two columns: time and velocity (in counts from the digitizer).

1. Plot the raw data with a time axis in hours.
2. Plot the power spectrum of your raw data as a function of frequency (in mHz) **without** any windowing.
3. Plot the power spectrum of your raw data after
   - removing the minor linear trend in the same way as Lab 3, and subsequently
   - applying a hanning window $w_n = 1 - \cos(2\pi\frac{n}{N}); 0 \leq n \leq N$ (where N is the length of the data file)
4. Plot on top of each other the power spectra from 2 and 3 between [0.1, 2.6] mHz, and comment on the difference.
5. Using plt.annotate(…), identify any normal modes you can see. Use the provided modes.pdf (Table 1) to help guide your identification.

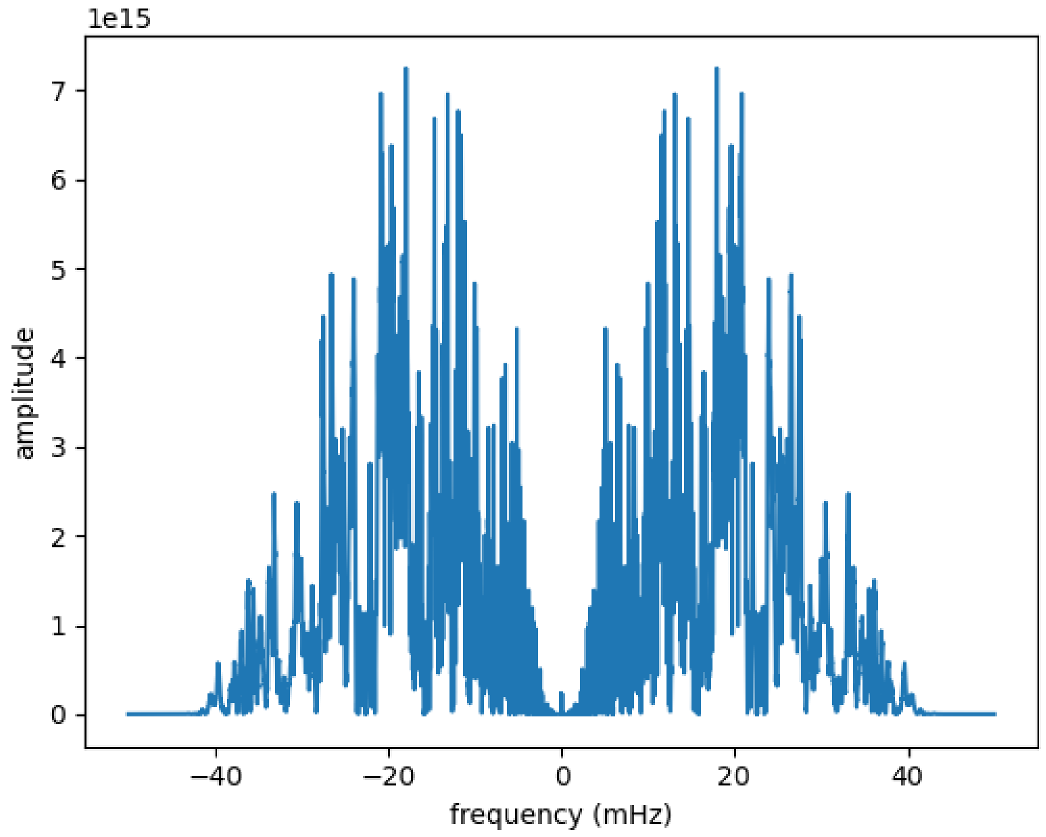
[5]:
```
#collaborators: none
#part 1

nwao = np.genfromtxt('nwao.vh1')
time = nwao[:,0]
velocity = nwao[:,1]
dt = 10 #seconds
hours = time/3600

plt.plot(hours,velocity)
plt.title('Raw data')
plt.xlabel('time (hours)')
plt.ylabel('velocity')
```

[5]:
```
Text(0, 0.5, 'velocity')
```



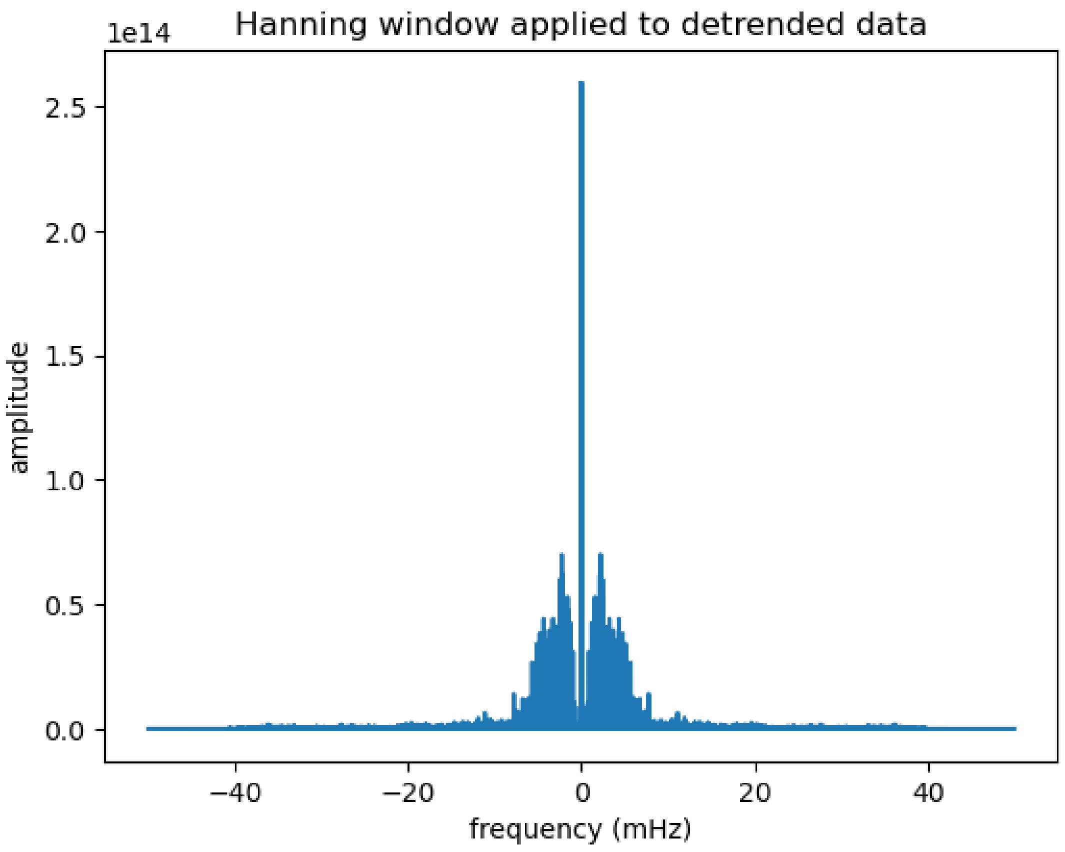
```
[6]: #part 2
     vel_fft = np.fft.fftshift(np.fft.fft(velocity))
     vel_freq = np.fft.fftshift(np.fft.fftfreq(len(time),dt)) *1000 #to make it mHz
     power = vel_fft*np.conj(vel_fft)
     plt.plot(vel_freq,power)
     plt.ylabel('amplitude')
     plt.xlabel('frequency (mHz)')
```

```
[6]: Text(0.5, 0, 'frequency (mHz)')
```



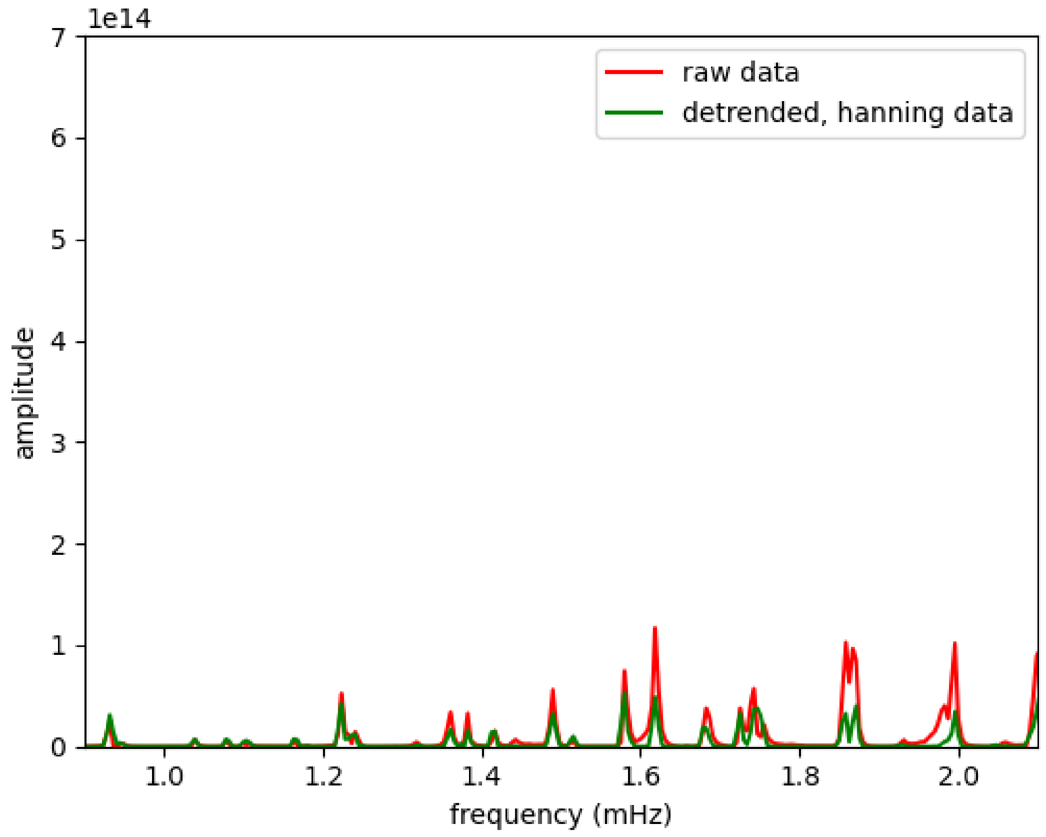
```
[7]: #part 3
     p = np.polyfit(time,velocity,1)
     trend = np.polyval(p,time)
     vel_detrend = velocity - trend
     N = len(velocity)
     n = np.arange(N)
     w = 1 - np.cos(2*np.pi*n/N)

     vel_han = vel_detrend * w
     hanfft = np.fft.fftshift(np.fft.fft(vel_han))
     hanpower = np.conj(hanfft)*hanfft

     plt.plot(vel_freq, hanpower)
     plt.title('Hanning window applied to detrended data')
     plt.xlabel('frequency (mHz)')
     plt.ylabel('amplitude')
```

```
[7]: Text(0, 0.5, 'amplitude')
```



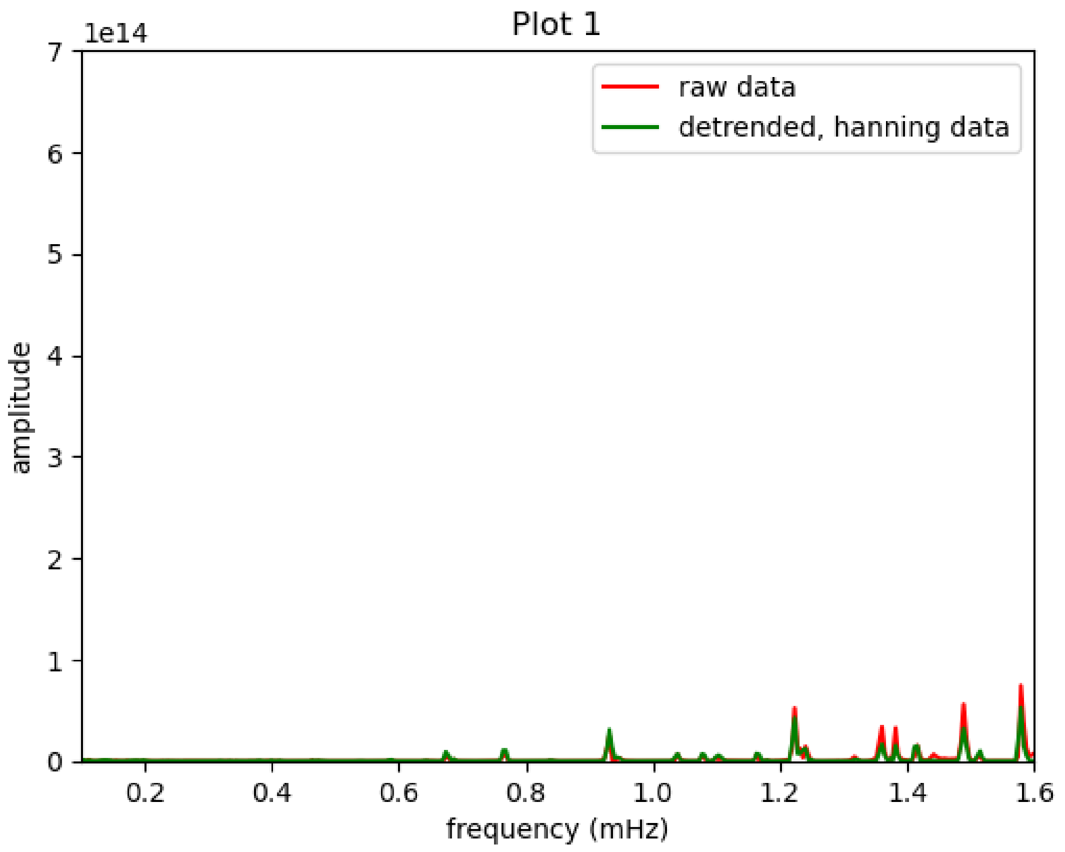
```
[16]: #part 4
      plt.plot(vel_freq,power, 'red', label = 'raw data')
      plt.plot(vel_freq,hanpower,'green', label = 'detrended, hanning data')
      plt.xlim(0.1,2.6)
      plt.xlabel('frequency (mHz)')
      plt.ylabel('amplitude')
      plt.legend()
      plt.ylim(0,7e14)
```

```
[16]: (0.0, 700000000000000.0)
```



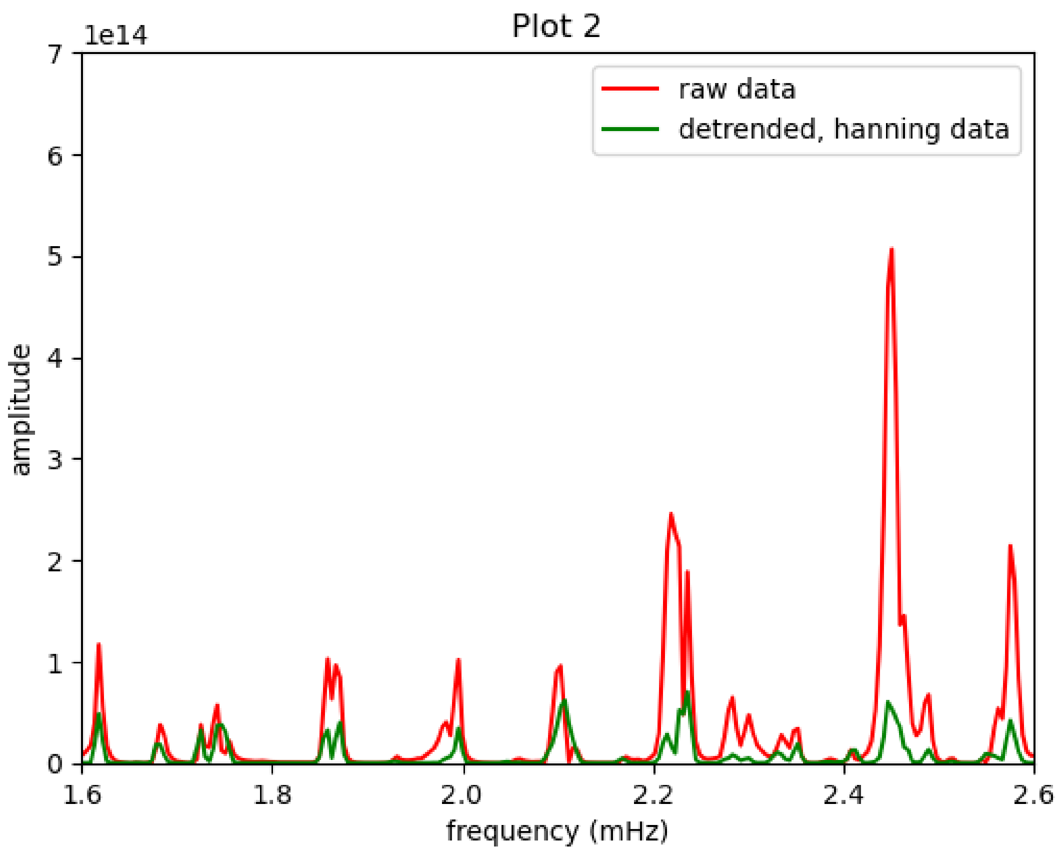
```
[21]: plt.plot(vel_freq,power, 'red', label = 'raw data')
      plt.plot(vel_freq,hanpower,'green', label = 'detrended, hanning data')
      plt.xlim(0.1,1.6)
      plt.xlabel('frequency (mHz)')
      plt.ylabel('amplitude')
      plt.title('Plot 1')
      plt.legend()
      plt.ylim(0,7e14)
```

```
[21]: (0.0, 700000000000000.0)
```



```
[20]: plt.plot(vel_freq,power, 'red', label = 'raw data')
      plt.plot(vel_freq,hanpower,'green', label = 'detrended, hanning data')
      plt.xlim(1.6,2.6)
      plt.xlabel('frequency (mHz)')
      plt.ylabel('amplitude')
      plt.title('Plot 2')
      plt.legend()
      plt.ylim(0,7e14)
```

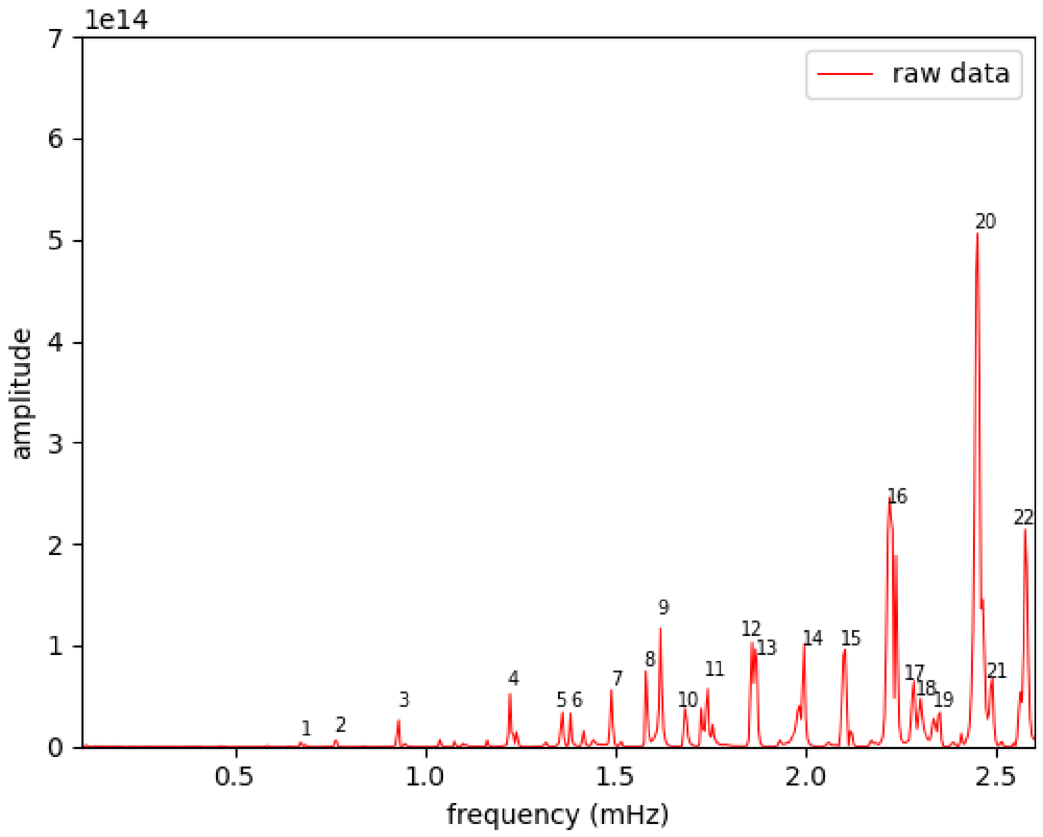
```
[20]: (0.0, 700000000000000.0)
```



To outline the difference between the two, I split the graph into two plots, plot 1 and 2 above. We can see from frequencies 0.1-1.6 mHz the original and detrended data peaks are aligned perfectly but once the frequency gets larger than 1.6mHz, the detrended data peaks slightly shifts away from the raw data. However, the number of peaks remain consistent even after detrending and applying the hanning window.

[12]:
```
#part 5
plt.plot(vel_freq,power, 'red',lw = 0.7, label = 'raw data')
plt.xlim(0.1,2.6)
plt.xlabel('frequency (mHz)')
plt.ylabel('amplitude')
plt.legend()
plt.ylim(0,7e14)
plt.annotate('1', xy = (0.673,1e13), fontsize = 7)
plt.annotate('2', xy = (0.76,1.5e13), fontsize = 7)
plt.annotate('3', xy=(0.925,4e13), fontsize = 7)
plt.annotate('4', xy=(1.215,6e13), fontsize = 7)
plt.annotate('5', xy=(1.34,4e13), fontsize = 7)
plt.annotate('6',xy=(1.38,3.9e13), fontsize = 7)
plt.annotate('7',xy=(1.485,6e13), fontsize = 7)
plt.annotate('8',xy=(1.575,8e13), fontsize = 7)
plt.annotate('9',xy=(1.61,1.3e14), fontsize = 7)
plt.annotate('10',xy=(1.66,4e13), fontsize = 7)
plt.annotate('11',xy=(1.73,7e13), fontsize = 7)
plt.annotate('12',xy=(1.83,1.1e14), fontsize = 7)
plt.annotate('13',xy=(1.87,0.9e14), fontsize = 7)
plt.annotate('14',xy=(1.99,1e14), fontsize = 7)
plt.annotate('15',xy=(2.09,10e13), fontsize = 7)
plt.annotate('16',xy=(2.21,2.4e14), fontsize = 7)
plt.annotate('17',xy=(2.257,6.6e13), fontsize = 7)
plt.annotate('18',xy=(2.285,5e13), fontsize = 7)
plt.annotate('19',xy=(2.33,4e13), fontsize = 7)
plt.annotate('20',xy=(2.44,5.1e14), fontsize = 7)
plt.annotate('21',xy=(2.47,6.7e13), fontsize = 7)
plt.annotate('22',xy=(2.54,2.2e14), fontsize = 7)
```

```
[12]: Text(2.54, 220000000000000.0, '22')
```


1e14
amplitude
raw data
frequency (mHz)