UiT The Arctic University of Norway

Faculty of Science and Technology

Programming Assignment 3

Gravitational waves and the LIGO signals

Trym Varland

FYS-2006, Autumn 2022

Task 1: Data

The sample rate for both of the signals is $f_s = 4096Hz$. The samples in both signals are synchronized in time, i.e., sample n in signal $x_H[n]$ and $x_L[n]$ occur at the same time.

a)

The code which would read the Hanford and Livingston signals from the data file, have i used the following code from the git `https://bit.ly/3bIcRlB` to read the data files.

b)

To find how many samples are in each of the signals, we can do this easily by using the `len(signal)` function in python. Which gives the following:

$$\texttt{len(Hanford)} = 131072 \text{ samples}$$
$$\texttt{len(Livingston)} = 131072 \text{ samples}$$

Which can also be found, by plotting the signals.

c)

We found in b) that the signals have $n = 131072$ samples, and we are given that the sample rate is $f_s = 4096Hz$. Than we can just divide the length of the signal by the samples.

$$t = \frac{\textit{Length of signal}}{\textit{Sample-rate}} = \frac{131072}{4096} = \underline{32.0s}$$

So we have 32.0 seconds of signals for both of the the data vectors, $x_H[n]$ and $x_L[n]$. This is also done in the code.

d)

Followed by the Nyquist-Shannon sampling theorem, the sample rate f_s has to be at least the difference between the smallest and biggest frequencies to avoid aliasing, which is defined as $f_s > f_{max} - f_{min}$. We have that the sample rate is $f_s = 4096Hz$. If compute the Nyquist sampling theorem, we have $4096Hz > 300Hz - (-300Hz) \Rightarrow 4096Hz > 600Hz$. As we can see $f_s = 4096$, is a good value for not breaking the Nyquist sampling theorem and to avoid aliasing.

e)

The sample spacing would be the inverse of the sample rate, and will yield the same for both of the signals. So we have.

$$T_s = \frac{1}{f_s} = \underline{\underline{2.44 \times 10^{-4}s}}$$

Task 2: Plotting the data

a)

In plot 1 we have the signals $x_H[n]$ and $x_L[n]$, with time in seconds on the horizontal axis and strain on the vertical axis.

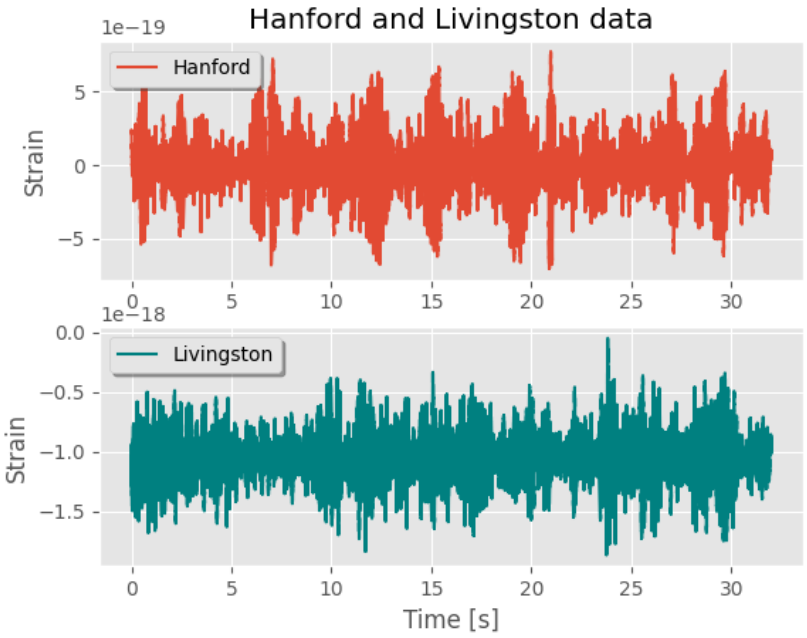

Figure 1: Plot of the Hanford and Livingston data, with time in seconds on the x-axis and strain on the y-axis.

b)

The minimum, maximum and the mean values of the signals, can easily be found by using `min()`, `max()` and `numpy.mean()` functions in python. Which gives the following values for the $x_H[n]$:

$$\text{Minimum value: } -7.045 \times 10^{-19}$$
$$\text{Maximum value: } 7.706 \times 10^{-19}$$
$$\text{Mean value: } 5.896 \times 10^{-23}$$

And for $x_L[n]$ we have:

$$\text{Minimum value: } -1.869 \times 10^{-18}$$
$$\text{Maximum value: } -4.600 \times 10^{-20}$$
$$\text{Mean value: } -1.052 \times 10^{-18}$$

Task 3: Selecting a tapered window function

a)

We are going to choose the tapered window function called the Hann window. A Hann window with length N is defined as.

$$w_H[n] = \begin{cases} sin^2(\pi n/N) & 0 \leq n \leq N-1 \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

b)

We are given the following signal:

$$x[n] = cos(2\pi f n/f_s)$$

With frequency $f = 31.5Hz$, sample frequency $f_s = 4096Hz$ and sample signal $n \in [0, 1, 2, \cdots, 4095]$. By plotting the signal $x[n]$ and the window signal $w[n]x[n]$, having the window signal the same length as the signal. We get the the following plot 2.

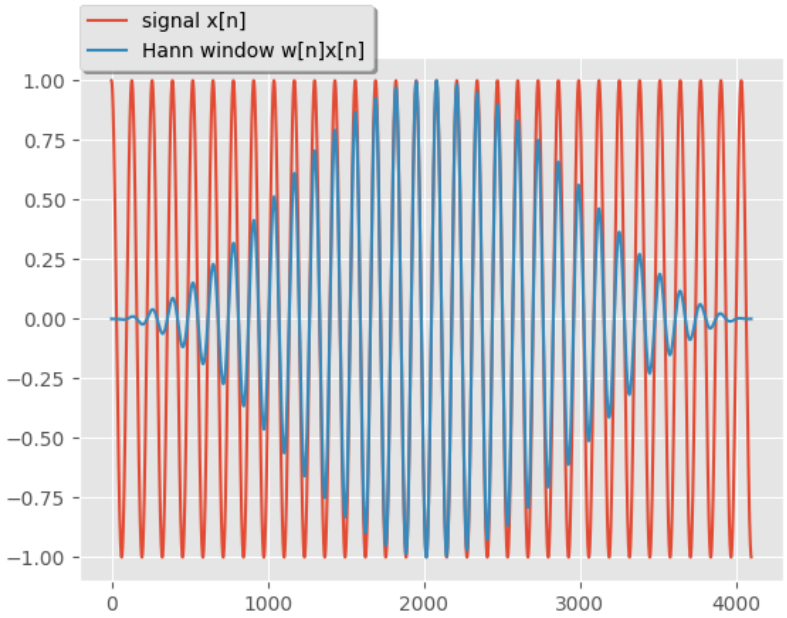

Figure 2: Plot of the signal $x[n]$ and window signal $w[n]x[n]$, where the Hann window function is used.

c)

The discrete Fourier frequency is defined as.

$$\hat{\omega}_k = \frac{2\pi}{N}k \tag{2}$$

Which gives radians per sample. Where k have integer values between 0 and N_1, that would say $k \in [0, 1, 2, \cdots, N-1]$, hence $\hat{\omega}_k \in [0, 2\pi]$. These values we shift, so that $\hat{\omega}_k \in [-\pi, \cdots, \pi]$ and covert dose values to frequency $-f_s/2 < f_k < f_s/2$ (principal spectrum). In the code we will do this by using the numpy function `np.fft.fftfreq` and `np.fft.fftshift`.
Which gives the following frequencies $f_k \in [-2048.0Hz, -2047.0Hz, \cdots,$ $2046.0Hz, 2047Hz]$, which would correspond to the principal spectrum.

d)

To find the values of k that correspond to a frequency f_k that is nearest to 31.5 and -31.5 hertz. I'm making a function that taking in f_k and the frequencies $31.5and -$ 31.5Hz, and will find the index in f_k that corresponding to those values. The index will be the k value.
The function gives that the corresponding value for 31.5Hz is $k = 31$Hz, and for -31.5Hz we have that $k = 4064$Hz.

e)

Here we have the power spectrum of the windowed signal $w[n]x[n]$ (with tapering) and the signal $x[n]$ (without tapering). Where we have the magnitude squared $10log_{10}(|\hat{x}[k]|^2)$ (decibel, power) on the y-axis, and the frequency in Hz on the x-axis. As the plot showed below 3.

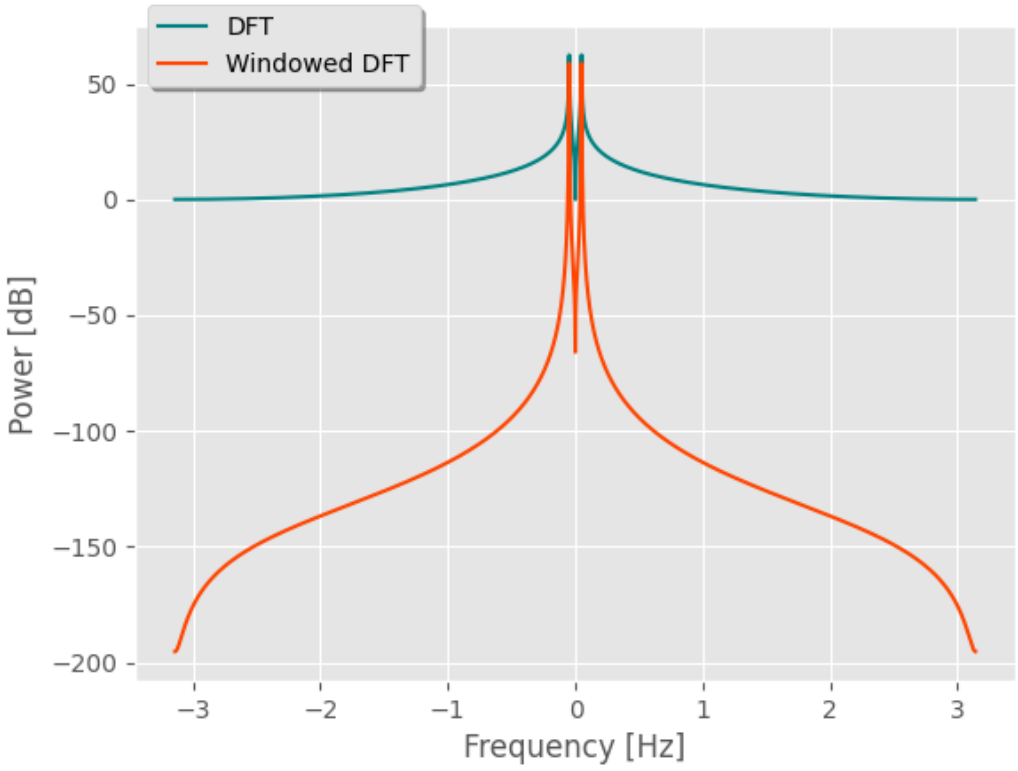

Figure 3: Power spectrum of the windowed signal $w[n]x[n]$ and the signal $x[n]$, with frequency (Hz) on the x-axis and power (dB) on the y-axis.

f)

Under we have a plot of the power spectrum, where $31.5Hz$ and $-31.5Hz$ is marked with black dashed line.

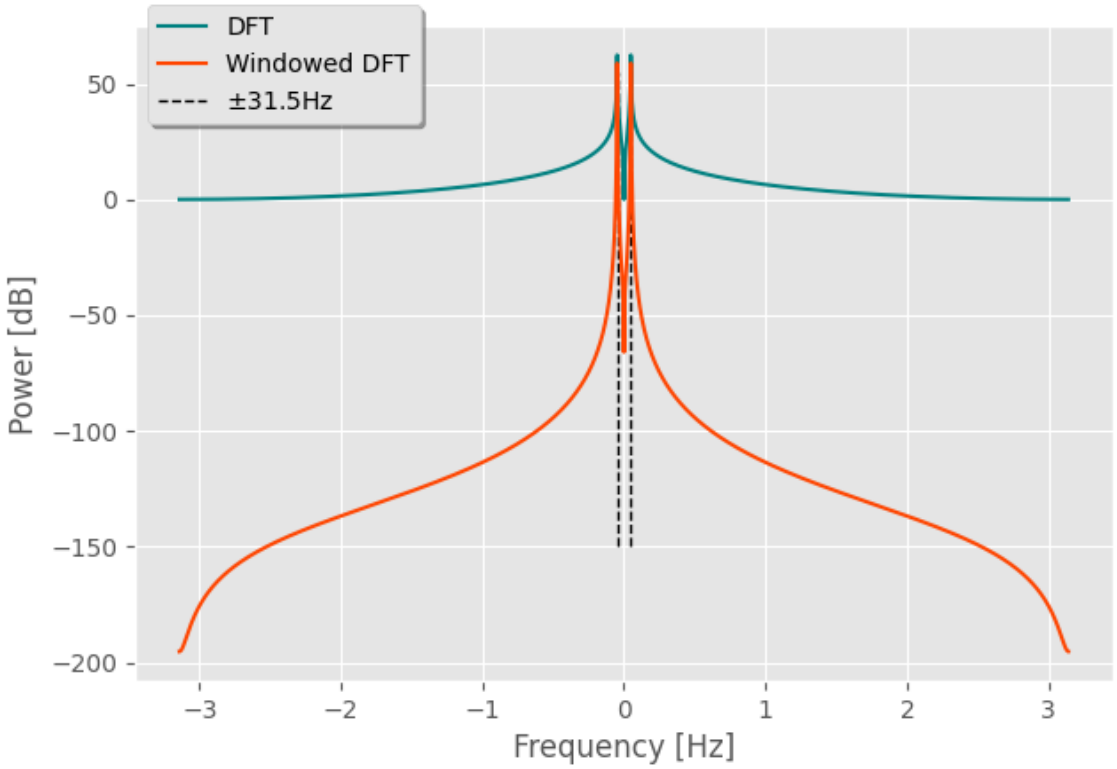

Figure 4: Power spectrum of the windowed signal $w[n]x[n]$ and the signal $x[n]$, $31.5Hz$ and $-31.5Hz$ is marked with black dashed line.

g)

If we zoom in on the plot of the power spectrum 5, we see that there is zero frequency component at the frequencies $-31.5Hz$ and $31.5Hz$. The zero frequency component at these frequencies comes from spectral leakage, which occurs when have the frequency between two discreet values, and gives a maximum amplitude error of some percent.

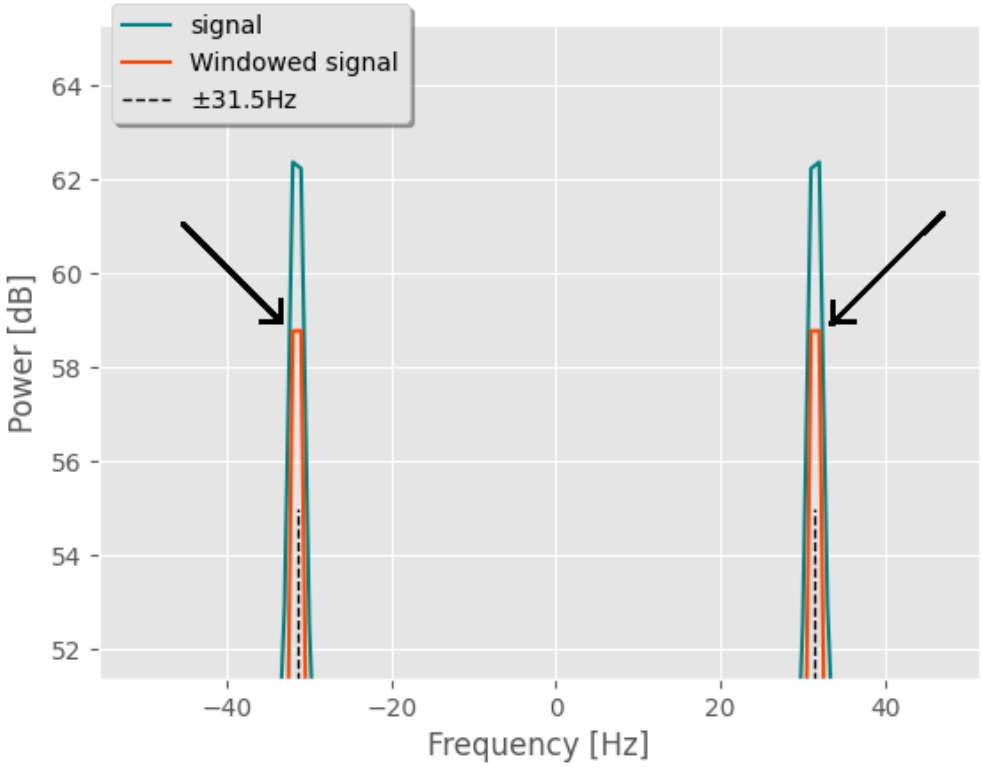

Figure 5: zoomed in plot of power spectrum, with arrows pointing at where we have zero frequency component.

h)

In Figure 6 we have zoomed on the ends of plot 4, where we can see that DFT ($10log_{10}(|\hat{x}[k]|^2)$ have 0dB but the windowed DFT ($10log_{10}(|\hat{x}_w[k]|^2)$ have ≈ -195dB. So the windowed function, have a significantly better frequency response.

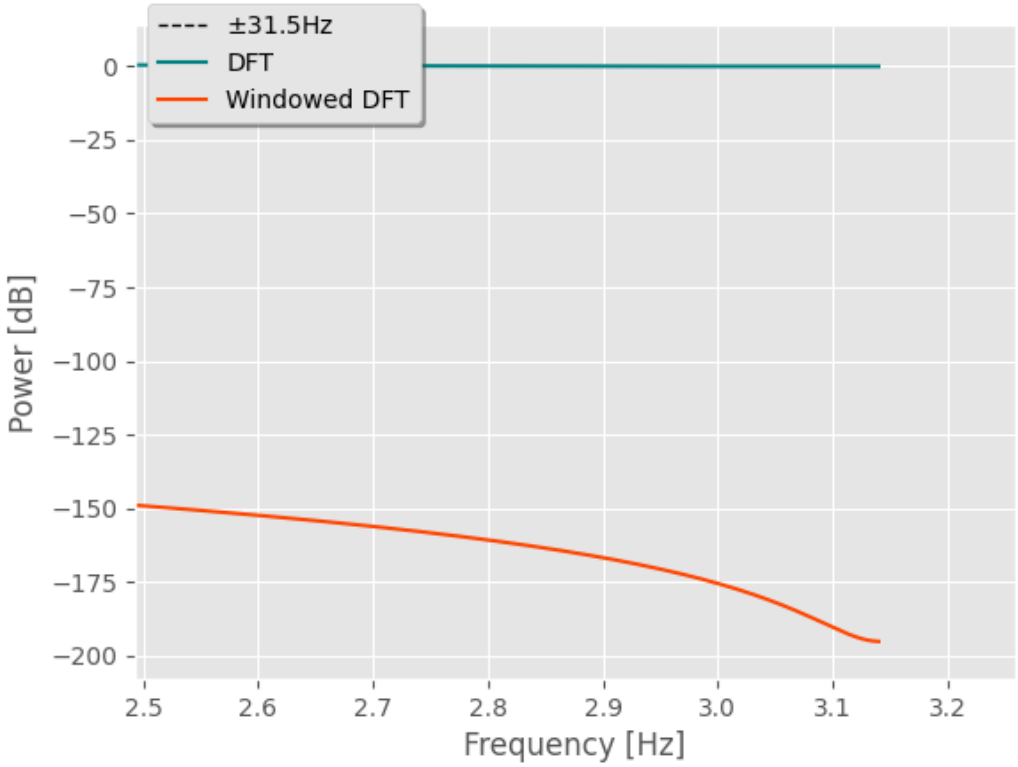

Figure 6: Zoomed in part of the Figure 4, to show how much greater the frequency response is for windowed DFT.

Task 4: Estimating the spectrum of the LIGO signal

a)

Now we gone apply what we did in task 3, to calculate the power spectrum of the LIGO signals $|\hat{x}_H[k]|^2$ and $|\hat{x}_L[k]|^2$. Here $\hat{x}_H[k]$ is the windowed DFT of the Hanford signal $x_H[n]$, and the same yields for the Livingston signal $x_L[n]$. We obtain the windowed DFT, by using.

$$\hat{x}[k] = \sum_{n=0}^{N-1} w[n]x[n]e^{-i\frac{2\pi}{N}kn} \tag{3}$$

And we will continue using the Hann window function 1.

b)

Under we have a plot of the power spectrum to the LIGO signals, with only positive frequencies. With frequency (Hz) on the horizontal axis, and power with a decibel scale on the vertical axis.

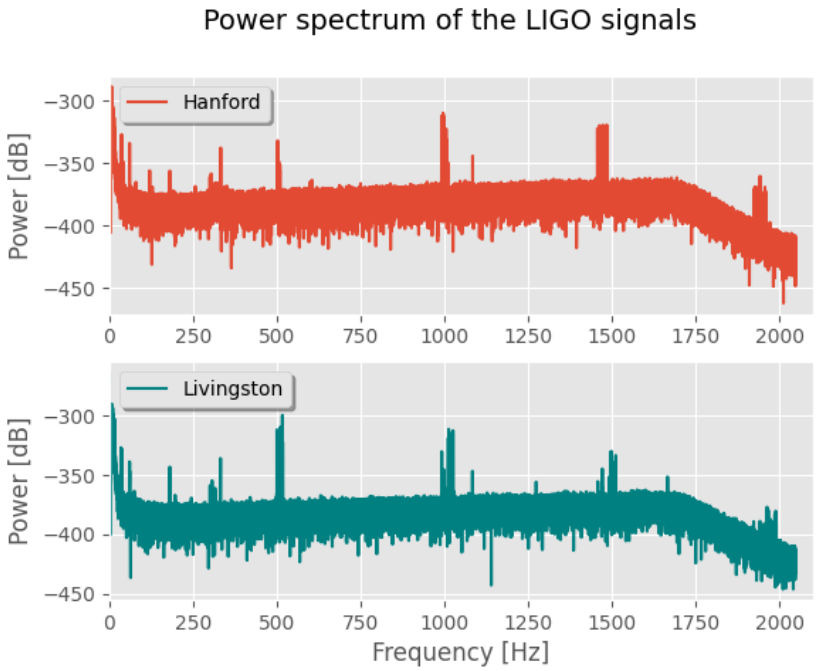

Figure 7: Power spectrum of the LIGO signals, with only positive frequencies.

c)

In figure 8 we have identified a number of frequency bands, which contain narrow band interference inside the boxes on the power spectrum of the LIGO signals. If we zoom in to the strong spectral components in the power spectrum, we see that we have a lot of strong spectral components therefor i take them as one.

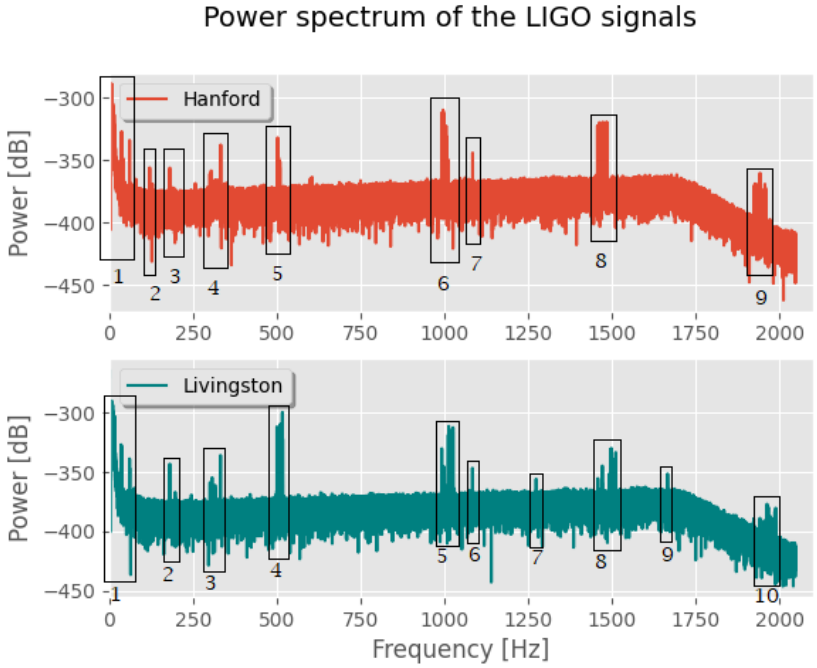

Figure 8: Power spectrum plot of the LIGO signals, where the frequency bands are marked with boxes.

Task 5: Whitening filter

In order to remove instrumental noise, you will next implement a whitening filter and apply it to the LIGO signal.
A whitening filter can be implemented as an FIR filter, which in frequency domain can be implemented as a multiplication:

$$\hat{y}[k] = \hat{h}[k]\hat{x}[k] \tag{4}$$

Here $\hat{y}[k]$ is the DFT of the output of the filter, $\hat{h}[k]$ is the DFT of the whitening filter, and $\hat{x}[k]$ is the windowed DFT of the input signal $x[n]$. The DFT of the whitening filter $\hat{h}[k]$, is expressed as.

$$\hat{h}[k] = \frac{1}{|\hat{x}[k]|} \tag{5}$$

a)

To find that $|\hat{y}[k]| = 1$, can imply the whitening filter 5 to the frequency domain 4, which gives.

$$\hat{y}[k] = \frac{1}{|\hat{x}[k]|} \cdot \hat{x}[k] = \pm 1$$

Hence:

$$|\hat{y}[k]| = |\pm 1| = 1$$

b)

When implementing a whitening filter $\hat{h}[k]$ in frequency domain for the LIGO data, i going to make a function in python that takes on the strain data for each of the LIGO data. This is easily done, by implementing the whitening filter 5.

c)

Now we use a inverse discrete Fourier transform to transform the whitened signal $\hat{y}[k]$ into the time domain, this is done the same way as we did in b) but now we inverse discrete Fourier transform the equation 4.

d)

Under we have a plot of the whitened signals of Hanford $y_H[n]$ and Livingston $y_L[n]$, where we have used time in seconds ($t \in [0, t_{max})$ on the horizontal axis and whitened strain $y[n]$ on the vertical axis. At the bottom we have a zoomed in comparison, where the gravitational wave signal should be. But it is really noisy, so it still hard to see it. It can look like it is laying between $16.40s$ and $16.45s$.

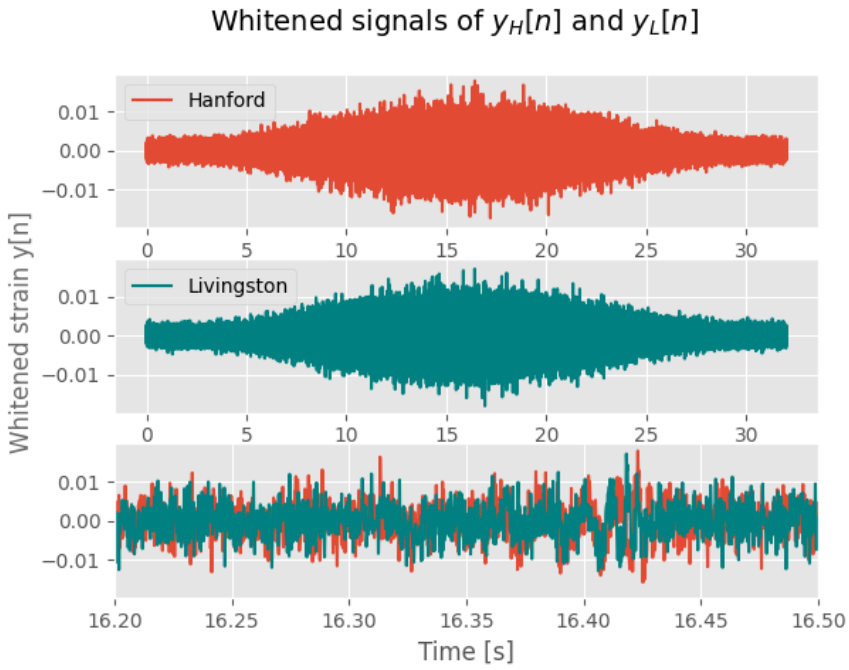

Figure 9: Plot of the the whitened signals of Hanford $y_H[n]$ and Livingston $y_L[n]$, with a zoomed in comparison at the bottom.

Task 6: Low-pass filtering

The gravitational wave signal is at frequencies below 300 Hz. We are going to design a simple running mean averaging low-pass filter

$$y[n] = \frac{1}{L}\sum_{k=0}^{L-1} x[n-k] \tag{6}$$

that will attenuate spectral components with frequencies above 300 Hz.

a)

We going to find the integer value of L such that the filter will reduce the power frequency components at $f = 300Hz$ by approximately $-6dB$, compared to to the filter output for a $f = 0Hz$ signal.
The easiest way to do this is by doing b), and than check for which values of L the filter will reduce the power frequency components at $f = 300Hz$ by approximately $-6dB$.
After Varying the integer value of L, we found out that $L = 8$ is the best value. Which reduced the power to $\approx -5.5db$.

b)

In figure 10 we have plot of the power spectral response of the filter in dB scale ($10log_{10}|\mathcal{H}|^2$). Where we have frequency (Hz) on the horizontal axis, and dB scale on the vertical axis. The gray lines indicate where $f = 300$ and $-6dB$ is, and the

orange dot show where $-6dB$ is on the power spectral response of the filter. Which verify that the $-6dB$ point is close to $f = 300Hz$.

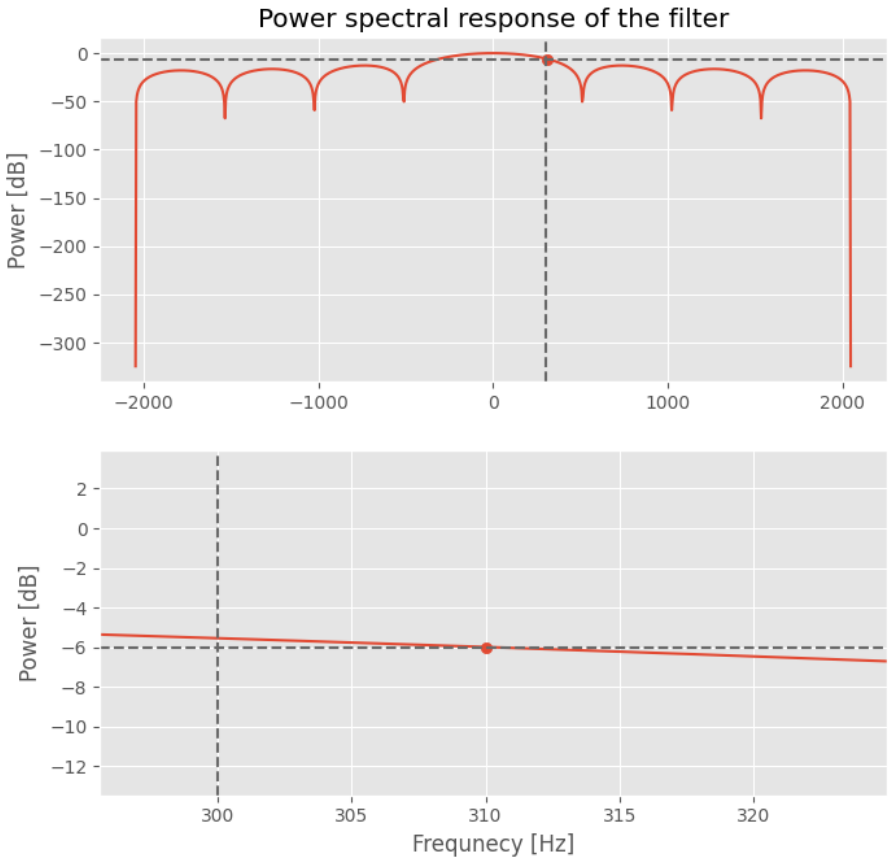

Figure 10: Plot of the power spectral response of the filter with a dB scale on the vertical axis and frequency on the horizontal axis, is also shows control line for $-6db$ and where $f = 300Hz$.

c)

We have that the $\mathcal{H}(\hat{\omega})$ is defined as

$$\mathcal{H}(\hat{\omega}) = \sum_{k=-\infty}^{\infty} h[k]e^{-i\hat{\omega}k} \tag{7}$$

By using eq 530 from the syllabus and that $h[n] = \frac{1}{L}\sum_{k=0}^{L-1}\delta[n-k]$, we now have.

$$\mathcal{H}(\hat{\omega}) = \sum_{k=0}^{L-1} \frac{1}{L} e^{-i\omega k} \tag{8}$$

Than we define $c = 1/L$, $\alpha = exp(-i\hat{\omega})$ and $s = \mathcal{H}(\hat{\omega})$, we can find a closed form solution.

$$s = c + c\alpha + c\alpha^2 + \cdots + c\alpha^{L-1} \tag{9}$$
$$\alpha s = c\alpha + c\alpha^2 + c\alpha^3 + \cdots + c\alpha^{L} \tag{10}$$
$$s - \alpha s = c - c\alpha^{L} \tag{11}$$
$$s(1-\alpha) = c(1-\alpha^{L}) \tag{12}$$
$$s = c\frac{(1-\alpha^{L})}{1-\alpha} \tag{13}$$

Forward we substituting back the values c, α and s.

$$\mathcal{H}(\hat{\omega}) = \frac{1}{L}\frac{1-e^{-i\hat{\omega}L}}{1-e^{-i\omega}} \tag{14}$$

We can than use a trick of multiplying the complex exponentials with $1 = e^{i\omega/2}e^{-i\omega/2}$.

$$\mathcal{H}(\hat{\omega}) = \frac{1}{L}\frac{1-e^{-i\hat{\omega}L}}{1-e^{-i\hat{\omega}}}\Big| \cdot \frac{e^{i\hat{\omega}L/2}e^{-i\hat{\omega}L/2}}{e^{i\hat{\omega}/2}e^{-i\hat{\omega}/2}} \tag{15}$$
$$= \frac{1}{L}\frac{e^{-i\hat{\omega}L/2}(e^{i\hat{\omega}L/2} - e^{-i\hat{\omega}L/2})}{e^{-i\hat{\omega}/2}(e^{i\hat{\omega}/2} - e^{-i\hat{\omega}/2})} \tag{16}$$

We can than use the $sin(\theta) = \frac{1}{2i}(e^{i\theta} - e^{-i\theta})$, to relate the symmetric pairs to a sinusoidal functions.

$$\mathcal{H}(\hat{\omega}) = \frac{1}{L}\frac{\cancel{2i} \cdot e^{-i\hat{\omega}L/2}sin(\frac{\hat{\omega}L}{2})}{\cancel{2i} \cdot e^{-i\hat{\omega}/2}sin(\frac{\hat{\omega}}{2})} \tag{17}$$
$$= \frac{1}{L}\frac{sin(\frac{\hat{\omega}L}{2})}{sin(\frac{\hat{\omega}}{2})}e^{-i\hat{\omega}L/2+i\hat{\omega}/2} \tag{18}$$
$$= \frac{1}{L}\frac{sin(\frac{\hat{\omega}L}{2})}{sin(\frac{\hat{\omega}}{2})}e^{-i\hat{\omega}(L-1/2)} \tag{19}$$

So the time delay τ introduced by the filter, is $\tau = \frac{L}{2} - \frac{1}{2} = \frac{8-1}{2} = 3.5$ samples. Than to convert the time delay to seconds we divide τ by the sample space ($\frac{1}{f_s}$), $\frac{3.5\text{samples}}{4096Hz} = \underline{\underline{8.5 \times 10^{-}4s}}$

d)

Going to apply the running mean average low-pass filter on the whitened Hanford ($y_H[n]$) and Livingston signals ($y_L[H]$), using a function in numpy called `np.convolve` and `np.repeat`. Which we are using to calculate the running mean averaging low-pass filter (eq.6).

e)

Undoing the effects of the filer time-delay by shifting the signal in time. Which we did by adjusting the time variable $t' = t - \tau$, where τ is the time delay which we found in c).

f)

In plot 11 we have the low-pass filtered whitened Hanford and Livingston signals, with time in seconds on the horizontal axis and strain on the vertical axis. As we can see the plot, is pretty similar to **Figure 193** from task paper.

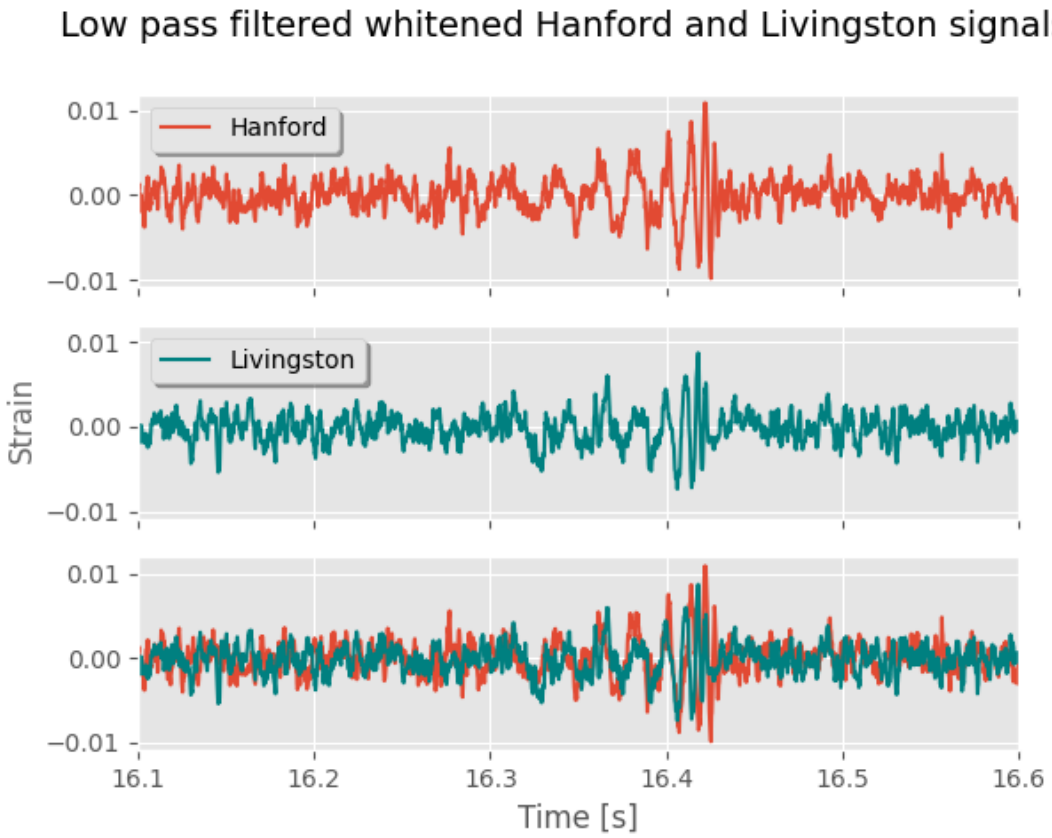

Figure 11: Low pass filtered whitened LIGO signals, with time in seconds on the horizontal axis and signal amplitude (strain) on the vertical axis. At the bottom is a comparison.

In plot 12 we see what happens to the whitened LIGO signal, when we run it through the running mean average low-pass filter. Which made the signal much clearer and not as much noise, we are able to see the gravitational wave clearly laying between $16.3s$ and $16.45s$.

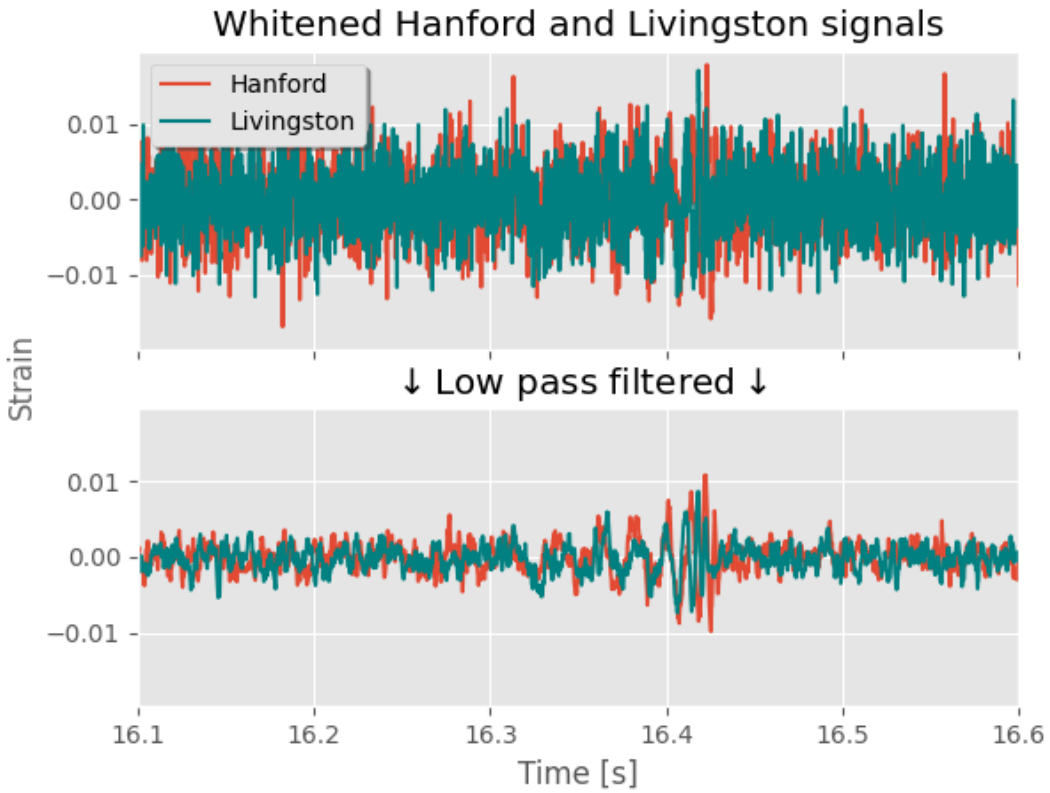

Figure 12: A plot visualization to see the big impact the running average low-pass filter have.

Task 7: Time delay

Gravitational waves are expected to propagate at the speed of light. The Hanford and Livingston detectors are separated by about 3000km. A gravitational wave will propagate this distance in about 10ms.
The relative time delay between the signal detected at Hanford and Livingston is expected to be $-10 < \tau < 10$ms, if it is moving at the speed of light in vacuum.

a)

To determine the time separation between the two signals, i making a algorithm that find the highest peak of the signals, than finding the corresponding time value. Than subtracting the these two time values, and adding this time delay to the original time. As we can see in Figure 12, the Hanford is laying in advance of the Livingston signal, so the **"new time"** will be added to the Livingston signal. This gives the plot in figure 13.

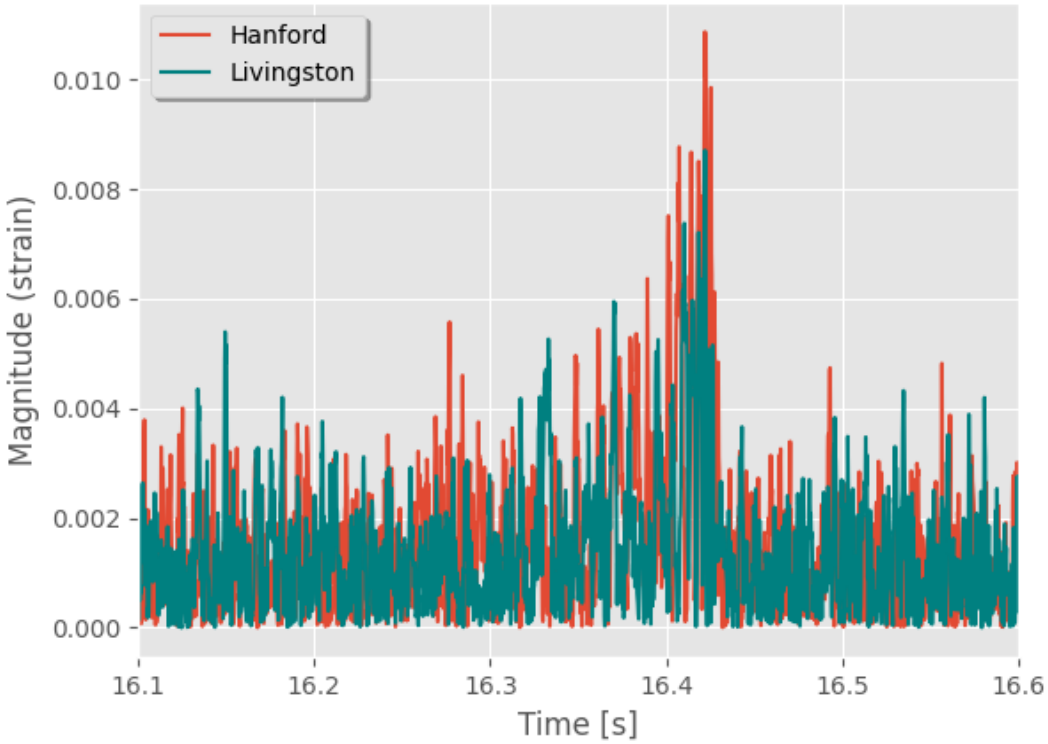

Figure 13: Plot of the *magnitudes* of the filtered gravitational wave signals ($|y_L[n]|$ and $|y_H[n+n_0]|$)

As we can see the signals aligned in time, by using this algorithm described above. The magnitude of the signal is used to avoid phase-time ambiguities.

b)

We found a) that the Livingston signal is $\tau = 4.0 \times 10^{-3}s$, behind the Hanford signal. The sample delay $n_0 = 4096\frac{\text{sample}}{s} \cdot 4 \times 10^{-3}s = 16$ samples, which will say that the Hanford signal comes 16 samples after the Livingston signal.

c)

As mention in b), the sample delay n_0 correspond to $\tau = 4.0 \times 10^{-3}s$.

d)

The time delay τ as we can see, is in agreement with the gravitational-wave propagation speed $-10 < 4 < 10$ms.

Task 8: Dynamic spectrum

a)

To calculate the dynamic spectrum of the low-pass filtered whitened signal $|\hat{x}[t,k]|$, this gone be implemented by using scipy that have a function called `spectrogram`.

b)

In Figure 14 we have dynamic power spectrum $|\hat{x}[t,k]|^2$ plot of the gravitation wave Hanford signal, in a dB scale. Here we can clearly see the time-frequency response behavior of the gravitational wave signal, which is the part of the plot highest dB.

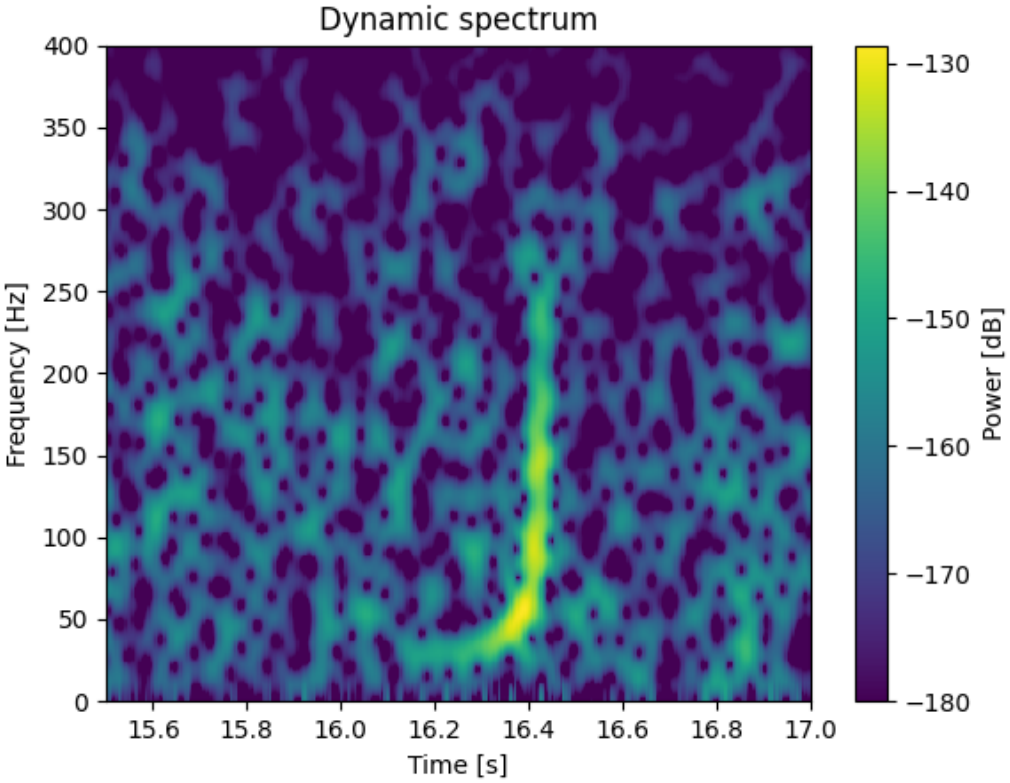

Figure 14: Dynamic power spectrum plot of the gravitation wave Hanford signal, with dB scale.

Figure 15 shows both the Hanford and Livingston signal in dynamic power spectrum plot, of the gravitation wave. In the bottom there is a comparison between the two signal, where the Livingston signal is transparent and laid on the top of the Hanford signal.

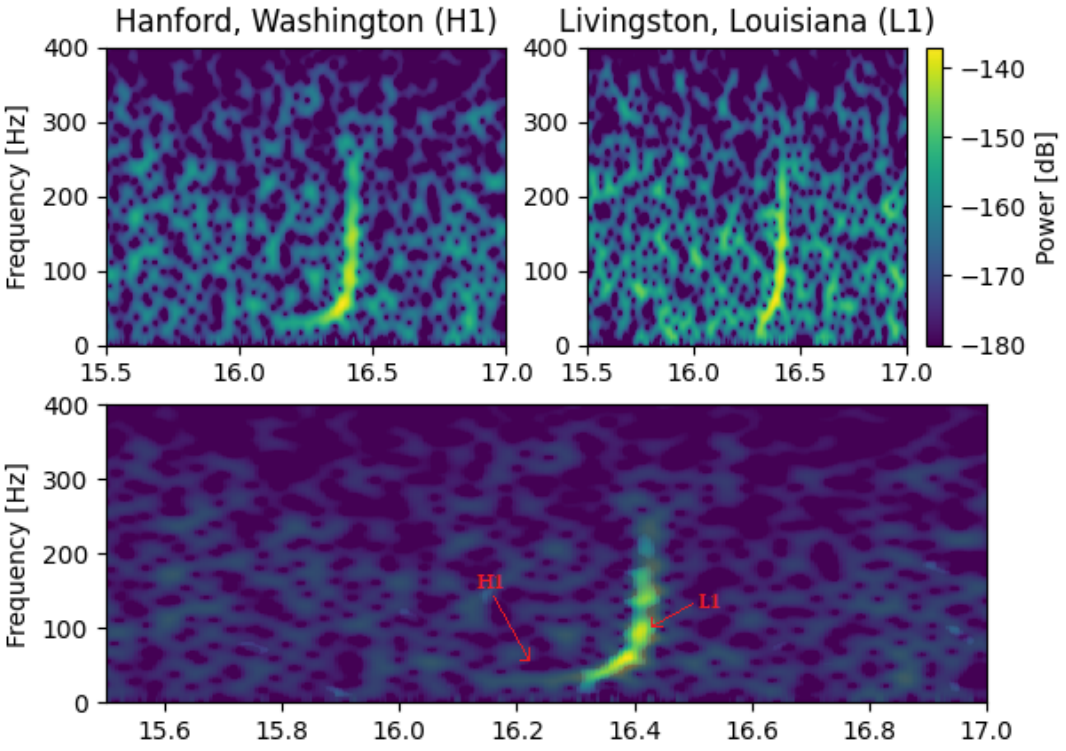

Figure 15: Dynamic power spectrum plot of Hanford and Livingston signal, at the bottom we have added a transparency to the Livingstone signal on top of the Hanford signal. Red marker shows which signal it is, and the gravitational wave to the signals.

Task 9: Improvement

To improve the signal processing part of task, I'm adding a different window and a band pass filter. I am gone use a window called Kaiser window, which have a adjustable parameter β and which would maximize the energy concentration. The band pass filter are we using, passes frequencies between 25 and 300Hz. By doing this we getting rid of a lot of noise in the LIGO signal, according to the original paper to the gravitational waves they had a band pass filter set to 35 $\rightarrow$ 350Hz. In Figure 16 we have the time domain, where have the low-pass and band-pass filtered Hanford signal. Which shows a great difference, between the two filters. The band-pass filter smoothing the signal, and gets rid of not wanted noise.

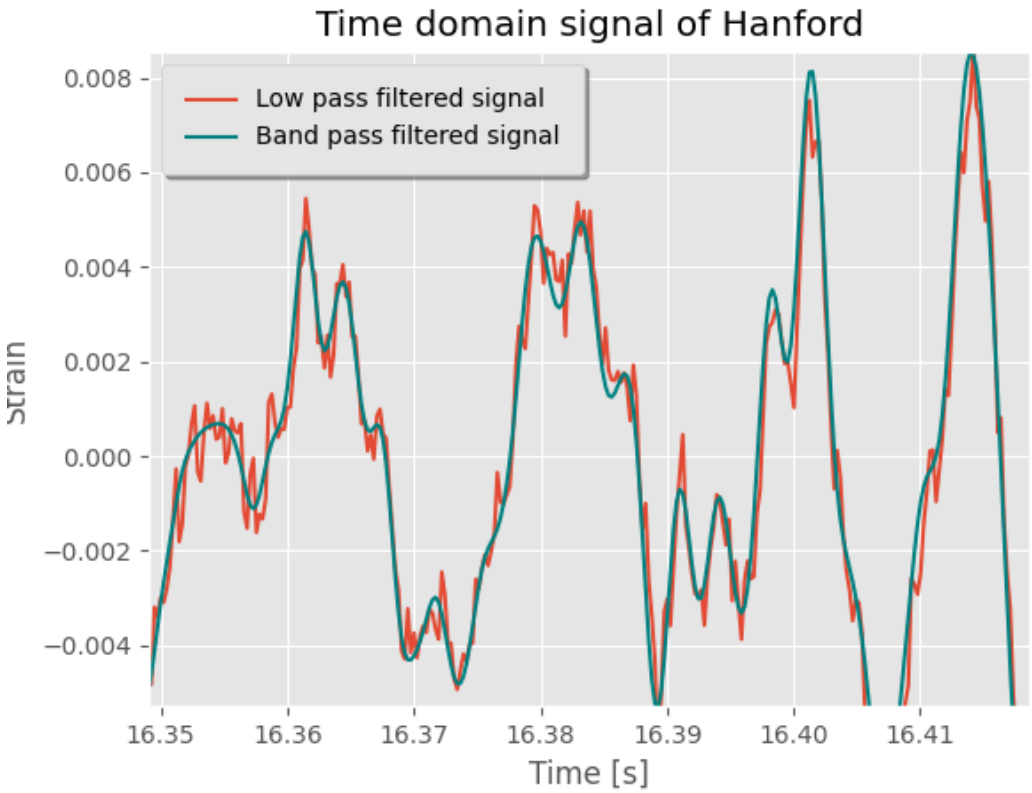

Figure 16: Low-pass filtered and band-pass filtered Hanford signal

Figure 17 shows how the band-pass filtered Hanford signal, is compered to the low-pass filtered Hanford signal. As expected we can see that the noise at 300Hz and up is gone, and from 0Hz to approximately 35Hz is also gone. Compared to the low-pass filtered signal, that have a lot of noise in these regions. We can also see that the gravitation wave for both of the filters, are pretty clear. But the band-pass filter makes the gravitation wave signal somewhat clearer and narrower, regarding the noise that it have filtered out. I also used a longer processing time to, get smaller pixels in the plot, that did the plot smoother.

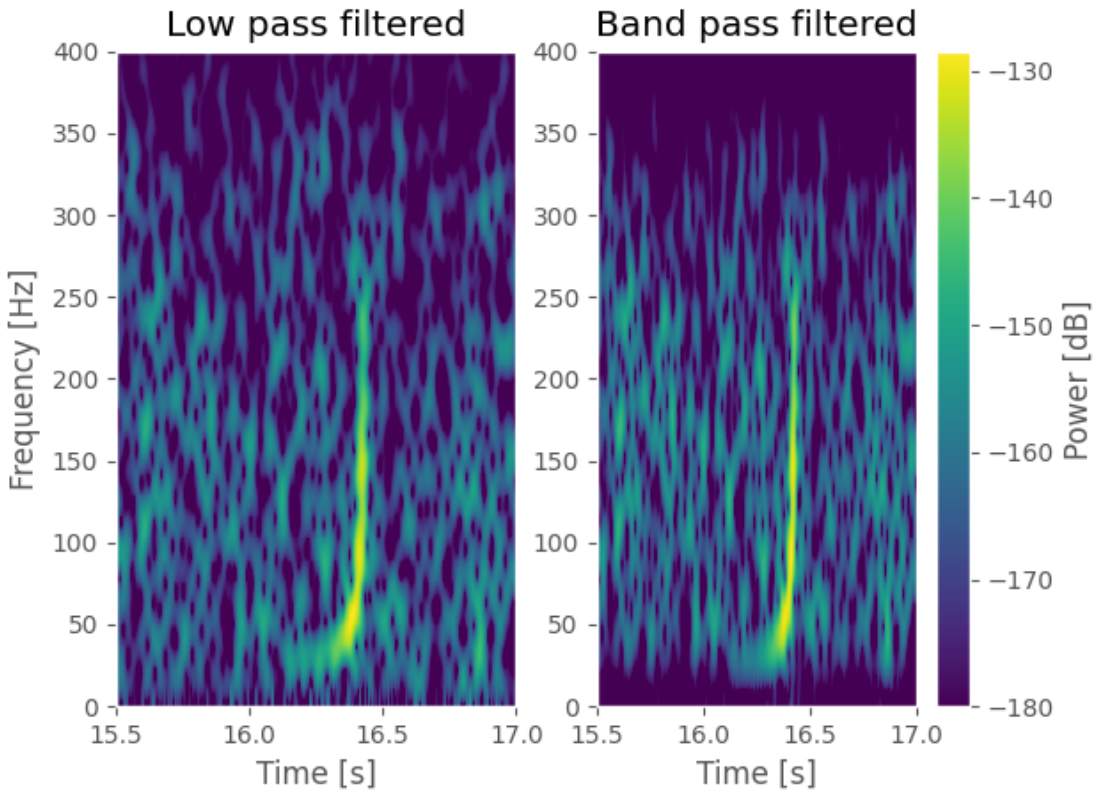

Figure 17: Dynamic spectrum plot of the Hanford signal, with low-pass and band-pass filter.

Appendix

Task 1→9: (Without 3)

```
import numpy as np
import matplotlib.pyplot as plt
import h5py
import scipy.signal as sc
import sklearn as sk
import matplotlib.gridspec as gridspec

plt.style.use("ggplot") # use ggplot style for plots

f_s = 4096.0 # Hz, sample rate
N = 131072 # number of samples

def read_data(file_name):
    h = h5py.File(file_name,"r")
    data = h["strain/Strain"][()]
    detector_name = h["meta/Detector"][()]
    start_time = h["meta/UTCstart"][()]
    h.close()
    return detector_name, start_time, data

# read hanford measurement
h1_name, h1_start_time, h1_strain = read_data("H-H1_LOSC_4_V2
    -1126259446-32.hdf5")
# read livingston measurement
l1_name, l1_start_time, l1_strain = read_data("L-L1_LOSC_4_V2
    -1126259446-32.hdf5")

""" ----------- task 1 ----------- """

def task_1():
    """ ----------- task 1b ----------- """
    # Number of samples in the signal
    print("----------- task 1b -----------")
    print("Number of samples in H1 data:", len(h1_strain), "samples
    ")
    print("Number of samples in L1 data:", len(l1_strain), "samples
    ")

    """ ----------- task 1c ----------- """
    # Length of the signals in seconds
    print("----------- task 1c -----------")
    print("Duration of H1 data:", len(h1_strain)/f_s, "seconds")
    print("Duration of L1 data:", len(l1_strain)/f_s, "seconds")

    """ ----------- task 1e ----------- """
    #sample spacing in units of seconds (inverse of sample rate)
    print("----------- task 1e -----------")
    print("Sample spacing in H1 data:", round(1/f_s, 6), "seconds")
    print("Sample spacing in L1 data:", round(1/f_s, 6), "seconds")


""" ----------- task 2 ----------- """

def task_2():
```

```
    """ ----------- task 2a ----------- """
    t = np.arange(0, 32 , 1/f_s) # seconds of each sample of the array signal
    # Plot the data
    plt.figure(1)
    plt.subplot(211)
    plt.plot(t, h1_strain, label = "Hanford")
    plt.ylabel("Strain")
    plt.title("Hanford and Livingston data")
    plt.legend(fancybox = True, shadow = True)
    plt.subplot(212)
    plt.plot(t, l1_strain, color = "teal", label = "Livingston")
    plt.xlabel("Time [s]")
    plt.ylabel("Strain")
    plt.legend(loc = "upper left", fancybox = True, shadow = True)
    plt.show()

    """ ----------- task 2b ----------- """
    print("----------- task 2b -----------")
    print("Minimum value of H1 data:", min(h1_strain))
    print("Maximum value of H1 data:", max(h1_strain))
    print("Mean value of H1 data:", np.mean(h1_strain))
    print("")
    print("Minimum value of L1 data:", min(l1_strain))
    print("Maximum value of L1 data:", max(l1_strain))
    print("Mean value of L1 data:", np.mean(l1_strain))


""" ---------------- Task 4a ---------------- """
# windowed FFT of data
def windowed(data):
    x_hat = np.fft.rfft(sc.hann(len(data))*data)
    return x_hat
def freqs(data):
    freqs = np.fft.rfftfreq(len(data), 1/f_s)
    return freqs


""" ---------------- Task 4b ---------------- """
# Plot of the power spectrum of the LIGO signals
def task_4b():
    plt.figure(1)
    plt.subplot(211)
    plt.plot(np.fft.fftshift(freqs(h1_strain)), np.fft.fftshift(10.0*np.log10(np.abs(windowed(h1_strain))**2.0)), label = "Hanford")
    plt.xlim(0, 2100)
    plt.ylabel("Power [dB]")
    plt.legend(loc = "upper left",fancybox = True, shadow = True)
    plt.subplot(212)
    plt.plot(np.fft.fftshift(freqs(l1_strain)), np.fft.fftshift(10.0*np.log10(np.abs(windowed(l1_strain))**2.0)), color = "teal", label = "Livingston")
    plt.xlim(0, 2100)
    plt.xlabel("Frequency [Hz]")
    plt.ylabel("Power [dB]")
    plt.legend(loc = "upper left", fancybox = True, shadow = True)
    plt.suptitle("Power spectrum of the LIGO signals", fontsize=14)
```

```
    plt.show()


""" ---------------- Task 5b ---------------- """
# whitening filter
def whitening_filter(data):
    h_hat = 1.0/np.abs(windowed(data))
    return h_hat


""" ---------------- Task 5c ---------------- """
# Whitened and filtered signal, gives the whitened filtered signal
def whitened_signal(data):
    h_hat = whitening_filter(data)
    x_hat = windowed(data)
    x_whitened = np.fft.irfft(h_hat*x_hat)
    return x_whitened


""" ---------------- Task 5d ---------------- """

def task_5d():
    t = np.arange(0, 32, (1/f_s))

    fig, axs = plt.subplots(3, sharex=False, sharey=True)
    axs[0].plot(t, whitened_signal(h1_strain) , label = "Hanford")
    axs[0].legend(loc="upper left")
    axs[1].plot(t, whitened_signal(l1_strain), color = "teal",
   label = "Livingston")
    axs[1].legend(loc="upper left")
    axs[2].plot(t, whitened_signal(h1_strain), label = "Hanford")
    axs[2].plot(t, whitened_signal(l1_strain), color = "teal",
   label = "Livingston")
    axs[2].set_xlim(16.2,16.5)
    axs[2].set_xlabel("Time [s]")
    fig.text(0.009, 0.5, "Whitened strain y[n]", va='center',
   rotation='vertical', color = "dimgray", fontsize = 12)
    fig.suptitle("Whitened signals of " r"$y_H[n]$ and $y_L[n]$",
   fontsize=14)
    plt.show()



""" ---------------- Task 6a ---------------- """

omhat = np.linspace(-np.pi,  np.pi , num=1000) # Angular frequency

f_hertz = omhat/(2*np.pi*(1/f_s)) # Convert to Hz

frequency_response = np.zeros(len(omhat), dtype = complex) #
   Initialize frequency response

# Calculate frequency response
L = 8
for k in range(0,L):
    frequency_response = frequency_response + (1/L)*np.exp(-1j*
   omhat*k)

```

```
""" ---------------- Task 6b ---------------- """
def task_6b():
    ax1 = plt.subplot(211)
    ax1.margins(0.05)           # Default margin is 0.05, value 0 means fit
    ax1.plot(f_hertz, 10.0*np.log10(np.abs(frequency_response)**2.0), label = "Frequency response")
    ax1.scatter([310], [-6])
    ax1.axvline(x = 300, color = "dimgray", linestyle = "--")
    ax1.axhline(y = -6, color = "dimgray", linestyle = "--")
    ax1.set_ylabel("Power [dB]")
    ax1.set_title("Power spectral response of the filter")

    ax2 = plt.subplot(212)
    ax2.plot(f_hertz, 10.0*np.log10(np.abs(frequency_response)**2.0), label = "Frequency response")
    ax2.scatter([310], [-6])
    ax2.axvline(x = 300, color = "dimgray", linestyle = "--")
    ax2.axhline(y = -6, color = "dimgray", linestyle = "--")
    ax2.set_ylabel("Power [dB]")
    ax2.set_xlabel("Frequnecy [Hz]")
    plt.show()

""" ---------------- Task 6d ----------------   """

tau = 8.5e-4 # s ,time delay

# Running mean average low-pass filter
def average_filter(data):
    # Filter signal using an averaging filter (y[n] = 1/L * sum_{k=0}^{L-1} x[n-k])
    # for the whitened signal.
    filtered_signal = np.convolve(np.repeat(1.0/L, L), data, mode = "same")
    return filtered_signal

""" ---------------- Task 6e ---------------- """

t = np.arange(0, 32, (1/f_s)) # Time vector
time = t - tau # Shifted time

""" ---------------- Task 6f ---------------- """

# Plot the filtered signals
def task_6f():
    fig, axs = plt.subplots(3, sharex=True, sharey=True)
    axs[0].plot(time, average_filter(whitened_signal(h1_strain)), label = "Hanford")
    axs[0].legend(loc="upper left", fancybox = True, shadow = True)
    axs[1].plot(time, average_filter(whitened_signal(l1_strain)), color = "teal", label = "Livingston")
    axs[1].legend(loc="upper left", fancybox = True, shadow = True)
    axs[2].plot(time, average_filter(whitened_signal(h1_strain)), label = "Hanford")
    axs[2].plot(time, average_filter(whitened_signal(l1_strain)), color = "teal", label = "Livingston")
    axs[2].set_xlim(16.1,16.6)
    axs[2].set_xlabel("Time [s]")
```

```
205     fig.text(0.009, 0.5, "Strain", va='center', rotation='vertical'
        , color = "dimgray", fontsize = 12)
206     fig.suptitle("Low pass filtered whitened Hanford and Livingston
        signals", fontsize=14)
207     plt.show()
208 
209     fig, axs = plt.subplots(2, sharex=True, sharey=True)
210     axs[0].plot(t, whitened_signal(h1_strain), label = "Hanford")
211     axs[0].plot(t, whitened_signal(l1_strain), color = "teal",
        label = "Livingston")
212     axs[0].set_title("Whitened Hanford and Livingston signals")
213     axs[0].legend(loc="upper left", fancybox = True, shadow = True)
214     axs[1].plot(time, average_filter(whitened_signal(h1_strain)),
        label = "Hanford")
215     axs[1].plot(time, average_filter(whitened_signal(l1_strain)),
        color = "teal", label = "Livingston")
216     axs[1].set_xlim(16.1,16.6)
217     axs[1].set_xlabel("Time [s]")
218     axs[1].set_title(r"$\downarrow$Low pass filtered$\downarrow$")
219     fig.text(0.009, 0.5, "Strain", va='center', rotation='vertical'
        , color = "dimgray", fontsize = 12)
220     plt.show()
221 
222     plt.plot(time, average_filter(whitened_signal(h1_strain)),
        label = "Hanford")
223     plt.show()
224 
225 
226 """ ---------------- Task 7a ---------------- """
227 def time_delay(data1, data2):
228     # Finding the time value for the highest peak in the two
        signals
229     # than finding the corresponding index in the time vector.
230     time_value_s1 = time[np.abs(average_filter(whitened_signal(
        h1_strain))).argmax()]
231     time_value_s2 = time[np.abs(average_filter(whitened_signal(
        l1_strain))).argmax()]
232 
233     n_0 = time_value_s1 - time_value_s2 # Time delay
234     New_time = time + n_0 # Shifted time vector with the time delay
235     return New_time, print("The time delay is: ", n_0, "s")
236 
237 def task_7a():
238     plt.plot(time, np.abs(average_filter(whitened_signal(h1_strain)
        )), label = "Hanford")
239     plt.plot(time_delay(h1_strain, l1_strain), np.abs(
        average_filter(whitened_signal(l1_strain))), color = "teal",
        label = "Livingston")
240     plt.xlim(16.4,16.425)
241     plt.xlabel("Time [s]")
242     plt.ylabel("Magnitude (strain)")
243     plt.legend(loc="upper left", fancybox = True, shadow = True)
244     plt.show()
245 
246 
247 
248 """ ---------------- Task 8a ---------------- """
249 # Defining each of the LGIO signal with Running mean average low-
```

```
    pass filter with whitened signal.
filtered_l1= average_filter(whitened_signal(l1_strain))
filtered_h1 = average_filter(whitened_signal(h1_strain))
# Calculation for the dynamic spectrum of the low-pass filtered
    whitened signal.
f, t, filtered_h1 = sc.spectrogram(filtered_h1, fs=f_s, window="
    hann", nperseg=400, noverlap=380, nfft=4096)
f, t, filtered_l1 = sc.spectrogram(filtered_l1, fs=f_s, window="
    hann", nperseg=400, noverlap=380, nfft=4096)

""" ---------------- Task 8b ---------------- """

def task_8b():
    plt.pcolormesh(t, f, 10.0*np.log10(np.abs(filtered_h1)**2.0),
    vmin=-180, cmap="viridis", shading="auto")
    #plt.pcolormesh(t, f, 10.0*np.log10(np.abs(filtered_l1)**2.0),
    shading = "auto", cmap = "viridis", vmin= -180)
    plt.xlim(15.5,17)
    plt.ylim([0, 400.0])
    plt.colorbar(label="Power [dB]")
    plt.xlabel("Time [s]")
    plt.ylabel("Frequency [Hz]")
    plt.title("Dynamic spectrum Hann window")
    plt.show()


def task_8b_comp():
    # Create 2x2 sub plots
    gs = gridspec.GridSpec(2, 2)
    plt.figure()
    plt.subplot(gs[0, 0]) # row 0, col 0
    plt.pcolormesh(t, f, 10.0*np.log10(np.abs(filtered_h1)**2.0),
    vmin=-180, cmap="viridis", shading="auto")
    plt.xlim(15.5,17)
    plt.ylim([0, 400.0])
    plt.ylabel("Frequency [Hz]")
    plt.title("Hanford, Washington (H1)")

    plt.subplot(gs[0, 1]) # row 0, col 1
    plt.pcolormesh(t, f, 10.0*np.log10(np.abs(filtered_l1)**2.0),
    shading = "auto", cmap = "viridis", vmin= -180)
    plt.xlim(15.5,17)
    plt.ylim([0, 400.0])
    plt.colorbar(label="Power [dB]")
    plt.title("Livingston, Louisiana (L1)")

    plt.subplot(gs[1, :]) # row 1, span all columns
    plt.pcolormesh(t, f, 10.0*np.log10(np.abs(filtered_h1)**2.0),
    vmin=-180, cmap="viridis", shading="auto")
    plt.pcolormesh(t, f, 10.0*np.log10(np.abs(filtered_l1)**2.0),
    shading = "auto", cmap = "viridis", vmin= -150, alpha=0.5)
    plt.xlim(15.5,17)
    plt.ylim([0, 400.0])
    plt.ylabel("Frequency [Hz]")
    plt.show()


""" ---------------- Task 9 ---------------- """
```

```
298 def kaiser_window(data):
299      x_hat = np.fft.rfft(sc.kaiser(len(data), beta=9)*data, 2*N)
300      return x_hat
301
302
303 def bandpass_filter(data):
304
305     lowcut = 25.0 # Hz
306     highcut = 300.0 # Hz
307
308     nyq = 0.5*f_s # Nyquist frequency
309     low = lowcut/nyq # Normalized lowcut frequency
310     high = highcut/nyq # Normalized highcut frequency
311
312     order = 3 # order of the filter
313
314     b, a = sc.butter(order, [low, high], btype="bandpass", analog=
    False) # Butterworth filter
315      y = sc.filtfilt(b, a, data, axis = 0) # This function applies a
     linear digital filter twice, once forward and once backwards.
316     return y # Returns a bandpass filtered signal
317
318 # Defining each of the LGIO signal with Running mean average band-
    pass filter with whitened signal.
319 bandpass_filtered_h1 = bandpass_filter(whitened_signal(h1_strain))
320 # Calculation for the dynamic spectrum of the band-pass filtered
    whitened signal.
321 f, t, bandpass_filtered_h1 = sc.spectrogram(bandpass_filtered_h1,
    fs=f_s, window=("kaiser", 8.6), nperseg=400, noverlap=380, nfft
    =4096)
322
323 def power_spectrum_Bandfilt():
324     plt.pcolormesh(t, f, 10.0*np.log10(np.abs(bandpass_filtered_h1)
    **2.0), vmin=-180, cmap="viridis", shading="auto")
325     plt.xlim(15.5,17)
326     plt.ylim([0, 400.0])
327     plt.colorbar(label="Power [dB]")
328     plt.xlabel("Time [s]")
329     plt.ylabel("Frequency [Hz]")
330     plt.title("Dynamic spectrum bandpass filtered signal")
331     plt.show()
332
333 def comp_between_bandlow():
334     # Plot of the low pass filtered whitened and band pass filered
    whitened LIGO signals
335     plt.plot(time,  average_filter(whitened_signal(h1_strain)),
    label = "Low pass filtered signal")
336     plt.plot(time, bandpass_filter(whitened_signal(h1_strain)),
    label = "Band pass filtered signal", color = "teal")
337     plt.xlim(15.5,17)
338     plt.xlabel("Time [s]")
339     plt.ylabel("Strain")
340     plt.title("Time domain signal of Hanford")
341     plt.legend(loc="upper left", fancybox=True, shadow=True,
    borderpad=1)
342     plt.show()
343
344     # Power spectrum plot of the low pass filtered whitened and
```

```
    band pass filered whitened LIGO signals
345     # Low pass filtered  Hanford signal
346     plt.subplot(1, 2, 1) # row 1, col 2 index 1
347     plt.pcolormesh(t, f, 10.0*np.log10(np.abs(filtered_h1)**2.0),
    vmin=-180, cmap="viridis", shading="auto")
348     plt.title("Low pass filtered")
349     plt.xlim(15.5,17)
350     plt.ylim([0, 400.0])
351     plt.xlabel("Time [s]")
352     plt.ylabel("Frequency [Hz]")
353 
354     # Band pass filtered Hanford signal
355     plt.subplot(1, 2, 2) # index 2
356     plt.pcolormesh(t, f, 10.0*np.log10(np.abs(bandpass_filtered_h1)
    **2.0), vmin=-180, cmap="viridis", shading="auto")
357     plt.title("Band pass filtered")
358     plt.xlim(15.5,17)
359     plt.ylim([0, 400.0])
360     plt.colorbar(label="Power [dB]")
361     plt.xlabel("Time [s]")
362     plt.savefig("bandpass_lowpass_filtered.png")
363     plt.show()
364 
365 if __name__ == "__main__":
366     #task_1()    # Data
367     #task_2()    # Plotting the data
368     #task_4b()   # Plot of the power spectrum of the LIGO signals
369     #task_5d()   # Plot of the whitened LIGO signals
370     #task_6b()   # Plot of power spectral response of the filter
371     task_6f()  # Plot of the low pass filtered whitened LIGO
    signals
372     #task_7a()  # Plot of the corrected time delay between the two
    signals
373     #task_8b()  # Plot of the dynamic spectrum of the LIGO signal
374     #task_8b_comp() # Plot of the dynamic spectrum of the LIGO
    signals and comparisson
375     #power_spectrum_Bandfilt() # Plot of the dynamic spectrum of
    the band-pass filtered whitened signal
376     #comp_between_bandlow() # Plot of the low pass filtered
    whitened and band pass filered whitened LIGO signals
377     pass
```

Task 3:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as s

plt.style.use("ggplot")

f_s = 4096 # Hz, sample rate
f = 31.5 # Hz, frequency of the signal

n = np.arange(4096) # number of samples

x_n = np.cos(2*np.pi*f*n/f_s) # signal
w_hann = s.hann(len(n)) # Hann window of length 4096 a)
w_x = w_hann*x_n # Hann windowed signal

""" ----------------- task 3b ----------------- """
# plot the signal and the Hann window signal
plt.plot(x_n, label = "signal x[n]")
plt.plot(w_x, label = "Hann window w[n]x[n]")
plt.legend(bbox_to_anchor=(0, 1.1), loc=2, borderaxespad=0,
    fancybox = True, shadow = True)
plt.show()

""" ----------------- task 3c ----------------- """

f_k = np.fft.fftfreq(f_s, 1/f_s) # frequency in Hz, array of length
     n containing the sample frequencies.
f_shifted = np.fft.fftshift(f_k) # frequency
print("Frequencies corresponding to principal spectrum:", f_shifted
    , "Hz")

""" ----------------- task 3d ----------------- """

# function to find the value of k corresponding to the frequency
    f_k nearest to -31.5 Hz and 31.5 Hz
def find_nearest(array, value):
    array = np.asarray(array) # convert array to numpy array
    idx = (np.abs(array - value)).argmin() # find index of the
    value in the array closest to the value
    return idx

print("k=", find_nearest(f_k, 31.5), "Hz, corresponds to the
    frequency 31.5 Hz")
print("k=",find_nearest(f_k, -31.5), "Hz, corresponds to the
    frequency -31.5 Hz")

""" ----------------- task 3e ----------------- """

plt.vlines(x=-31.5*2*np.pi/f_s, ymin=-150, ymax=55, linestyle = "--
    ", color="black", lw = 1, label = r"$\pm31.5 $Hz")
plt.vlines(x=31.5*2*np.pi/f_s, ymin=-150, ymax=55, linestyle = "--"
    , color="black", lw = 1)
plt.plot(f_shifted*2*np.pi/f_s, 10.0*np.log10(np.abs(np.fft.
    fftshift(np.fft.fft(x_n)))**2.0), label = "DFT", color = "teal")
plt.plot(f_shifted*2*np.pi/f_s, 10.0*np.log10(np.abs(np.fft.
    fftshift(np.fft.fft(w_x)))**2.0), label="Windowed DFT", color =
```

```
    "orangered")
plt.xlabel("Frequency [Hz]")
plt.ylabel("Power [dB]")
plt.legend(bbox_to_anchor=(0, 1.05),loc = "upper left", fancybox =
    True, shadow = True)
plt.show()
```