FYS2006: Coding assignment 3

Candidate 35

November 2021

1 Task 1

1a) Code was written to read the Hanford and Livingston signals $x_H[n]$ and $x_L[n]$ (see listing 1).
1b) $X_H[n]$ and $X_L[n]$ both contain $N = 131,072$ samples.
1c) The signals represent $0.000244140625 \cdot 131072s = 32s$.
1d) The sampling rate of both signals is $f_s = 4096Hz$. This means that for the signal to not be undersampled, it should not have frequency components with frequencies higher than 2048 Hz.
1e) The sample spacing is 0.000244140625 seconds.

Listing 1: Task 1. Reading data from hdf5 file.

```python
import h5py

X_H_FILE, X_L_FILE = "H1_LOSC.hdf5", "L1_LOSC.hdf5"
f_s = 4096

# Read Hanford and Livinstons signals.
def get_strain(file_name):
    d = h5py.File(file_name, "r")
    strain = d["strain/Strain"][()]
    return strain

x_H = get_strain(X_H_FILE)
x_L = get_strain(X_L_FILE)

# How many samples are in each of the signals?
print(f"x_H & x_L contain {len(x_H)} & {len(x_L)} samples.")

# How many seconds of signal do the signals represent?
print(f"x_H & x_L represent {len(x_H)/f_s} & {len(x_L)/f_s} seconds.")

# Is the samplerate high enough to retain frequencies in range (-300, 300)?
print(f"""The sample rate is {f_s}Hz. This is high enough to contain
    frequencies in the range {-f_s/2} to {f_s/2} Hz.""")

# What is the sample spacing?
print(f"The sample spacing is {1/f_s:.3E} seconds.")
```

2 Task 2

See figures 1 and 2 for graphs and maximum, minimum, and averages. See listing 2 for code.

Listing 2: Task 2

```python
import matplotlib.pyplot as plt
import h5py
import numpy as np

X_H_FILE, X_L_FILE = "H1_LOSC.hdf5", "L1_LOSC.hdf5"

def plot_signal(file_name):
    d = h5py.File(file_name, "r")
    strain = d["strain/Strain"][()]
    duration = d["meta/Duration"][()]
    name = d["meta/Detector"][()].decode()
    d.close()
    plt.plot(np.linspace(0, duration, strain.shape[0]), strain)
    title = f"{name}. Min: {np.min(strain):.3E}, Max: {np.max(strain):.3E}, Avg: {np.mean(strain):
    plt.title(title)
    plt.ylabel("Strain")
    plt.xlabel("Seconds")
    plt.show()

for file_name in [X_H_FILE, X_L_FILE]:
    plot_signal(file_name)
```

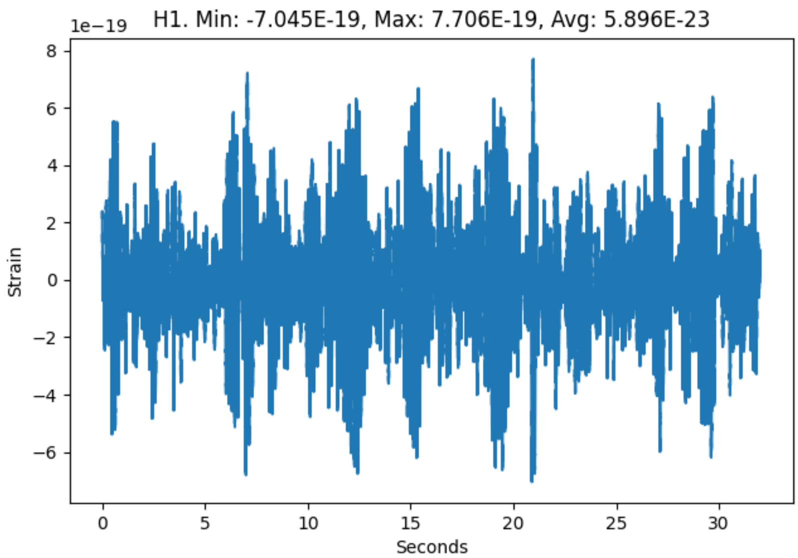

Figure 1: Plot of signal H1.

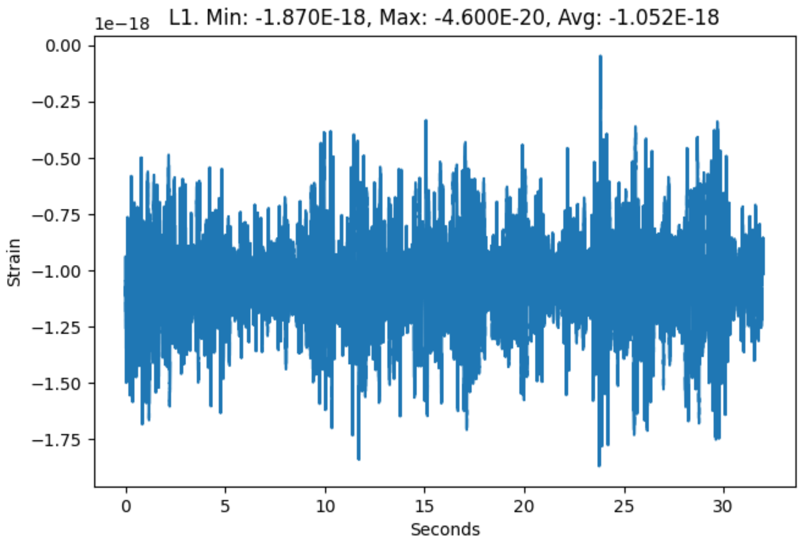

Figure 2: Plot of signal L1.

3 Task 3

3a) See listing 3 for code.
3b) See figure 3 for graph.
3c) A frequency $\hat{\omega}_k = \pi k/N$ rad/sample corresponds to the frequency $(k - \frac{N}{2}) \cdot f_s/N$ Hz.
3d) This means that the value k corresponding to f_k is $k = \frac{N}{2} + f_k \cdot N/f_s = f_k + 2048$. For $f_k = 31.5Hz$, this gives $k = 2079.5$. For $f_k = -31.5Hz$, $k = 2016.5$. Since k is an integer, we can round to 2080 and 2017, respectively.
3e & 3f) See figure 4 for graph.
3g) Autocorrelation
3h) -267.5 - (-72.2) dB = -195.3 dB

Listing 3: Task 3

```python
import matplotlib.pyplot as plt
import numpy as np

N = 4096
f_s = 4096
f = 31.5

def to_decibel(x):
    return 10 * np.log10(np.abs(x)**2)

# 3a: Choose a tapered window function and implement it
def apply_window(x):
    filter = np.hanning(len(x))
    return x * filter

# Apply your window to a sinusoidal signal
def task_3b():
    n = np.arange(0, N, 1, dtype=np.float_)
    x = np.cos(2 * np.pi * f * n / f_s)
    w_x = apply_window(x)
    plt.plot(n, x, label="x[n]")
    plt.plot(n, w_x, label="w[n]x[n]", alpha=0.8)
    plt.xlabel("Samples [n]")
    plt.legend(loc="upper right")
    plt.show()


# Estimate power spectrum of windowed signal and non-windowed signal.
# Plot both power spectrums in dB. Mark 31.5 and -31.5Hz.
def task_3ef():
    n = np.arange(0, N, 1, dtype=np.float_)
    x = np.cos(2 * np.pi * f * n / f_s)
    w_x = apply_window(x)

    freqs = np.fft.fftfreq(N, d=1/f_s)
    freqs = np.fft.fftshift(freqs)

    X = np.fft.fft(x)
    X = np.abs(X) / len(X)
    X = np.fft.fftshift(X)

    w_X = np.fft.fft(w_x)
    w_X = np.abs(w_X) / len(w_X)
    w_X = np.fft.fftshift(w_X)
```

```python

    fig, [ax1, ax2] = plt.subplots(nrows=1, ncols=2, sharey=True)
    ax2.tick_params(labelleft=True)

    ax1.plot(freqs, to_decibel(X))
    ax1.axvline(31.5, ls=":", c="r")
    ax1.axvline(-31.5, ls=":", c="r")
    ax1.set_title("Power spectrum of signal x[n]")
    ax1.set_xlabel("Freq [Hz]")
    ax1.set_ylabel("Sq. magnitude [dB]")

    ax2.plot(freqs, to_decibel(w_X))
    ax2.axvline(31.5, ls=":", c="r")
    ax2.axvline(-31.5, ls=":", c="r")
    ax2.set_title("Power spectrum of windowed signal w[n]x[n]")
    ax2.set_xlabel("Freq [Hz]")
    ax2.set_ylabel("Sq. magnitude [dB]")

    plt.show()
    # 3h)
    print(f"X[pi] = {to_decibel(X)[-1]}, w_X[pi] = {to_decibel(w_X)[-1]}")


# Execute tasks:
task_3b()
task_3ef()
```

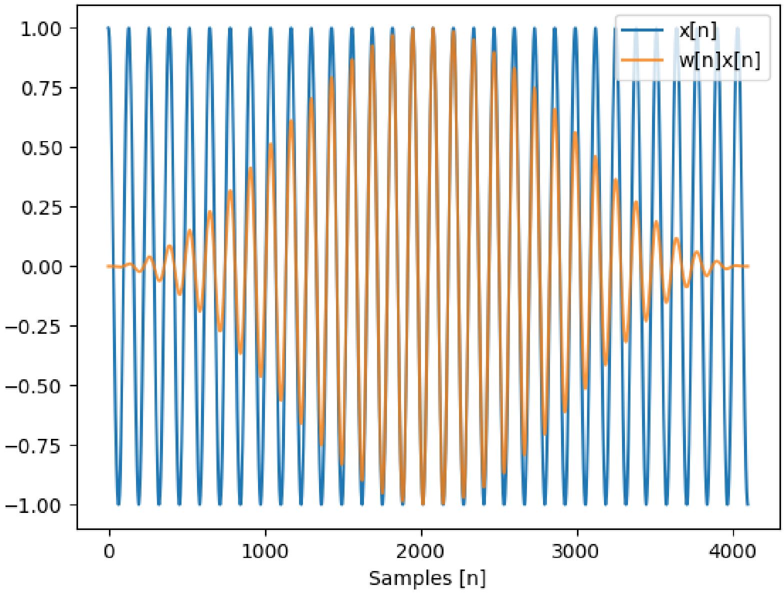

Figure 3: Window function applied to sinusoidal function.

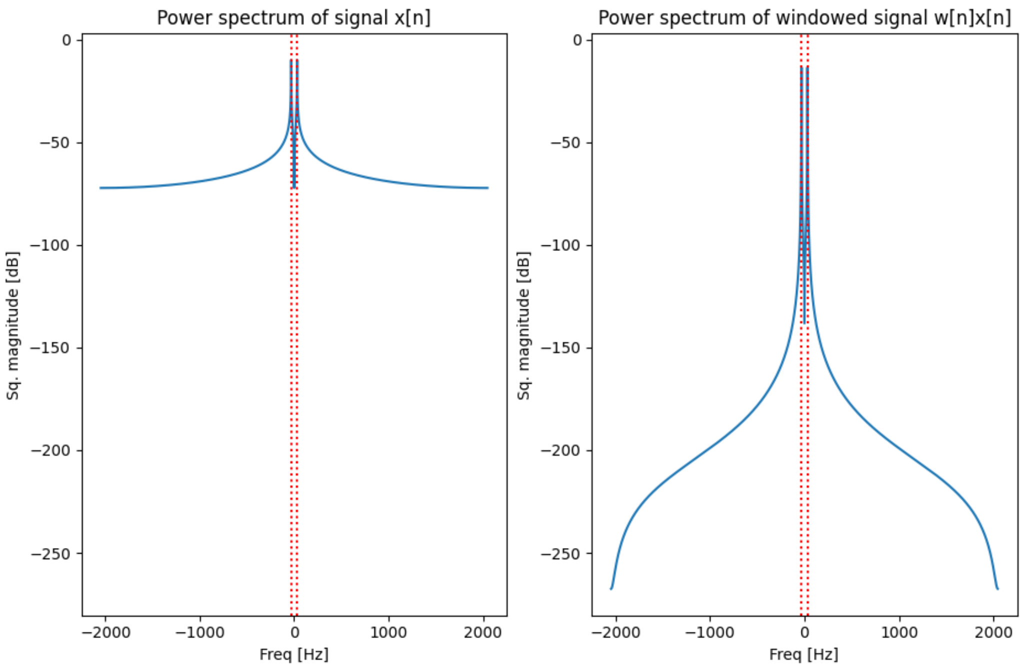

Figure 4: Task 3e, 3f. Power spectrum of windowed and non-windowed signal.

4 Task 4

Listing 4: Task 4

```
import matplotlib.pyplot as plt
import numpy as np
import h5py
X_H_FILE, X_L_FILE = "H1_LOSC.hdf5", "L1_LOSC.hdf5"
f_s = 4096

def get_strain(file_name):
    return h5py.File(file_name, "r")["strain/Strain"][()]

def apply_window(x):
    return x * np.hanning(len(x))

def dB(x):
    return 10 * np.log10(np.abs(x)**2)

# Calculate the power spectrum of the squared abs of the DFT of the windowed LIGO signals.
def task_4ab():
    x_H = get_strain(X_H_FILE)
    x_L = get_strain(X_L_FILE)
    N = len(x_H)

    xw_H = apply_window(x_H)
    xw_L = apply_window(x_L)

    Xh_H = np.fft.fft(xw_H)
    Xh_L = np.fft.fft(xw_L)

    freqs = np.fft.fftfreq(N, d=1/f_s)

    idx = N // 2

    fig, [ax1, ax2] = plt.subplots(nrows=1, ncols=2, sharey=True)
    ax2.tick_params(labelleft=True)

    ax1.plot(freqs[:idx], dB(Xh_H[:idx]))
    ax1.set_title("Power spectrum of "+"$|\^x_L[k]|^2$")
    ax1.set_xlabel("Freq [Hz]")
    ax1.set_ylabel("Power [dB]")
    ax1.legend(["H1"])

    ax2.plot(freqs[:idx], dB(Xh_L[:idx]))
    ax2.set_title("Power spectrum of "+"$|\^x_H[k]|^2$")
    ax2.set_xlabel("Freq [Hz]")
    ax2.set_ylabel("Power [dB]")
    ax2.legend(["L1"])

    # Plot some regions with narrow band interference
    ax1.axvspan(985, 1040, facecolor="r", alpha=0.5)
    ax2.axvspan(985, 1040, facecolor="r", alpha=0.5)
    ax1.axvspan(490, 525, facecolor="r", alpha=0.5)
    ax2.axvspan(490, 525, facecolor="r", alpha=0.5)
    plt.show()
task_4ab()
```

4a) See listing 4.
4b+c) See figure 6.

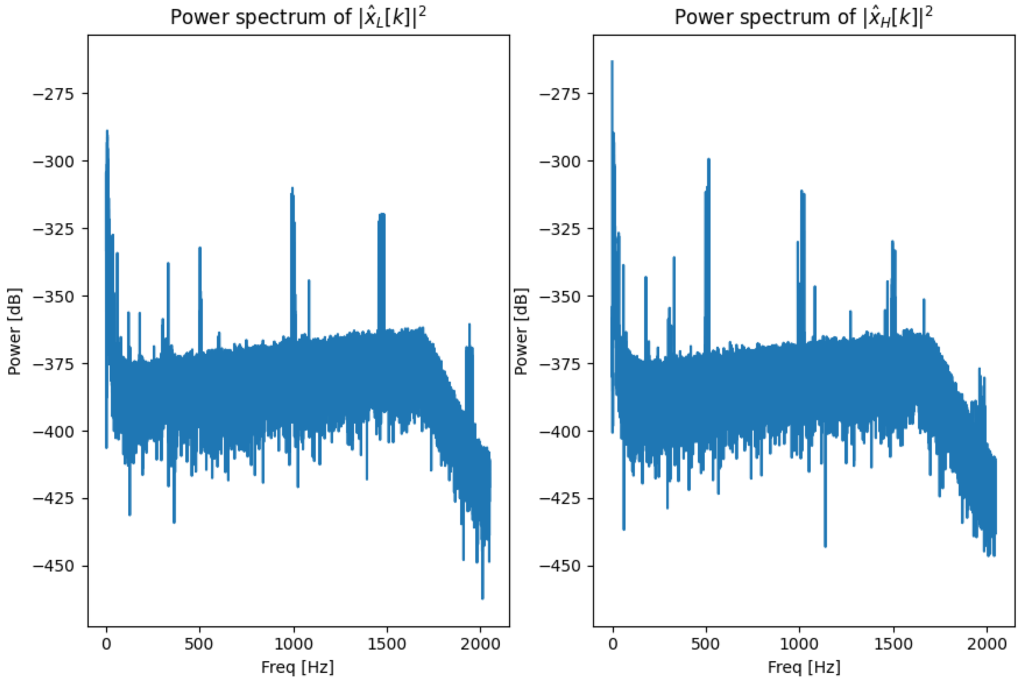

Figure 5: Task 4. H1 on the left, L1 on the right.

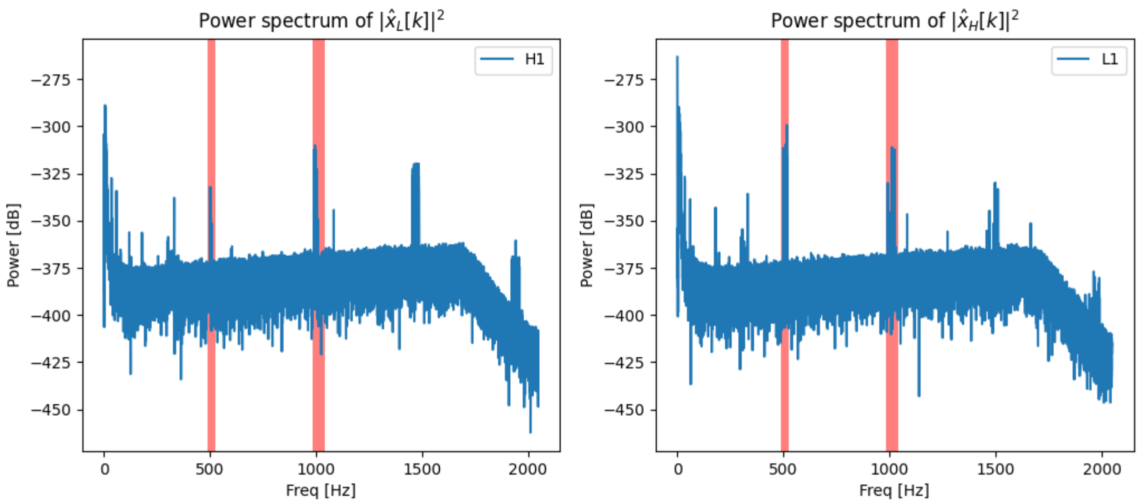

Figure 6: Two bands containing narrow band interference marked: 490-525Hz and 985-1040Hz.

5 Task 5

Listing 5: Task 5

```
import matplotlib.pyplot as plt
import numpy as np
import h5py

X_H_FILE, X_L_FILE = "H1_LOSC.hdf5", "L1_LOSC.hdf5"
f_s = 4096

def get_strain(file_name):
    return h5py.File(file_name, "r")["strain/Strain"][()]

def apply_window(x):
    return x * np.hanning(len(x))

# 5b) Implement a whitening filter in frq. domain hH[k] for x_H and x_L.

def whiten_signal_fd(x):
    xw = x * np.hanning(len(x)) # Apply window to signal
    Xw = np.fft.rfft(xw) # FFT of windowed signal
    hH = 1 / np.abs(Xw) # Make filter
    yH = Xw * hH # Apply filter
    return yH

# 5c)
# Use filter to whiten signal and transform it to time-domain.

x_H = get_strain(X_H_FILE)
x_L = get_strain(X_L_FILE)

N = len(x_H)
# Whiten signals
yH_H = whiten_signal_fd(x_H)
yH_L = whiten_signal_fd(x_L)

# Transform to time-domain
y_H = np.fft.irfft(yH_H)
y_L = np.fft.irfft(yH_L)

# 5d)
# Plot whitened signals
fig, [ax1, ax2] = plt.subplots(nrows=1, ncols=2, sharey=True)
t = np.linspace(0, N/f_s, N)

ax1.set_xlim([16.2, 16.5])
ax2.set_xlim([16.2, 16.5])

ax1.plot(t, y_H)
ax1.legen
ax2.plot(t, y_L)
ax1.set_ylabel("Strain")
ax1.set_xlabel("Seconds")
ax2.set_xlabel("Seconds")

plt.show()
```

5b+c) See listing 5.
5d) See figure 8.

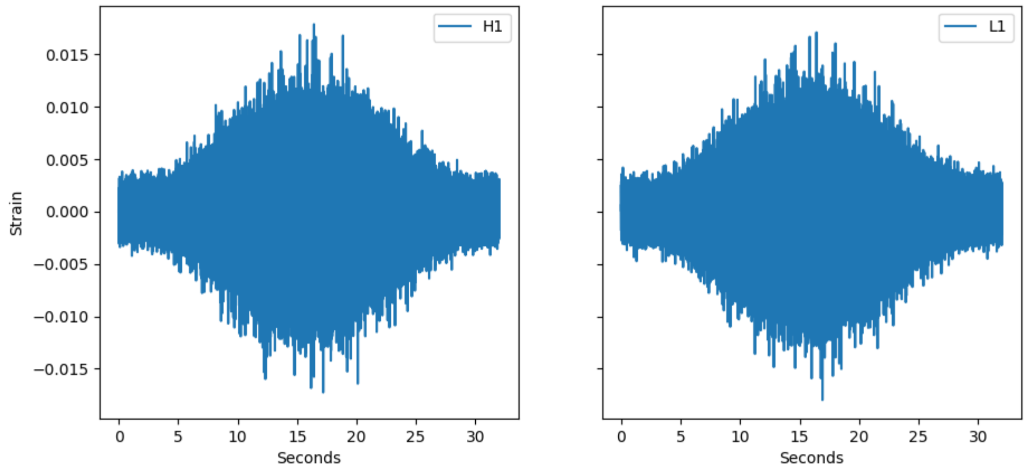

Figure 7: H1 and L1 signals after whitening filter.

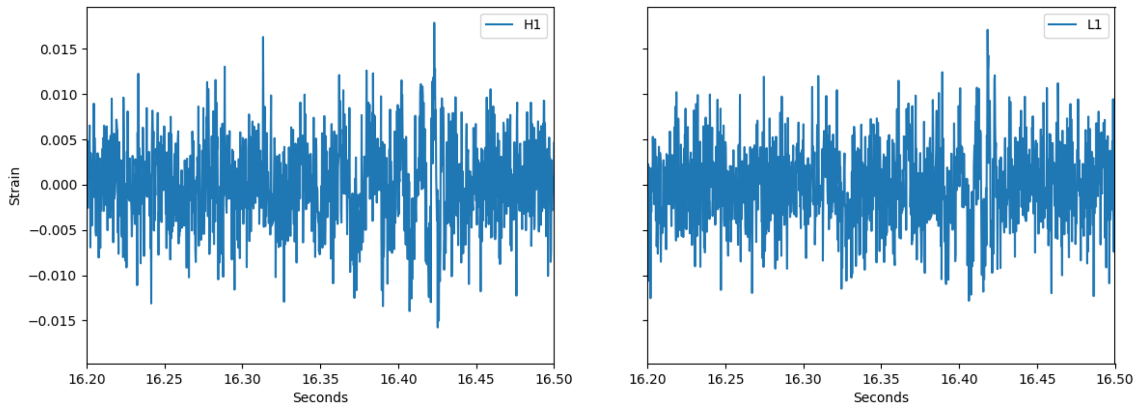

Figure 8: H1 and L1 signals after whitening filter, with x-axis between 16.2 and 16.5 seconds.

6 Task 6

Listing 6: Task 5

```
import matplotlib.pyplot as plt
import numpy as np
import h5py

X_H_FILE, X_L_FILE = "H1_LOSC.hdf5", "L1_LOSC.hdf5"
f_s = 4096

def get_strain(file_name):
    return h5py.File(file_name, "r")["strain/Strain"][()]

# Whiten signal (time-dimension)
def whiten_signal_td(x):
    xw = x * np.hanning(len(x)) # Apply window to signal
    Xw = np.fft.rfft(xw) # FFT of windowed signal
    hH = 1 / np.abs(Xw) # Make filter
    yH = Xw * hH # Apply filter
    return np.fft.irfft(yH)


# 6a)
# Find an integer value of L such that the low-pass filter will reduce
# the power of frequency components at f = 300Hz by approximately
# -6 dB compared to the filter output for a f = 0 Hz signal

# This was determined experimentally to be 8
filter_len = 8

def apply_filter(x, length):
    w = np.ones(length, dtype=np.float_) / length
    return np.convolve(x, w, "valid")

# 6b)
# Plot the power spectral response of the filter in dB scale.
def task_6b(filter_len, f_s):
    # Equation 574 in "13 - Frequency response":
    # H(omegaH) = sum(h[k]*e**-i*omegaH*k)
    n = 4096

    omegaH = np.linspace(-np.pi, np.pi, n)
    freq = omegaH * f_s / (2 * np.pi)

    H = np.zeros(n, dtype=np.complex)
    for k in range(filter_len):
        H += (1/filter_len) * np.exp(-1j*omegaH*k)

    plt.plot(freq, 10*np.log10(np.abs(H)**2))
    plt.grid()
    plt.axhline(-6, ls=":", c="g")
    plt.axvline(300, ls=":", c="g")
    plt.ylabel("$10\log_{10}|H(\hat{\omega})|^2$")
    plt.xlabel("Frequency [Hz]")
    plt.show()

```

```
# 6c)
# What is the time delay added by the filter?
"""The filter delays the signal by 4.5 samples, due to the convolution"""


def task_6cdef():
    # 6d)
    # Apply the filter to the whitened H and L signals
    x_H = get_strain(X_H_FILE)
    x_L = get_strain(X_L_FILE)

    y_H = whiten_signal_td(x_H)
    y_L = whiten_signal_td(x_L)

    lopass_H = apply_filter(y_H, filter_len)
    lopass_L = apply_filter(y_L, filter_len)

    # 6e)
    # Undo the time shifting
    def get_t(filter_len, N, tot_time, f_s):
        n = N - filter_len + 1
        t_start = (filter_len + 1) / (f_s * 2)
        t_end = tot_time - t_start
        return np.linspace(t_start, t_end, n)

    # 6f)
    # Plot the low-pass filtered H and L signals between 16.1 and 16.6 seconds.

    N = len(x_H)
    t = get_t(filter_len, N, N/4096, f_s)

    fig, [ax1, ax2] = plt.subplots(nrows=2, ncols=1)
    ax1.set_xlim([16.1, 16.6])
    ax2.set_xlim([16.1, 16.6])

    ax1.grid()
    ax2.grid()

    ax1.plot(t, lopass_H)
    ax2.plot(t, lopass_L)

    ax1.legend(["H1"])
    ax2.legend(["L1"])
    ax1.set_ylabel("Strain")
    ax2.set_xlabel("Time [s]")
    ax2.set_ylabel("Strain")
    plt.show()

task_6cdef()
```

6a) This was achieved experimentally to be L=8.
6b) See figure 9.
6c) The filter delays the signal by 4.5 samples, due to the convolution.
6d+e) See listing 6. 6f) See figure 10.

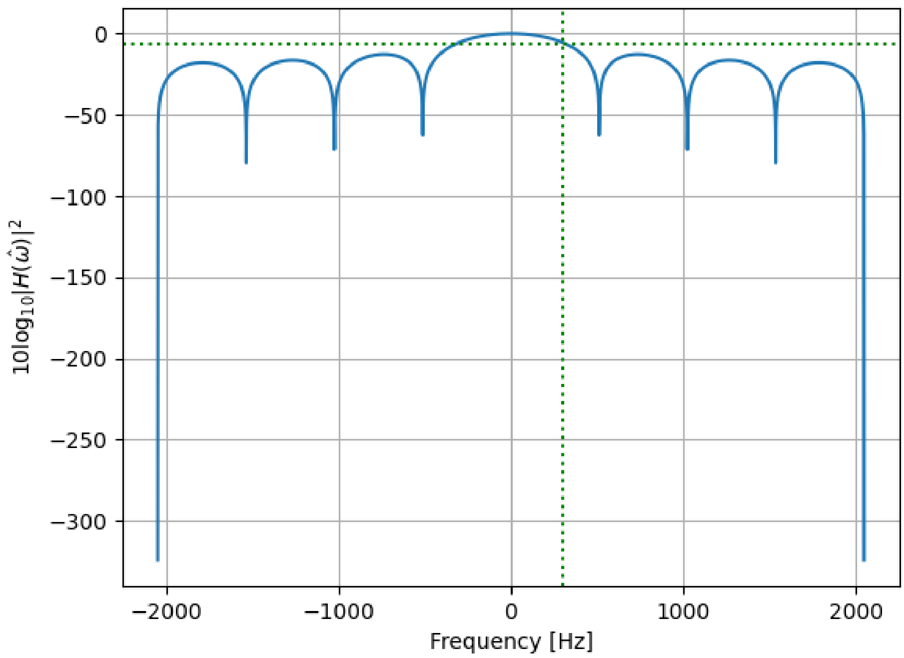

Figure 9: Power spectral response of filter, with L=8. Lines at -6dB and 300Hz.

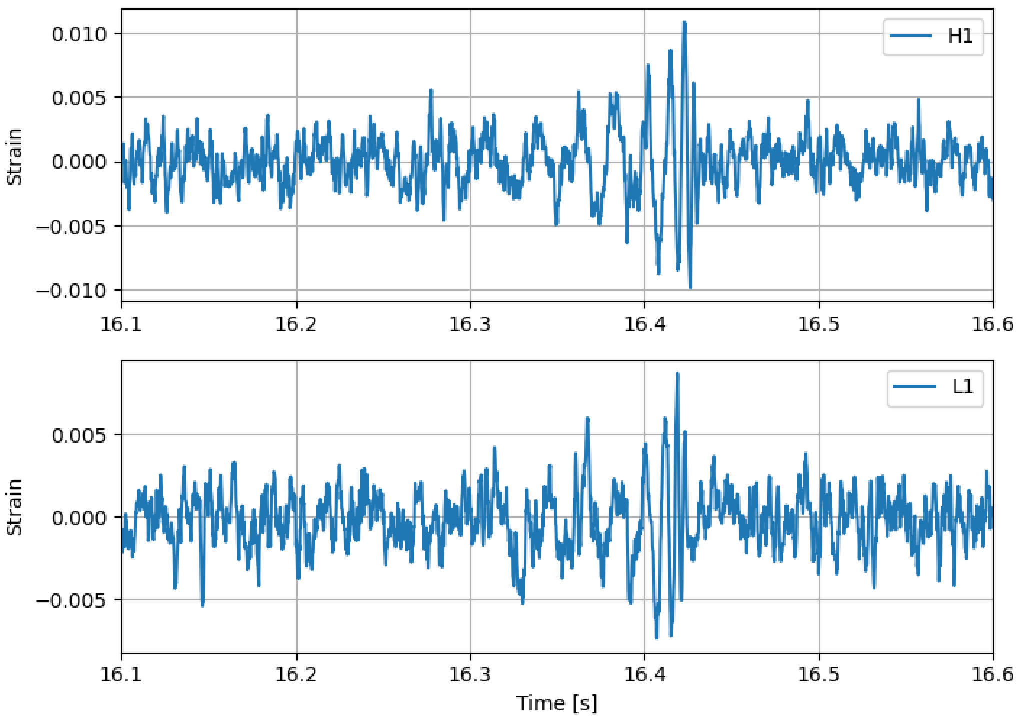

Figure 10: Low pass filtered and whitened H1 and L1 signals.

7 Task 7

Listing 7: Task 7

```python
import matplotlib.pyplot as plt
import numpy as np
import h5py
X_H_FILE, X_L_FILE = "H1_LOSC.hdf5", "L1_LOSC.hdf5"
f_s = 4096

def get_strain(file_name):
    return h5py.File(file_name, "r")["strain/Strain"][()]

# Whiten signal (time-dimension)
def whiten_signal_td(x):
    xw = x * np.hanning(len(x)) # Apply window to signal
    Xw = np.fft.rfft(xw) # FFT of windowed signal
    hH = 1 / np.abs(Xw) # Make filter
    yH = Xw * hH # Apply filter
    return np.fft.irfft(yH)

filter_len = 8
def apply_filter(x, length):
    w = np.ones(length, dtype=np.float_) / length
    return np.convolve(x, w, "valid")

def get_t(filter_len, N, tot_time, f_s):
    n = N - filter_len + 1
    t_start = (filter_len + 1) / (f_s * 2)
    t_end = tot_time - t_start
    return np.linspace(t_start, t_end, n)

# Whiten and filter signals

x_H = get_strain(X_H_FILE)
x_L = get_strain(X_L_FILE)

y_H = whiten_signal_td(x_H)
y_L = whiten_signal_td(x_L)

lopass_H = apply_filter(y_H, filter_len)
lopass_L = apply_filter(y_L, filter_len)

N = len(x_H)
t_L = get_t(filter_len, N, N/4096, f_s)

H_shift = -30
t_H = t_L + H_shift *(1 / f_s)

plt.plot(t_L, lopass_L, alpha=.8)
plt.plot(t_H, -1*lopass_H, alpha=.5)
plt.legend(["L", f"H shifted {H_shift} samples, inverted"])
plt.grid()
plt.xlim(16.1, 16.6)
plt.xlabel("Time [s]")
plt.ylabel("Strain")
plt.show()
```

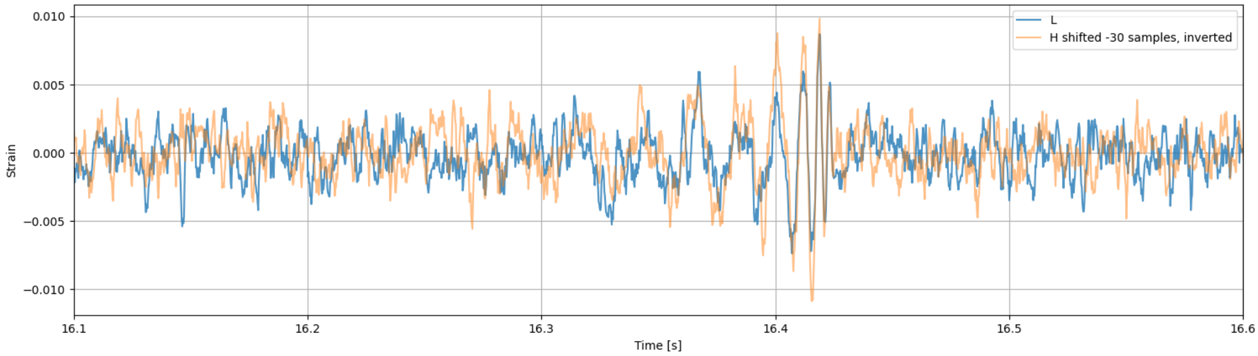

Figure 11: Low-pass filtered and whitened signals superimposed. H1 signal was inverted, and shifted to the left by 30 samples. The value of 30 was achieved experimentally until the peaks aligned.

7b+c+d) The H-signal was inverted, and shifted 30 samples to the left (see figure 11). This corresponds to 30 / 4096 = 0.00732 seconds, which is 7.32 milliseconds. This is in agreement with the gravitational-wave propagation speed, ie. $-10 < \tau < 10ms$.