EE386 Digital Signal Processing Lab **Jul-Dec 2021**

Experiment 6 Report: Digital IIR Filter Design

Author: Harshavardhan D *Email:* harshavardhand.191ee123@nitk.edu.in

Note:
The codes used in this task have been uploaded as a separate .rar file along with the pdf report. Also, all the code has been written using Google Colab and certain pieces of code might be different from when written in Jupyter Notebook due to compatibility issues. Thus it is recommended to run the code on Google Colab for verification. Throughout this document, the value of α used is given by:
α = 1+ mod(123,3) = 1

Problem 1:Butter-worth filter design

(Part1: Designing a Digital Filter of given specifications & Finding its Transfer-Function of the Filter)

(Solution)
In this task we are asked to design a low pass digital Butter worth filter which has a maximum pass-band ripple of -α dB, and an edge frequency of 10 Hz. The filter also should have a minimum stop-band attenuation of 40 dB from a stop-band edge frequency of 20 Hz. Assume a sampling frequency of 720 $\frac{samples}{sec}$.
In order to do this we follow the Bi-linear Transformation method to design digital filters. First off, we have the given specifications of the filter as follows
Sampling Frequency (F_s) = 720 $\frac{samples}{sec}$, This implies T =$\frac{1}{F_s}$
Pass-band ripple gain (δ_p) = -α dB = -1 dB = 0.8125
Pass-band edge frequency ($\Omega_p^{'}$) = ($\Omega_c^{'}$) = 10Hz = $20\pi\frac{rad}{sec}$
Stop-band ripple gain (δ_s) = -40 dB = 0.01 Stop-band edge frequency ($\Omega_s^{'}$) = 20Hz = $40\pi\frac{rad}{sec}$

Steps followed to design the digital filter using the bi-linear transformation technique:

1. Using the given specifications, we find the pre-warping analog frequencies in the following manner
   - Desired discrete time pass-band edge frequency ($\omega_p^{'}$) = ($\frac{\Omega_p^{'}}{F_s}$) = $\frac{\pi}{36}\frac{rad}{sample}$
   - Desired discrete time stop-band edge frequency ($\omega_s^{'}$) = ($\frac{\Omega_s^{'}}{F_s}$) = $\frac{\pi}{18}\frac{rad}{sample}$
   - Pre-Warped Passband edge frequency ($\Omega_p^{''}$) = ($\frac{2}{T}tan(\frac{\omega_p^{'}}{2})$) = 62.87175$\frac{rad}{sec}$
   - Pre-Warped Stopband edge frequency ($\Omega_s^{''}$) = ($\frac{2}{T}tan(\frac{\omega_s^{'}}{2})$) = 125.9836$\frac{rad}{sec}$

2. Using the above analog frequencies we find the $H(s)$ of the analog filter: This is done by
   - We know that for a butter-worth-filter design $\epsilon = \sqrt{\frac{1-\delta_p^2}{\delta_p^2}} = 0.7175$
   - Now in order to find the ratio ($\frac{1}{k} = \frac{\Omega_s}{\Omega_p} = \frac{\Omega_s''}{\Omega_p''}$) we apply the low pass to low pass filter transformation. thus we get $\frac{1}{k} = \frac{\Omega_s}{\Omega_p} = 2$
   - We know that for a butterworth-filter design the order of the filter $N = \frac{log(\frac{1}{\epsilon}\sqrt{\frac{1-\delta_s^2}{\delta_s^2}})}{log(\frac{\Omega_s}{\Omega_p})} = 8$
   - Therefore, the Transfer function $H_{LP}(s) = \frac{1}{(s^2+0.390181s+1)(s^2+1.111140s+1)(s^2+1.663s+1)(s^2+1.961571s+1)}$
     Note: This butter-worth denominator polynomial was taken from this link. Now we apply the low pass to low pass filter transformation:
     s' $= \Omega_p \frac{s}{\Omega_p'}$ where, $\Omega_p = \epsilon^{\frac{1}{N}} = 0.95935$; $s^{'} = \frac{0.95935s}{20\pi}$
     This is done by first using the *buttord()* functiong from the scipy library which take in the the (pre-warped) pass-band and stop-band frequencies of the digital filter, The maximum loss in the passband (dB) & the minimum attenuation in the stopband (dB) and outputs the lowest order for the Butterworth filter which meets specs along with the cut-off frequency of the filter. We then find $H(s')$ using the *butter()* function from the scipy library which takes input as the the order of the filter, the type of filter along with the critical frequencies of the filter in question & outputs the numerator & denominator polynomials of the transfer function of the IIR filter after applying the lowpass to lowpass transformation in our case.

3. Lastly we apply the bilinear transformation to $H_{LP}(s')$ to get $H(z)$ where $s' \rightarrow \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}$. This is done using the *bilinear()* function from the scipy library that takes the input as the numerator and denominator polynomials obtained in the previous step along with the sampling rate mentioned above and outputs the numerator & denominator polynomials of the transformed digital filter transfer function. Thus the final transfer function of the filter is obtained as
   $H(z) = \frac{2.034e^{-11}z^8+1.627e^{-10}z^7+2.034e^{-11}z^8+5.696e^{-10}z^6+1.139e^{-9}z^5+1.424e^{-9}z^4+5.696e^{-10}z^2+1.627^{-10}z+2.034e^{-11}}{(z^8-7.513z^6+24.71z^6-46.47z^5+54.64z^4-41.14z^3-5.215z+0.6145)}$

(Part2: Plotting the Pole zero plot of the digital Filter.)

(Solution)

Now we have to plot the poles and zeros of the transfer function obtained in part 1 i.e., H(z) on the z-plane. This is done by using the *tf2zpk()* from the scipy library. This function takes in the numerator and denominator polynomial coefficients of the transfer function and outputs the zeroes, poles and the system gain of the transfer function.

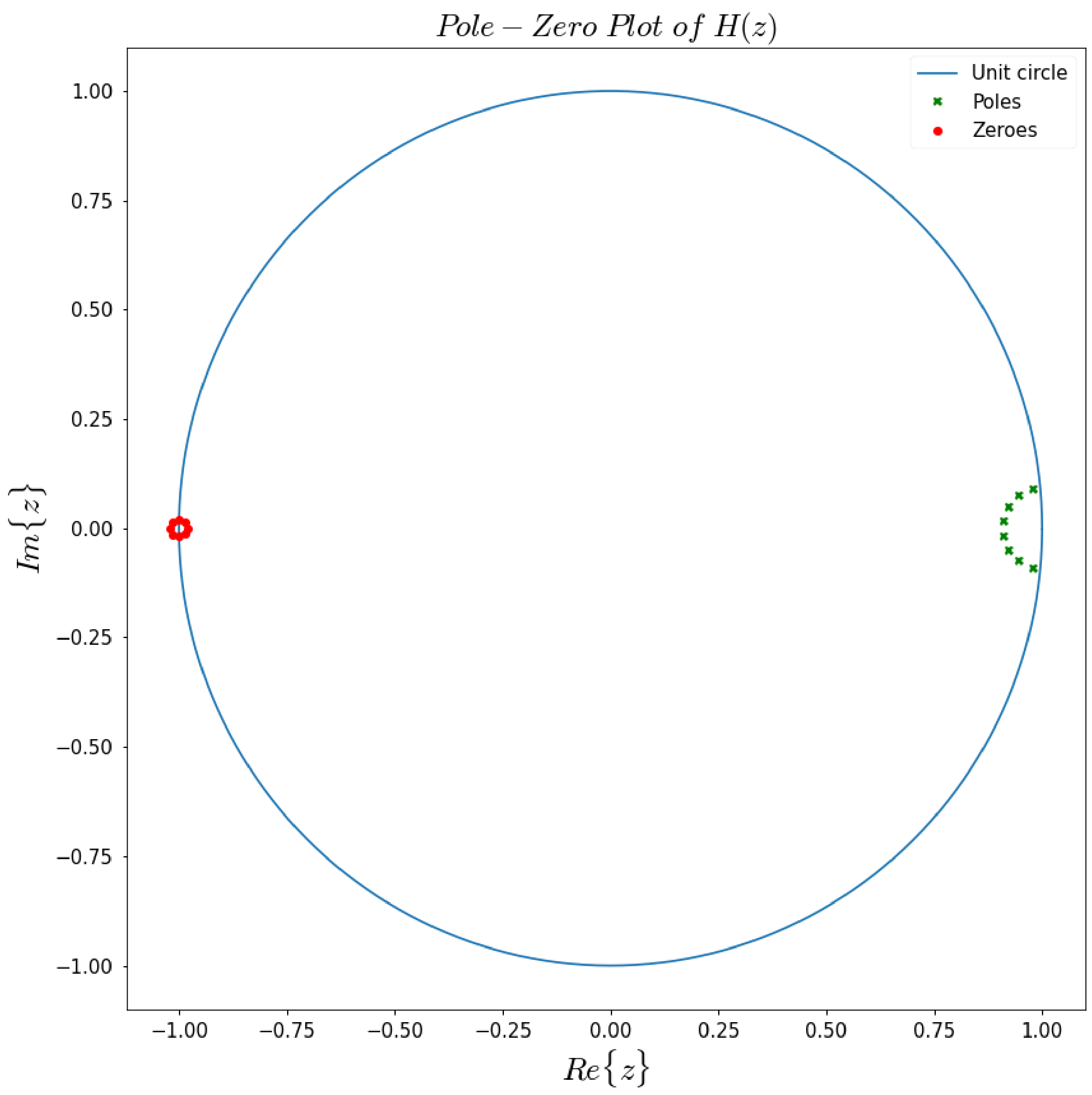

Figure 1: Pole-Zero plot of designed IIR filter

We know that the all stable poles lie inside the unit circle $|z| < 1$. This is because the original Butterworth filter's poles lie on the unit circle in the left half plane and the mapping from s to z, though nonlinear, roughly preserves this arrangement. From Fig.1 we can notices that all 8 poles of $H(z)$ lie inside the unit circle on the z-plane meaning that the system is stable.

(Part3: Plotting the Bode plot of the digital filter)

(Solution)

Here in order to plot the bode plot of the given system defined by $H(z)$ where we take $z = e^{jw}$, we use the *freqz()* function from the scipy library which take in inputs as the numerator and denominator polynomials of the linear filter in question and outputs the frequency response ($H(e^{jw})$), as complex numbers in the form of a numpy array along with the frequencies w at which ($H(e^{jw})$) was computed in ($\frac{radians}{sample}$) which is later normalized to Hz using the the expression $F = \frac{\omega}{2\pi}F_s$. Here, while plotting the bode plot the the magnitude in the dB scale & the phase angles are plotted against the normalized frequency (in Hz) on a semi-log plot.

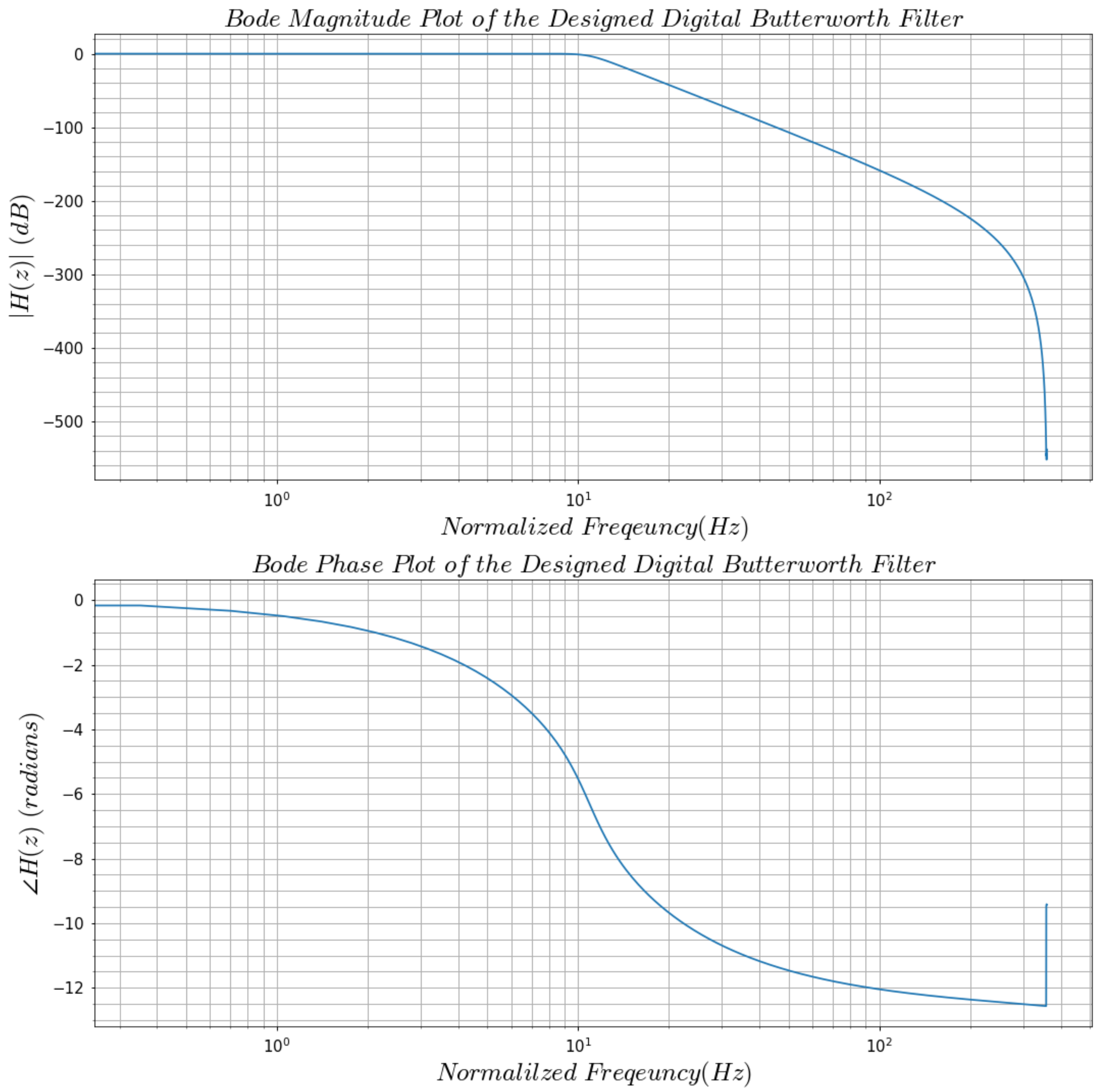

Figure 2: Bode plot of designed IIR filter

We notice that the magnitude plot has a DC gain 0dB and the cutoff frequency is at around 10 Hz. Due to the nonlinear frequency mapping, the roll-off does not have a constant slope due to which all high frequencies get compressed into a tiny range (warping). Since we compensated using prewarping, if we zoom in, we will see that the 1dB and 40dB frequencies occur at approximately 10Hz and 20Hz.

(Part4: Plotting the Impulse & Step Response of the designed low-pass Butter-worth Filter)

(Solution)

In this task we are asked to plot the impulse and step responses on the system we designed (i.e., the Low-pass Butter-worth Filter). This is done by first defining an input sequence (i.e., the impulse and step inputs in the form of a numpy array whose data-points occur at intervals of frequency same as sampling frequency (F_s)) and the feeding it to the *lfilter()* function from the scipy library that takes in the numerator & denominator coeffcient 1-D vectors of the transfer function of the system in question (i.e. $H(z)$) along with an N-dimensional input array (i.e., the impulse and step inputs we just defined) as inputs and returns the output of the digital filter by filtering the input data sequence along one-dimension with the IIR filter we designed. This output is plotted for a duration of 1 sec as seen in Fig.3.

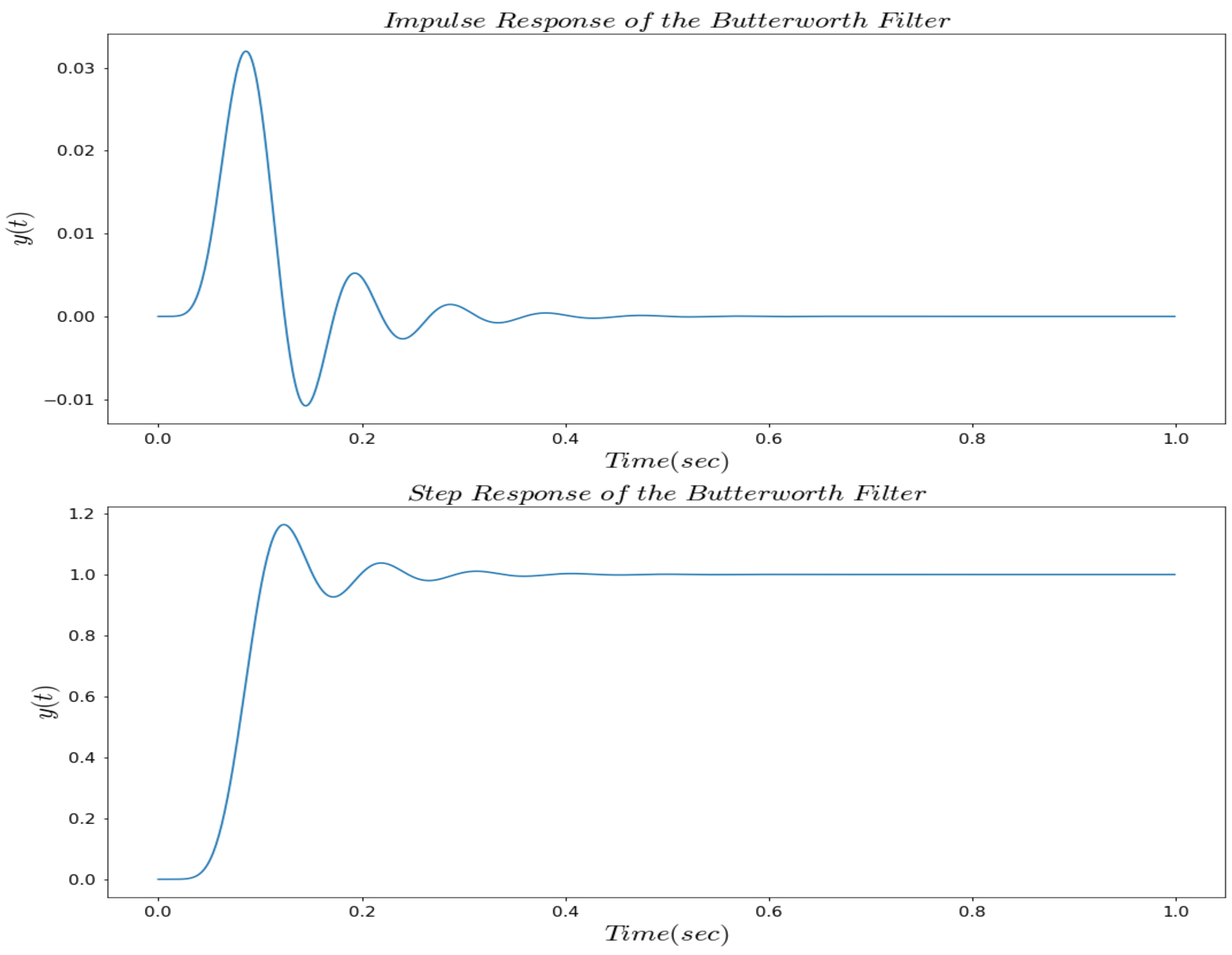

Figure 3: Bode plot of designed IIR filter

A comparison of the impulse and step responses of the Butter-worth & the Chebyshev filter is done in Problem 4.

Problem 2: Filtering

(Part1: Designing a Digital Filter of given specifications & Finding its Transfer-Function of the Filter)

(Solution)

Here, we need to filter the given signal with unknown frequency components using the IIR Butter-worth filter we designed in Problem 1. First, the unknown signal contained in *ECG_Data*.txt is read into a numpy array using the **np.loadtxt()** function from the numpy library. Upon investigation, it was found that the signal is of length 6884 samples. This ECG signal was then filtered by passing it through the *lfilter()* function from the scipy library by treating it as an 1-D input sequence. The original ECG signal along with the filtered ECG signal has been plotted against time in seconds after renomalization as seen in Fig.4.

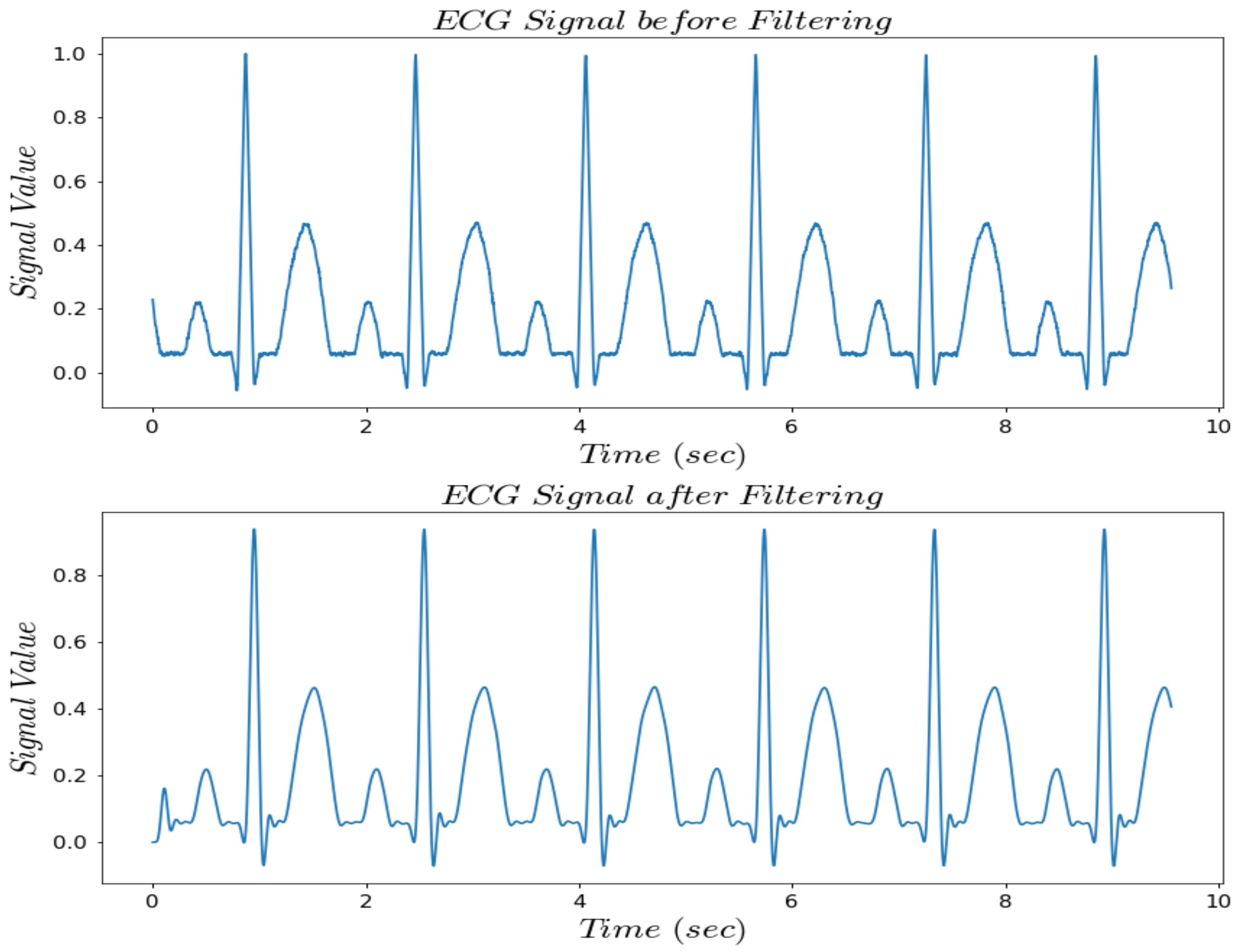

Figure 4: Plots of the ECG signal before and after filtering

From the figure above we can notice that the original ECG signal contains many small, high-frequency variations, but upon filtering we observe no rapidly varying or high frequency components. This because the high frequency components have been attenuated by the low pass Butter-worth filter that we passed the signal through.

(Part2: Verification via FFT)

(Solution)

In order to verify this the FFT of the original ECG signal and the filtered signal were plotted using the *fft()* function from the scipy library as seen in Fig.5. This signal was normalised by dividing by the number of samples and was plotted against re-normalised frequency in Hz, using a sampling frequency of 720Hz. The Frequency axis has been limited to 500 samples to understand the plot better.

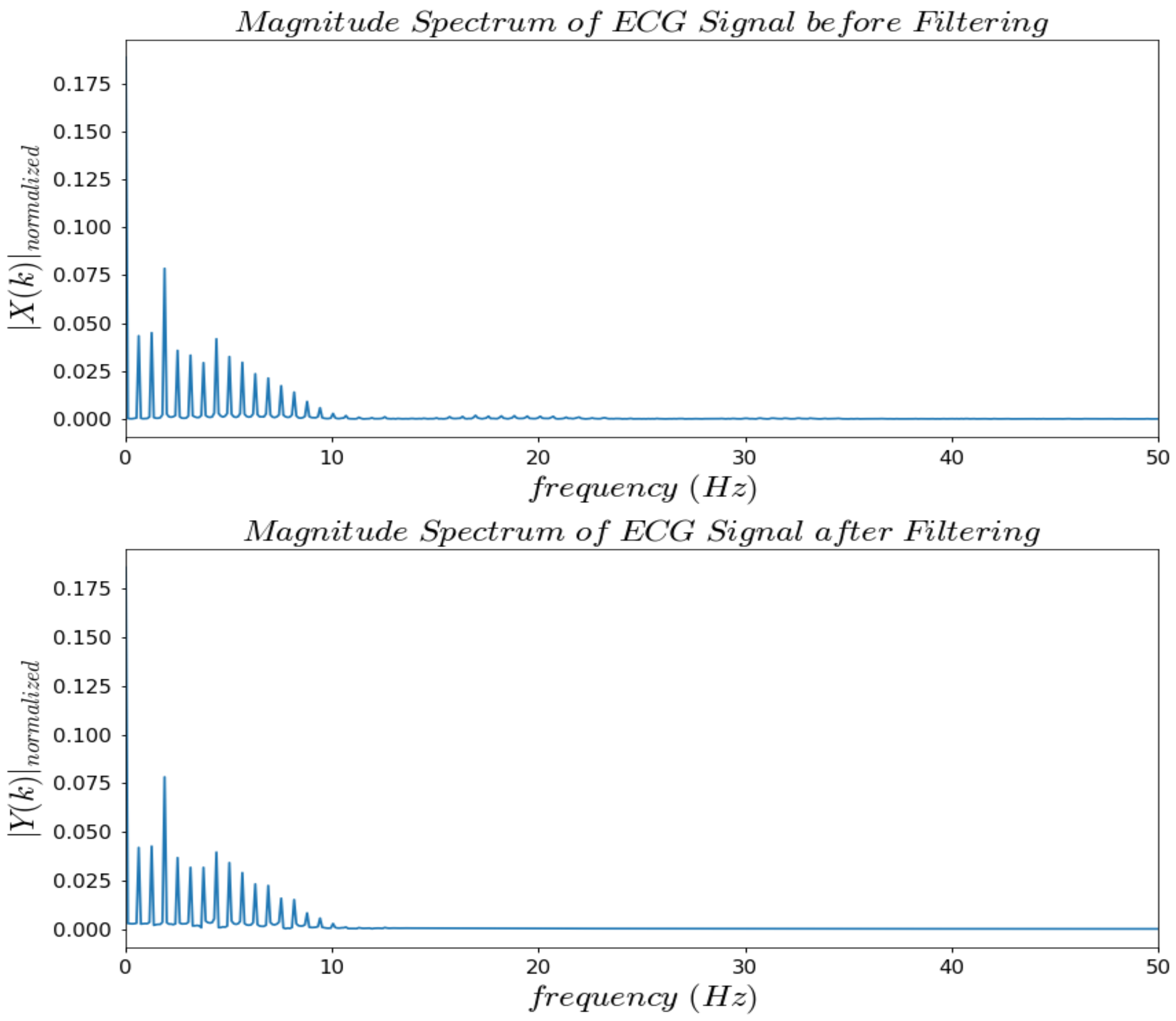

Figure 5: FFT Magnitude Spectrum Plots of the ECG signal before and after filtering

From the above figure we can see that the original signal consisted of certain high frequencies at around 20Hz, but in the filtered signal we notice that they have been remove/filtered out giving us a smooth waveform.

Problem 3: Filtering — Time-Frequency Analysis

(Part1: Plotting Spectrogram of given Instrument Signal)

(Solution)

Here, the *instru*1*.wav* file is first read using the *wavefile.read()* function and its sampling frequency and the signal array were stored. The signal values were normalized by dividing by 32767.0 to account for the fact that data is stored in .wav file in 16-bit format. The above signal array, sampling frequency, window length (600), overlap number (500), and window (Hamming), were used as parameter to plot the spectrogram using the *specgram()* function as seen in Fig.6.

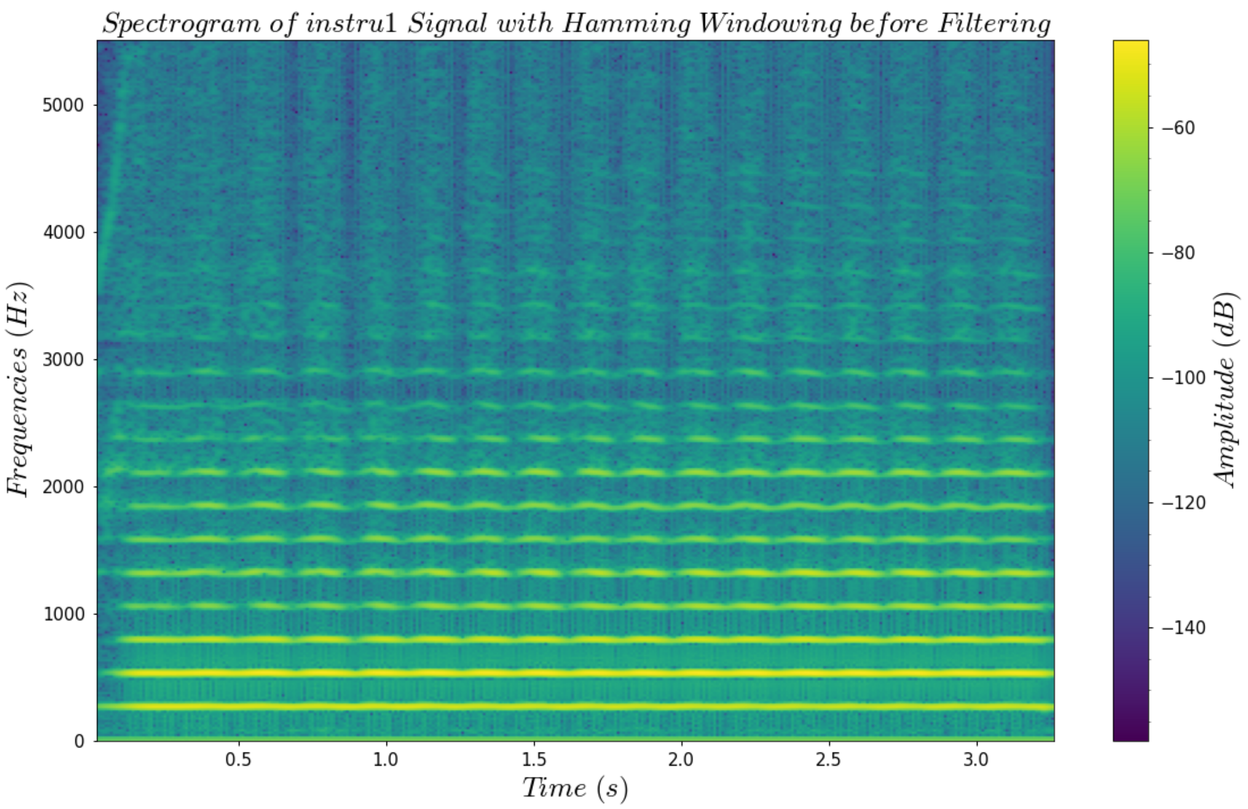

Figure 6: Spectrogram plot of Instrument Signal

Now in order to extract the fundamental frequency we use the *find_ff()* function defined in Experiment 3. The fundamental frequency obtained for the signal with prominence = 0:6, is 262.47Hz.

Now from the Spectrogram (Fig.6) we can notice that the bottom-most line that represents the fundamental frequency appears around $\frac{1}{4}$ of the first y-axis division from 0-1000 Hz which gives

us an approximate value of 250Hz. The filter needs to be designed as a bandpass filter. Thus the specifications are chosen as:
3dB attenuation i.e., $\delta_p = -3dB$ at frequencies $\Omega_l^{'} = 400\pi\frac{rad}{sec}$ $\Omega_u^{'} = 600\pi\frac{rad}{sec}$
40dB attenuation i.e., $\delta_s = -40dB$ at frequencies $\Omega_{s1}^{'} = 200\pi\frac{rad}{sec}$ $\Omega_{s2}^{'} = 800\pi\frac{rad}{sec}$
. The design of the Butter-worth bandpass digital filter is carried out by following the procedure similar to that mentioned in Problem 1 using the bilinear transform. The sampling frequency for the filter is set at $F_s = 11kHz$. Then the signal is convolved with the filter using the *lfilter()* function from the scipy library . The spectrogram of this new filtered signal is again plotted using the using the *specgram()* function as seen in Fig.7.

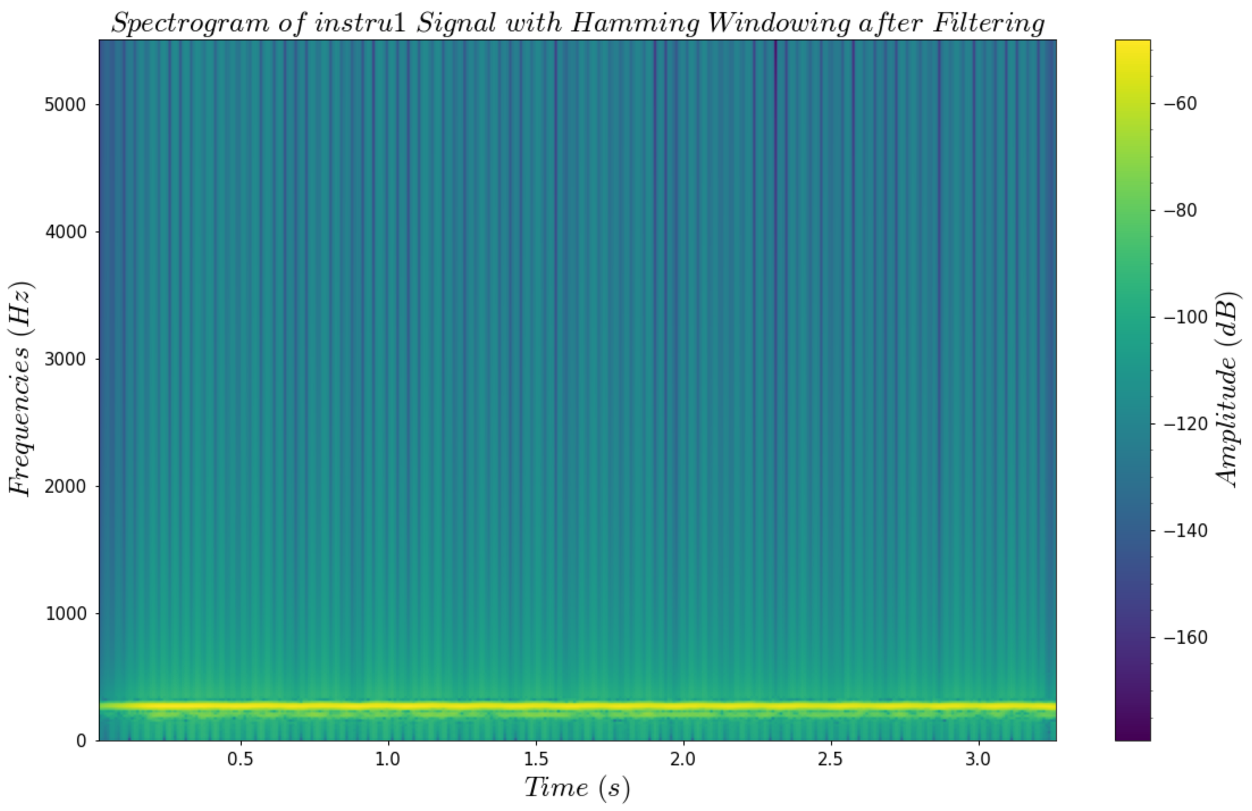

Figure 7: Spectrogram plot of Filtered Instrument Signal

Here we notice that all the higher harmonics as well as the Dc component have been remove/attenuated as they are no more parallel lines in the spectrogram when compare to fig.6. The only remaining line in the spectrogram corresponds to the fundamental frequency. We use the $find\_ff()$ function to verify this, and obtained the fundamental to be around 262Hz. The filtered signal was then written to a .wav file, using the wavfile.write() function with the appropriate sampling frequency and scaling, and the track which is available in the gitHub repository under the name $instru1\_filtered.wav$. Upon playing this we notice that only the fundamental frequency component of the signal can be heard.

Problem 4: Chebyshev filter design

(Part1: Designing a Digital Filter of given specification)

(Solution)

In case of the Chebyshev Filter design, the procedure is similar to the case of Butterworth Filter design as seen in Problem 1 until the point where we arrive at the selectivity k of the said filter. Here we find the order of the filter using the expression $N = \frac{cosh^{-1}(\frac{1}{\epsilon}\sqrt{\frac{1-\delta_s^2}{\delta_s^2}})}{cosh^{-1}(\frac{\Omega_s}{\Omega_p})} = 5$
In terms of the code, here we use the scipy function $cheb1ord()$, which calculates order required and cutoff frequency for a Chebyshev type-1 filter instead of $buttord()$ function for Butterworth. Upon further investigation we find that the required order of the Chebyshev filter is obtained as calculated above. This means that the Chebyshev filter will have a steeper slope that the Butterworth filter in the transition band. We then use the function $cheby1()$ function from the scipy library that is analogous to the $butter()$ function we used for the case of the butterworth filter design and pass in the order, cutoff frequency, filter type (i.e., lowpass in our case) which then outputs the numerator & denominator polynomial after performing the lowpass to lwpass transformation. These numerator and denominator polynomial coefficients obtained are again passed to the bilinear mapping function (i.e., $bilinear()$) to get the numerator and denominator coefficient vectors for H(z) that can be represented in the transfer function form.

(Part2: Plotting the Bode plot of the digital filter)

(Solution)

The we now generate the Bode plots were for the Chebyshev filter, which in a similar manner to the Probelm 1 as seen in Fig.8.

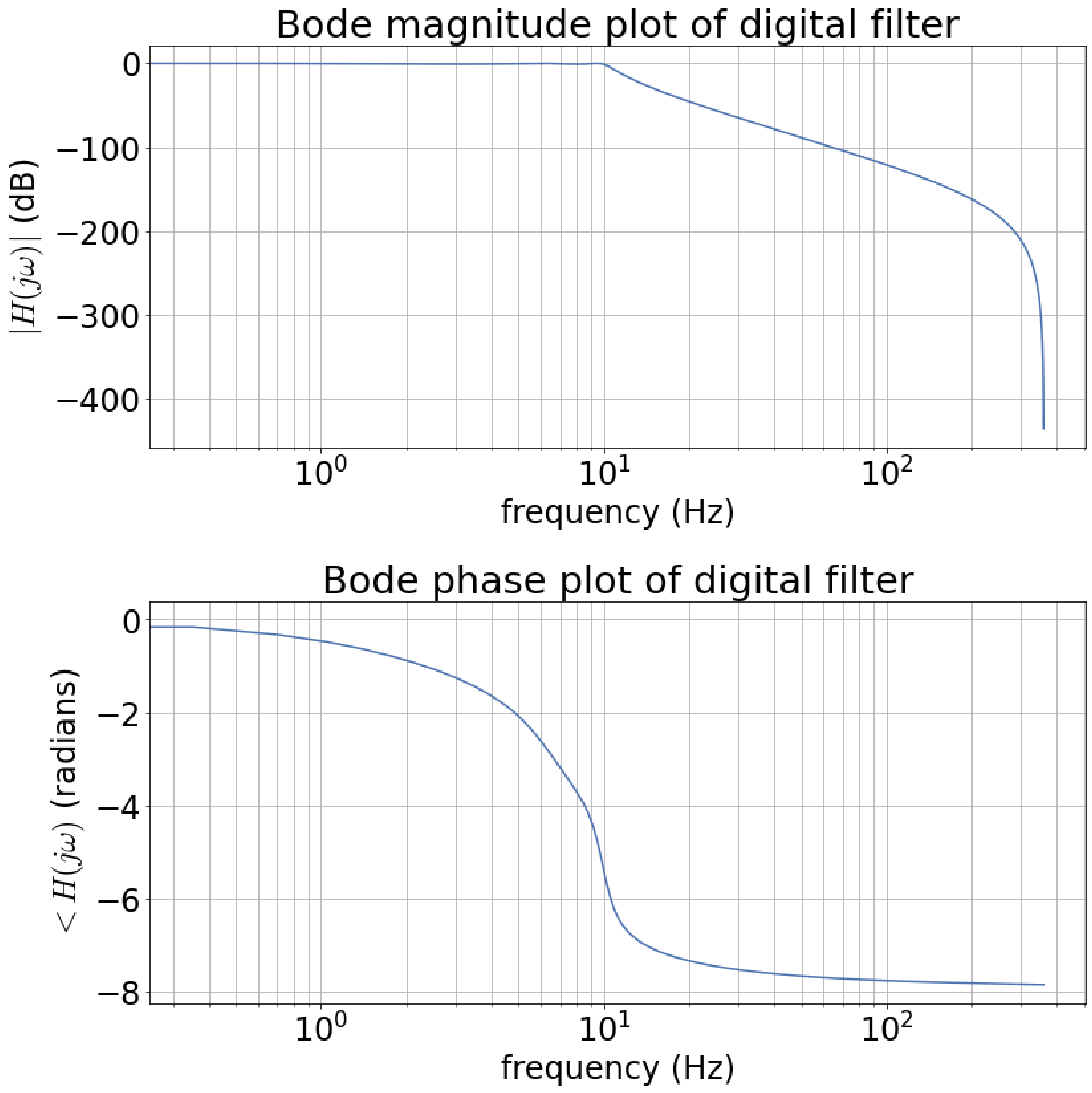

Figure 8: Bode plot of designed ChebyChev Filter

(Part4: Plotting the Impulse & Step Response of the designed low-pass Butterworth & Chebyshev Filters)

(Solution)

We then generate the impulse and step responses for both the filters, Butterworth and Cheby-1, for a duration of 1 sec ina manner similar to that mentioned in problem 1 using the $lfilter()$ function as seen in Fig.9.

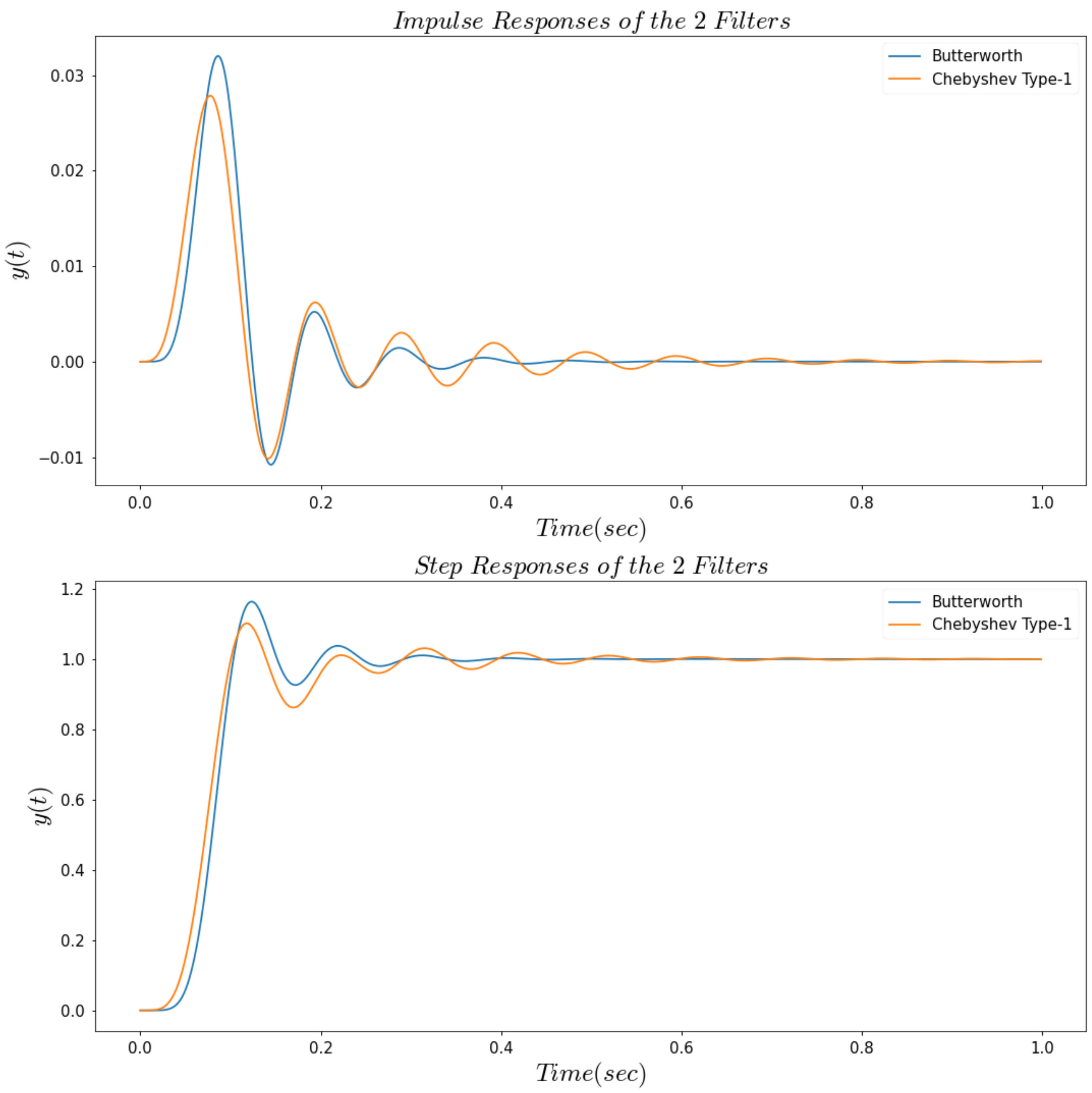

Figure 9: Impulse & Step responses of the 2 filters

If we compare the Chebyshev and Butterworth filter responses, we can see in both cases that there is an overshoot, but the overshoot in case of the Butterworth filter is higher meaning that in this aspect the Chebyshev filter performs better. Furthermore if we compare the settling times we notice that the damped oscillations in the Chebyshev filter take a longer time to die down than those of the Butterworth filter, meaning the Butterworth filter has a smaller settling time meaning that the Butterworth filter is better in this aspect.

A Code Repositories

The codes to reproduce the results can be found in the GitHub repository
`https://github.com/Harsha1569/DSP_Lab.git`.
Links to documentation of libraries used

- `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.buttord.html`
- `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html`
- `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.bilinear.html`
- `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.tf2zpk.html`
- `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.freqz.html`
- `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.lfilter.html`