Ain Shams University

Digital Signal Processing - ECE-451s

Digital Filter Design Using Matlab

Team 27

Ahmad Ayman Ibrahim	2200563
Ziad Alaa Kassem	2200278
Pavly Osama Zarif	2200509
Hazem Youssef Mahmoud	2200183

Contents

Introduction

In this project, we used MATLAB to design digital filters and analyze them to study their behavior in both time and frequency domains. A real audio signal was used to demonstrate the effects of echo distortion, equalizing and filtering. The goal was to compare four different IIR filter types and understand the differences in sharpness, phase changes and distortion.

Functions

General Requirements

The *Sec3Tasks* function plots all the requirements needed in section 3, starting from the pole-zero plot all the way to the filter's impulse response. It is then applied as needed in the main code. The function is divided into segments and explained below.

It also important to give credit to AI for providing the idea of dividing the plots each into their own tabs for easier access and making the results visually pleasing.

```
function Sec3Tasks_2_0(String_title, num, den, num_zplane, den_zplane, Freq_Vector, F_sampling, N_Samples,
imp_len ,show_zoom, sosgain)
    % show_zoom:
    %   - true: shows zoomed plots
    %   - false: shows only full-range plots

    %%---------------------------------- 1: Pole-Zero Plot (Separate Window) ------------
    figure('Name', ['Pole-Zero Plot ' String_title], 'NumberTitle','off');
    zplane(num_zplane, den_zplane);
    title(['Pole-Zero Plot ' String_title]);
    grid on;

    %%----------------------------------- Create Main Tabbed Figure ----------------------
    f = figure('Name', ['Filter Analysis ' String_title], 'NumberTitle', 'off', ...
           'Position',[100 50 1400 900]);
    tg = uitabgroup(f);
    tab2 = uitab(tg,'Title','Magnitude');
    tab3 = uitab(tg,'Title','Phase');
    tab4 = uitab(tg,'Title','Group Delay');
    tab5 = uitab(tg,'Title','Impulse Response');
```

Here we define the function that takes 11 arguments: The plot title/name, to properly differentiate between plots, *num* and *den* are second order sections (SOS) of filter coefficients and *num_zplane* and *den_zplane* are your original coefficients. Both are passed into the function as *zplane* does not handle SOS coefficients. The frequency vector is also passed as an argument under *Freq_Vector*, so is the sampling frequency and number of samples. *imp_len* is how many samples are shown in the impulse response (to ignore the extra information). *show_zoom* is an argument that plots both the original and zoomed-in plot if set to true. *sosgain* is the second-order section gain to normalize the filter magnitude.

```
%%---------------------------------- Compute Frequency Data -----------------------------
    if size(num, 2) == 6 %detects if num is in SOS format, which is a matrix with 6 columns
        % SOS format - only needs 2 arguments
        H = freqz(num, N_Samples, F_sampling, 'whole');  % without whole: 0->fs/2 , with whole 0->fs
    else
        % Normal (b,a) format
        H = freqz(num, den, N_Samples, F_sampling, 'whole');
    end
    H_shift = fftshift(H);
    x_mag_dB = 20 * log10(abs(H_shift) + eps);% EPS is very small constant to avoid log 0
    x_phase  = angle(H_shift);
    phase_unwrapped = unwrap(x_phase);        % removes discontinuities in phase at +-Pi


    % Group Delay
    % Computing group delay using logarithmic identity

    w = linspace(-pi, pi, N_Samples).';   %define freq vector from -pi to pi

    Hc = H_shift(:);                      %transforms vector into column vector to match


    dH_dw = gradient(Hc, w);            %computes derivative of H(e^jw) with respect to w

    % group delay
    gd = -imag(dH_dw ./ Hc); % H'/H is the derivative of ln(H(e^jw)), taking the    imaginary part of it
represents derivative of phase

    % masks zeros and deep stopband
    mag_thresh = 1e-6;
    gd(abs(Hc) < mag_thresh) = NaN;
```

This part computes the necessary calculations for magnitude, phase and group delay plots.

Second-order sections (SOS) were used to improve numerical stability when analyzing high-order IIR filters (Task 2&3). Implementing the filters as cascaded second-order sections reduces coefficient sensitivity and prevents numerical errors in the computation of frequency response, phase, and group delay.

First the code detects whether the given coefficients are in SOS format (SOS are always matrices with 6 columns) and computes accordingly. The denominator coefficients are ignored in the *freqz* function for second order sections, so it's simply not passed.

The frequency response *H* is then shifted to match our frequency axis, and transformed into magnitude and phase responses.

Group delay is computed via the logarithmic derivative of the frequency response instead of the *grpdelay* function and instead of finding the gradient of the unwrapped phase where:

$$\tau_g = -Imag(\frac{1}{H(e^{jw})}\frac{dH(e^{jw})}{dw})$$

$$\text{as } \ln(H) = \ln|H| + j\angle \text{H}$$

```
%%---------------------------------- 2: Magnitude TAB ----------------------------------

    p2 = uipanel('Parent',tab2);

    if show_zoom
        % Two subplots: full range + zoomed
        ax1 = subplot(2,1,1,'Parent',p2);
        plot(ax1, Freq_Vector, x_mag_dB);
        xlabel(ax1,'Frequency (Hz)');
        ylabel(ax1,'Magnitude (dB)');
        title(ax1,['Magnitude Response ' String_title]);
        grid(ax1,'on');
        xlim(ax1,[-F_sampling/2, F_sampling/2]);

        ax2 = subplot(2,1,2,'Parent',p2);
        plot(ax2, Freq_Vector, x_mag_dB);
        xlabel(ax2,'Frequency (Hz)');
        ylabel(ax2,'Magnitude (dB)');
        title(ax2,['Magnitude Response (Zoomed) ' String_title]);
        grid(ax2,'on');
        xlim(ax2,[-100,100]);
    else
        % Single plot: full range only
        ax1 = axes('Parent',p2);
        plot(ax1, Freq_Vector, x_mag_dB);
        xlabel(ax1,'Frequency (Hz)');
        ylabel(ax1,'Magnitude (dB)');
        title(ax1,['Magnitude Response ' String_title]);
        grid(ax1,'on');
        xlim(ax1,[-F_sampling/2, F_sampling/2]);
    end

    %%---------------------------------- 3: Phase TAB ----------------------------------
    p3 = uipanel('Parent',tab3);

    if show_zoom
        % Two subplots: full range + zoomed
        ax1 = subplot(2,1,1,'Parent',p3);
        plot(ax1, Freq_Vector, phase_unwrapped);
        xlabel(ax1,'Frequency (Hz)');
        ylabel(ax1,'Phase (rad)');
        title(ax1,['Phase Response ' String_title]);
        grid(ax1,'on');
        xlim(ax1,[-F_sampling/2, F_sampling/2]);

        ax2 = subplot(2,1,2,'Parent',p3);
        plot(ax2, Freq_Vector, phase_unwrapped);
        xlabel(ax2,'Frequency (Hz)');
        ylabel(ax2,'Phase (rad)');
        title(ax2,['Phase Response (Zoomed) ' String_title]);
        grid(ax2,'on');
        xlim(ax2,[-100,100]);
    else
        % Single plot: full range only
        ax1 = axes('Parent',p3);
        plot(ax1, Freq_Vector, phase_unwrapped);
        xlabel(ax1,'Frequency (Hz)');
        ylabel(ax1,'Phase (rad)');
        title(ax1,['Phase Response ' String_title]);
        grid(ax1,'on');
        xlim(ax1,[-F_sampling/2, F_sampling/2]);
    end
    %%---------------------------------- 4: Group Delay TAB -----------------------------
    p4 = uipanel('Parent',tab4);

    if show_zoom
        % Two subplots: full range + zoomed
        ax1 = subplot(2,1,1,'Parent',p4);
        plot(ax1, w, gd);
        xlabel(ax1,'\omega (rad/sample)');
        ylabel(ax1,'Group Delay (samples)');
        title(ax1,['Group Delay ' String_title]);
        grid(ax1,'on');
        xlim(ax1,[-pi,pi]);

        ax2 = subplot(2,1,2,'Parent',p4);
        plot(ax2, w, gd);
        xlabel(ax2,'\omega (rad/sample)');
        ylabel(ax2,'Group Delay (samples)');
        title(ax2,['Group Delay (Zoomed) ' String_title]);
        grid(ax2,'on');
        xlim(ax2,[-0.01,0.01]);
    else
        % Single plot: full range only
        ax1 = axes('Parent',p4);
        plot(ax1, w, gd);
        xlabel(ax1,'\omega (rad/sample)');
        ylabel(ax1,'Group Delay (samples)');
        title(ax1,['Group Delay ' String_title]);
        grid(ax1,'on');
        xlim(ax1,[-pi,pi]);
    end

    %%---------------------------------- 5: Impulse Response TAB ------------------------
    if size(num, 2) == 6  % SOS format
        [h_imp, n_imp] = impz(num, imp_len);
    else  % Normal (b, a) format
        [h_imp, n_imp] = impz(num, den, imp_len);
    end

    p5 = uipanel('Parent',tab5);
    ax = axes('Parent',p5);
    stem(ax, n_imp, h_imp, 'filled', 'MarkerSize', 2);
    xlabel(ax,'n (samples)');
    ylabel(ax,'h[n]');
    title(ax,['Impulse Response ' String_title]);
    grid(ax,'on');
end
```

The above code is for plotting inside each labelled tab. Each with their respective axis according to the requirements, and impulse response computed via *impz* function.

This outputs a single window with multiple tabs for easy access and readability.

Mean Square Error

A function to calculate the mean square error between two given signals (output being y and input being x)

```
function mse = calc_mse(y, x)


    if length(y) ~= length(x) % lengths must be equal
        error('Both vectors must be same lengths');
    end

    mse = (1/(length(y))) * sum((y - x).^2);

end
```

Energy Loss Percentage

This function calculates the percentage of energy lost after applying a signal to a filter.

```
function loss_percent = calc_energylost(x, y)

    L = max(length(x), length(y)); % which of the two is bigger

    % fill shorter vector with zeroes
    x(end+1:L) = 0; % extends X to length L the rest being zeroes
    y(end+1:L) = 0; % same if Y is the shorter one

    % Compute energies
    Ex = sum(abs(x).^2); % x is original signal before filtering
    Ey = sum(abs(y).^2); % y is signal AFTER filtering

    % Energy loss percentage
    loss_percent = (1 - Ey/Ex) * 100;

end
```

The shorter vector of two signals is zero-padded to avoid errors in the calculation.

Task 0

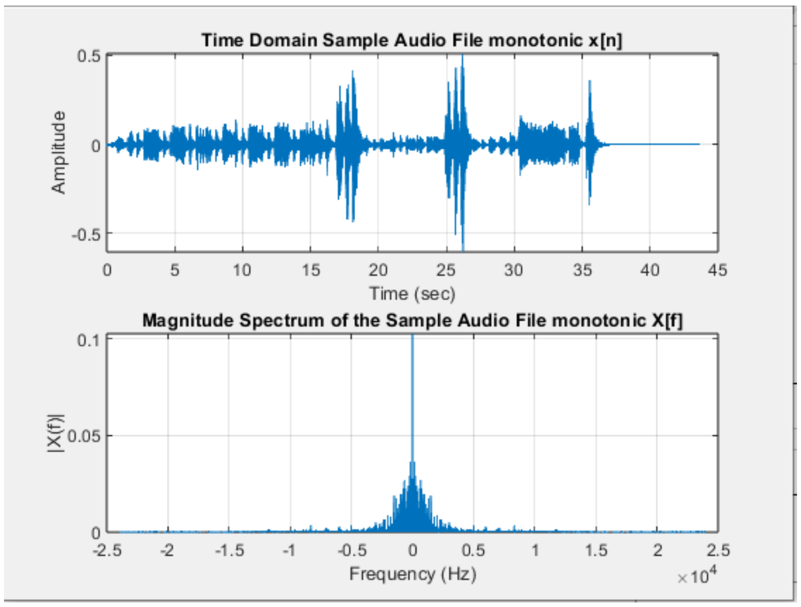
The audio signal was first imported using the MATLAB *audioread* function, which also provides the sampling frequency *Fs*. Based on this sampling frequency, the corresponding time and frequency axes were defined. The signal was then plotted in the time domain.

It was observed that the audio signal is stereo, as it consists of two channels. To simplify the processing, the stereo signal was converted to a monophonic signal by taking the mean of the two channels. This conversion does not significantly affect the audio content and facilitates further analysis. The resulting monophonic signal *x_mono* was then plotted in the time domain.

The frequency-domain representation of the signal was obtained using the Fast Fourier Transform (FFT). The spectrum was centered around zero frequency using the *fftshift* function, and the magnitude spectrum was plotted.

Finally, the signal energy was computed in both the time domain and the frequency domain by summing the squared magnitude of the signal. The results (displayed in fig. 1) show that the energy in both domains is equal, which verifies Parseval's theorem.

```matlab
fprintf('\n=== TASK 0: Audio File ===\n\n');
%Reading Sound File & Defining main parameters:
[x, Fs] = audioread("sample_audio_file.wav");   % returns x: Audio signal samples , Fs: Sampling frequency
(Found in file prop.)
N = length(x);
df= Fs/N;
ts=1/Fs;
t = (0:N-1) / Fs;                   % time axis [nts]
f = (-Fs/2 : df : (Fs/2)-df);       % -Fs/2 < f < Fs/2

%{
figure;
plot(t, x);
xlabel("Time (sec)");
ylabel("Amplitude");
title("Time Domain Sample Audio File x[n]");   % the plot will have 2 colors as the audio file is stereo not
mono -> 2 vectors
grid on;
%}

%As we saw the plot had 2 colors -> audio is stereo
%Converting the audio to mono -> to handle it as 1 vector signal samples
x_mono = mean(x, 2);


X = fft(x_mono);                % compute FFT
X_shifted = fftshift(X)*ts;    % center 0 frequency

figure;

subplot(2,1,1);
plot(t, x_mono);
xlabel("Time (sec)");
ylabel("Amplitude");
title("Time Domain Sample Audio File monotonic x[n]");
grid on;

subplot(2,1,2);
plot(f, abs(X_shifted));
xlabel("Frequency (Hz)");
ylabel("|X(f)|");
title("Magnitude Spectrum of the Sample Audio File monotonic X[f]");
grid on;

Energy_time = sum(abs(x_mono).^2)*ts

Energy_freq = sum(abs(X_shifted).^2)*df
```



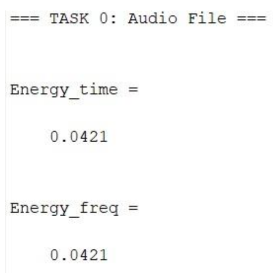

```
=== TASK 0: Audio File ===

Energy_time =

    0.0421

Energy_freq =

    0.0421
```

Figure 1

Task 1

Part 1

In this part, we construct a digital echo system whose transfer function is $H(z) = 1 + 0.9z^{-1000} + 0.8z^{-2000} + 0.7z^{-3000}$.

We derive the LCCDE for the transfer function by hand as shown in figure 2. Using the given coefficients b_k, we first construct the numerator of the system by creating a row vector of length 3001, where all coefficients are initially set to zero. The nonzero coefficients are then assigned at sample indices $n = 1,1001,2001,$ and 3001, corresponding to the required delays.

Since the system is an FIR system, the denominator is equal to 1. Finally, the previously explained function (*Sec3Tasks _2_1*) is used to plot all the required results.

```
%% --------------------- Task 1 -----------------
fprintf('\n=== TASK 1: Digital Echo System ===\n\n');

% Given parameters
b_k = [1, 0.9, 0.8, 0.7];
D = 1000;

% Build FIR numerator: H(z)=1 + 0.9 z^-1000 + 0.8 z^-2000 + 0.7 z^-3000
num_H = zeros(1, 3*D + 1);
num_H(1)          = b_k(1);
num_H(D+1)        = b_k(2);
num_H(2*D+1)      = b_k(3);
num_H(3*D+1)      = b_k(4);

den_H = 1;   % FIR filter denominator

fprintf("Generating plots for Echo System H(z)...\n");
Sec3Tasks_2_1(' - Echo System H(z)', num_H, den_H, num_H, den_H, f, Fs, N, 3500 ,true, 1);
```

`

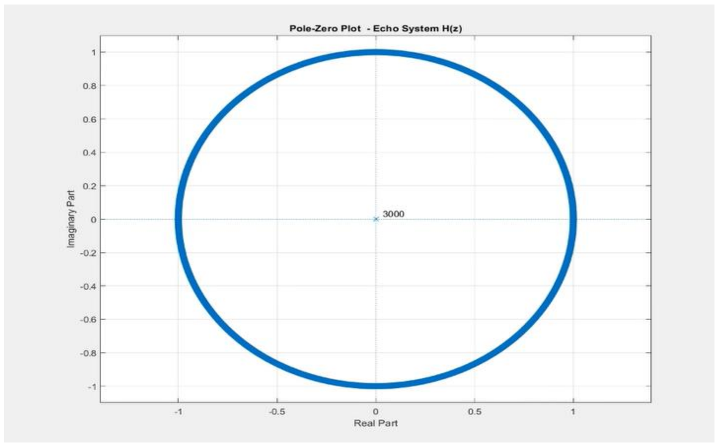
$$H[z] = 1 + 0{,}9\,z^{-1000} + 0{,}8\,z^{-2000} + 0{,}7\,z^{-3000}$$

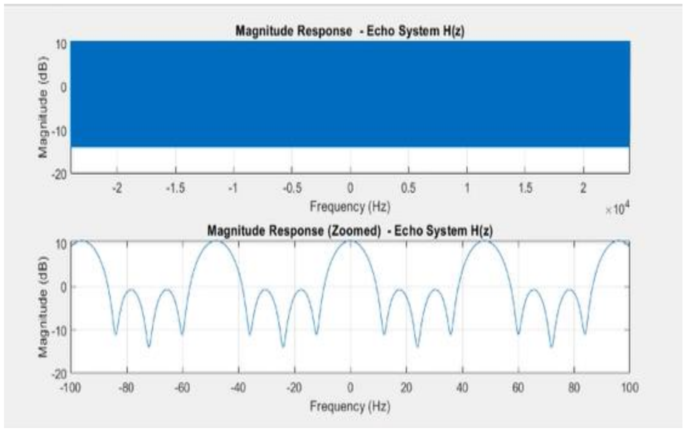
$$\frac{y[z]}{x[z]} = 1 + 0{,}9\,z^{-1000} + 0{,}8\,z^{-2000} + 0{,}7\,z^{-3000}$$

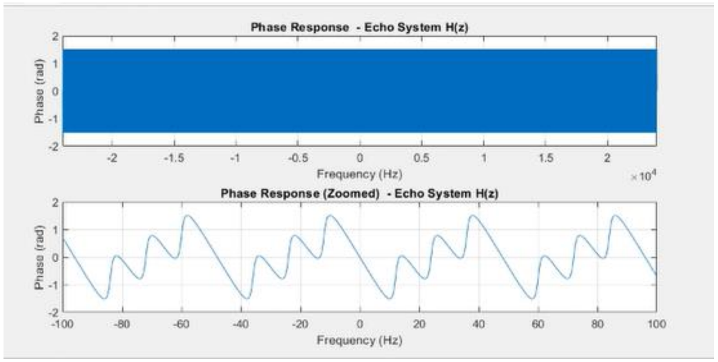
$$y[z] = x[z]\left[1 + 0{,}9\,z^{-1000} + 0{,}8\,z^{-2000} + 0{,}7\,z^{-3000}\right]$$

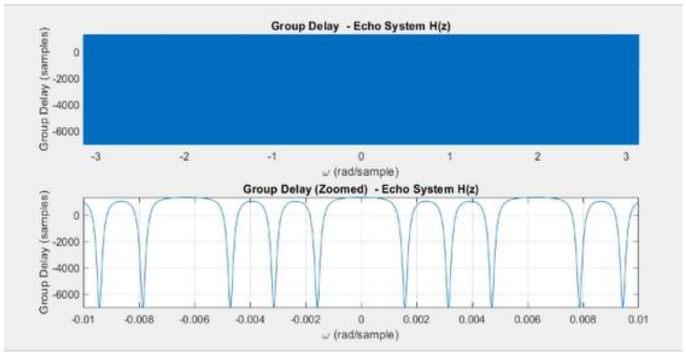
$$\boxed{y[n] = x[n] + 0{,}9\,x[n-1000] + 0{,}8\,x[n-2000] + 0{,}7\,x[n-3000]}$$

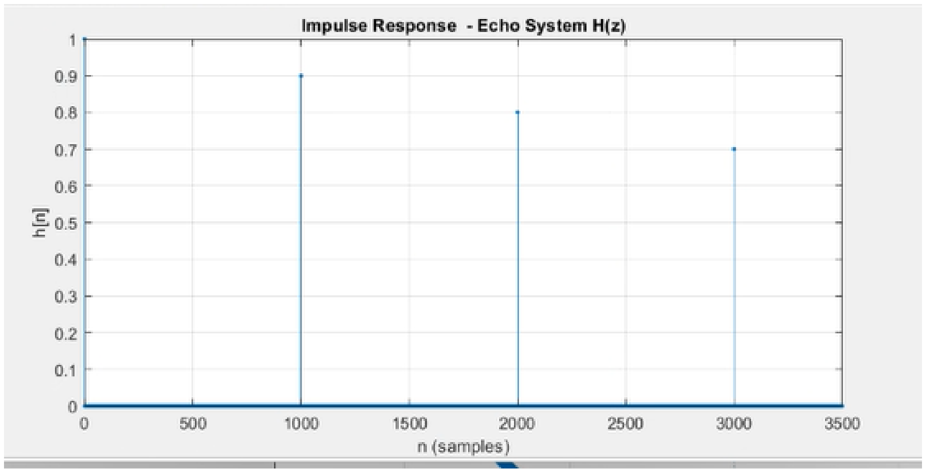
Figure 2







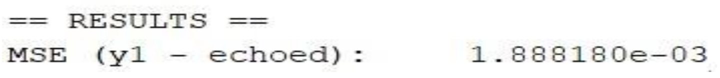




Part 2

In this part, the input signal x_{mono} is applied to the echo system using the *filter* function. The mean square error (MSE) is then calculated using the previously explained function *calc_mse*. After that, the output of the echo system is normalized to prevent clipping and to ensure that the audio signal can be safely saved. Finally, the processed audio signal is saved using the `audiowrite` function.

```
y1 = filter(num_H, den_H, x_mono);

% Compute MSE for y1
MSE_y1 = calc_mse(y1, x_mono);

% Normalize and save the echoed audio
y1_normalized = y1 / max(abs(y1));
audiowrite('audio_with_echo.wav', y1_normalized, Fs);
```



```
== RESULTS ==
MSE (y1 - echoed):      1.888180e-03
```

Part 3

In this part, we construct the equalizer system whose transfer function is the reciprocal of the echo system $H(z)$. Using the echo system, the numerator of the equalizer is set to 1, while the denominator is chosen as the numerator of the echo system.

The LCCDE is obtained by hand as shown in figure 3.

The previously explained function (*Sec3Tasks*) is then used to plot all the required results.

```
% G(z) = 1 / H(z) -> num_G = 1, den_G = num_H
num_G = 1;
den_G = num_H;

fprintf("Generating plots for Equalizer System G(z)...\n");
Sec3Tasks_2_1(' - Equalizer System G(z)', num_G, den_G, num_G, den_G, f, Fs, N , 20000, true, 1);
```

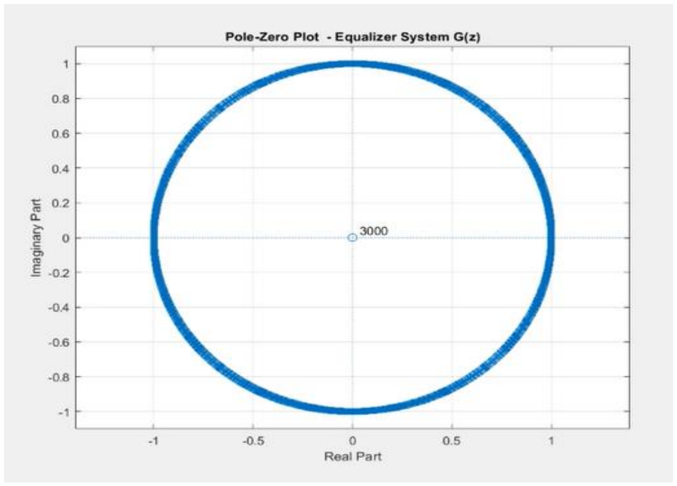
$$G[z] = \frac{1}{1+0{,}9\,z^{-1000} + 0{,}8\,z^{-2000} + 0{,}7\,z^{-3000}}$$

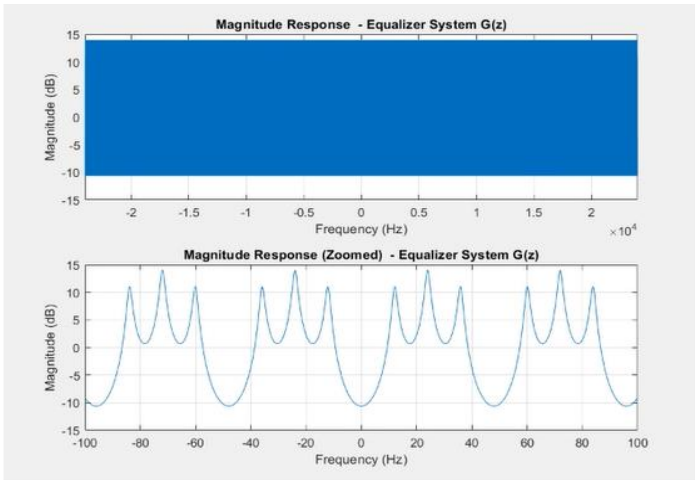
$$\frac{Y[z]}{X[z]} = \frac{1}{1+0{,}9\,z^{-1000} + 0{,}8\,z^{-2000} + 0{,}7\,z^{-3000}}$$

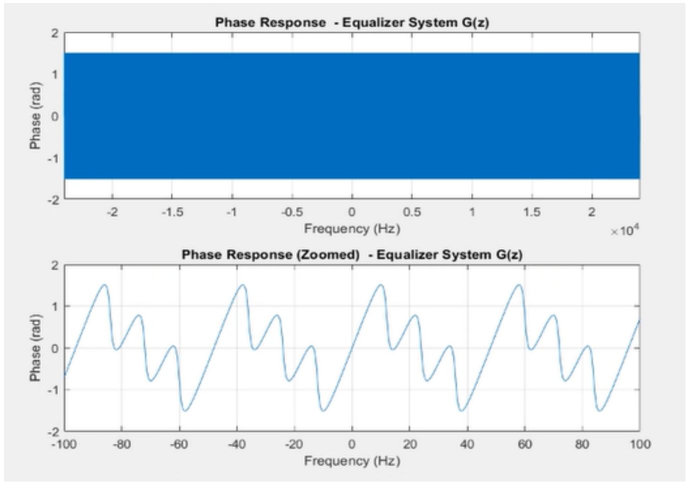
$$Y[z]\left[1+0{,}9\,z^{-1000} + 0{,}8\,z^{-2000} + 0{,}7\,z^{-3000}\right] = X[z]$$

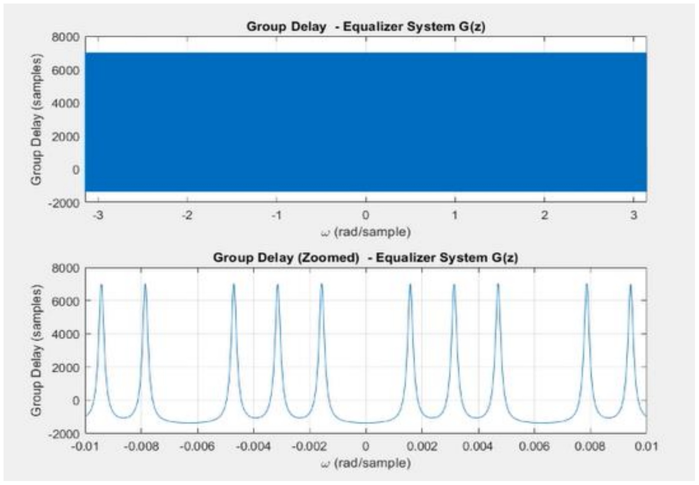
$$\boxed{y[n] = x[n] - 0{,}9\,y[n-1000] - 0{,}8\,y[n-2000] + 0{,}7\,y[n-3000]}$$

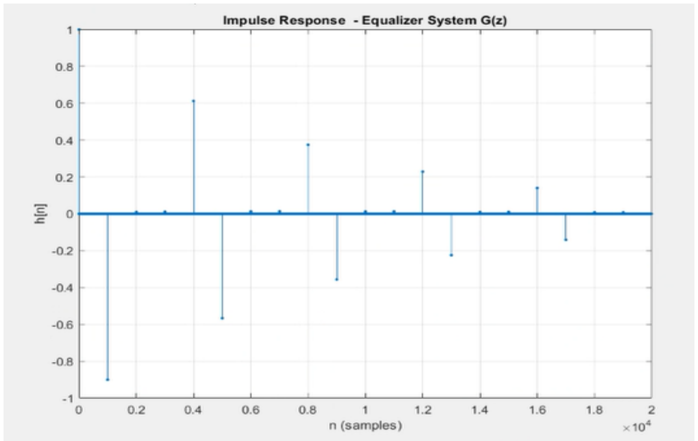
Figure 3







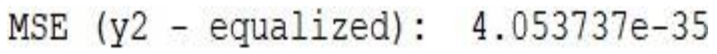




Part 4

In this part, the output of the echo system Y_1 is applied to the equalizer system using the *filter* function. The mean square error (MSE) is then computed using the previously explained function *calc_mse*. The output of the equalizer system is subsequently normalized to prevent clipping and to ensure that the audio signal can be safely saved. Finally, the processed audio signal is saved using the *audiowrite* function.

```
y2 = filter(num_G, den_G, y1);

% Compute MSE for y2
MSE_y2 = calc_mse(y2, x_mono);

% Normalize and save
y2_normalized = y2 / max(abs(y2));
audiowrite('audio_equalized.wav', y2_normalized, Fs);

fprintf('\n== RESULTS ==\n');
fprintf('MSE (y1 - echoed):     %.6e\n', MSE_y1);
fprintf('MSE (y2 - equalized):  %.6e\n', MSE_y2);
```



```
MSE (y2 - equalized):  4.053737e-35
```

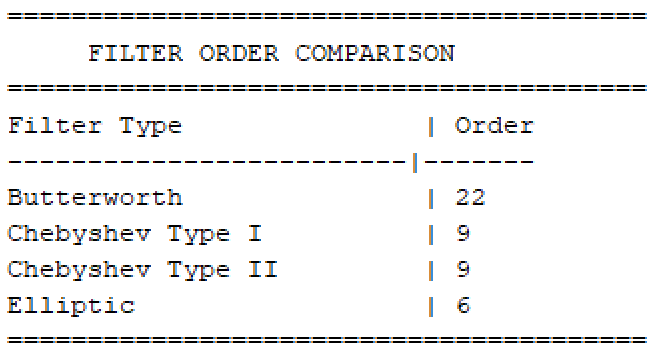
Task 2

It's required in this task to create four different types of digital filters: butterworth, chebyshev1, chebyshev2 and elliptical filters. It's required to design these filters with a passband frequency of 3 KHz, stopband of 4 KHz, a maximum allowed passband ripple of 1 dB and a minimum attenuation for the stopband ripple of 50 dB. The frequencies are normalized to *w* and passed into the filter order function (*buttord, cheb1ord, etc.)* and the order is given to the filter function itself. A for loop then plots all requirements using the *Sec3Tasks* function, calculates MSE and energy lost in each filter.

```
%% ---------------------- Task 2: Design of Digital IIR LPF -----------------
fprintf('\n=== TASK 2: Design of Digital IIR LPF ===\n\n');

% filter specs
fp = 3000;       % passband edge in Hz
fstop = 4000;    % stopband edge in Hz
Ap = 1;          % max passband ripple (dB)
As = 50;         % min stopband attenuation (dB)

% normalize frequencies
Wp = fp / (Fs/2);        % passband edge
Ws = fstop / (Fs/2);     % stopband edge

% Design all four filter types
fprintf('=== Designing Filters ===\n');

% BUTTERWORTH FILTER
[n_butter, Wn_butter] = buttord(Wp, Ws, Ap, As);          % returns min order & cuff off 
freq
[b_butter, a_butter] = butter(n_butter, Wn_butter,'low'); %returns filter coefficients
```

```
% CHEBYSHEV TYPE I FILTER
[n_cheby1, Wn_cheby1] = cheb1ord(Wp, Ws, Ap, As);
[b_cheby1, a_cheby1] = cheby1(n_cheby1, Ap, Wn_cheby1,'low');

% CHEBYSHEV TYPE II FILTER
[n_cheby2, Wn_cheby2] = cheb2ord(Wp, Ws, Ap, As);
[b_cheby2, a_cheby2] = cheby2(n_cheby2, As, Wn_cheby2,'low');

% ELLIPTIC FILTER
[n_ellip, Wn_ellip] = ellipord(Wp, Ws, Ap, As);
[b_ellip, a_ellip] = ellip(n_ellip, Ap, As, Wn_ellip,'low');


% Filter Order Comparison Table
fprintf('\n========================================\n');
fprintf('     FILTER ORDER COMPARISON\n');
fprintf('========================================\n');
fprintf('%-25s | Order\n', 'Filter Type');
fprintf('-------------------------|-------\n');
fprintf('%-25s | %d\n', 'Butterworth', n_butter);
fprintf('%-25s | %d\n', 'Chebyshev Type I', n_cheby1);
fprintf('%-25s | %d\n', 'Chebyshev Type II', n_cheby2);
fprintf('%-25s | %d\n', 'Elliptic', n_ellip);
fprintf('========================================\n\n');

% Apply filters to audio and compute metrics
fprintf('=== Applying Filters to Audio Signal ===\n\n');

% Store filter data
filters = {'butter', 'cheby1', 'cheby2', 'ellip'};
filter_names = {'Butterworth', 'Chebyshev Type I', 'Chebyshev Type II', 'Elliptic'};
filter_coefs = { % a 1*4 array with each cell having a struct b and a, b for num coeff.
and a for den coeff.
    struct('b', b_butter, 'a', a_butter);
    struct('b', b_cheby1, 'a', a_cheby1);
    struct('b', b_cheby2, 'a', a_cheby2);
    struct('b', b_ellip, 'a', a_ellip)
};

% storage for results
MSE_results = zeros(4, 1);         %empty 4*1 array
Energy_loss_results = zeros(4, 1); %empty 4*1 array

for i = 1:4
    fprintf('Processing: %s...\n', filter_names{i});

    b = filter_coefs{i}.b;
    a = filter_coefs{i}.a;

    % high order filters cause instability due to their very large/small
    % coefficients
    % convert to second-order section for numerical stability
    [z, p, k] = tf2zpk(b, a);
    [sos, g] = zp2sos(z, p, k);

    % Generate plots using SOS (Section 3 requirements)
    Sec3Tasks_2_1([' - ' filter_names{i} ' LPF'], sos, 1, b, a, f, Fs, N, 600, false, g);
%pass num as SOS and den is ignored for second order sections

    % apply filter to audio
    y_filtered = filter(b, a, x_mono);
    % Compute MSE
    MSE_results(i) = calc_mse(y_filtered, x_mono);

    % Compute energy loss percentage
    Energy_loss_results(i) = calc_energylost(x_mono, y_filtered);

end

% Results Summary
fprintf('\n========================================\n');
fprintf('     FILTER PERFORMANCE COMPARISON\n');
fprintf('========================================\n');
fprintf('%-25s | MSE           | Energy Loss (%%)\n', 'Filter Type');
fprintf('-------------------------|---------------|----------------\n');
for i = 1:4
    fprintf('%-25s | %.6e | %.4f\n', filter_names{i}, MSE_results(i),
Energy_loss_results(i));
end
fprintf('========================================\n\n');

% find minimum mse and energy lost
[min_MSE, idx_MSE] = min(MSE_results);
[min_loss, idx_loss] = min(Energy_loss_results);

fprintf('=== BEST FILTER ANALYSIS ===\n');
fprintf('Best filter (minimum MSE): %s\n', filter_names{idx_MSE});
fprintf('   MSE = %.6e\n\n', min_MSE);
fprintf('Best filter (minimum energy loss): %s\n', filter_names{idx_loss});
fprintf('   Energy Loss = %.4f%%\n\n', min_loss);
```

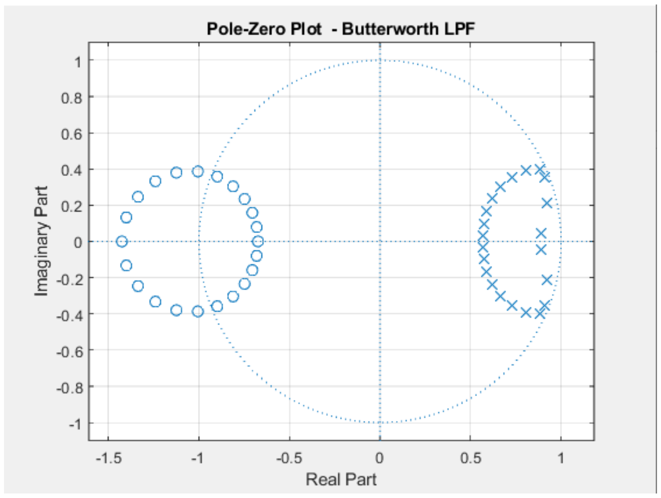
The filter orders are as follows:

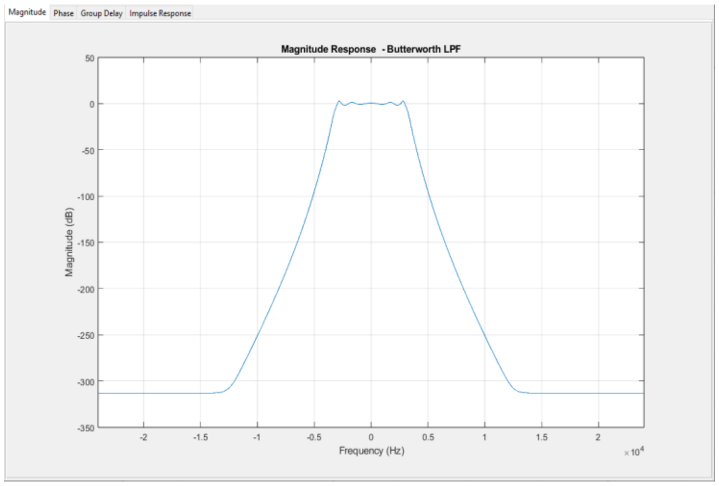


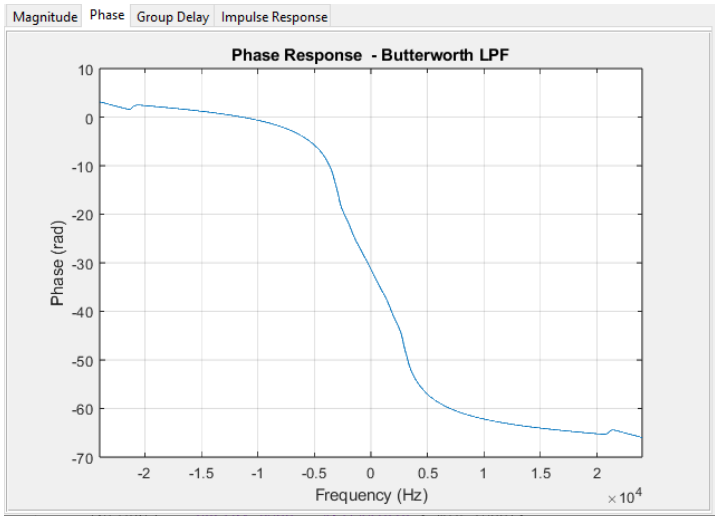
```
========================================
     FILTER ORDER COMPARISON
========================================
Filter Type               | Order
-------------------------|-------
Butterworth               | 22
Chebyshev Type I          | 9
Chebyshev Type II         | 9
Elliptic                  | 6
========================================
```

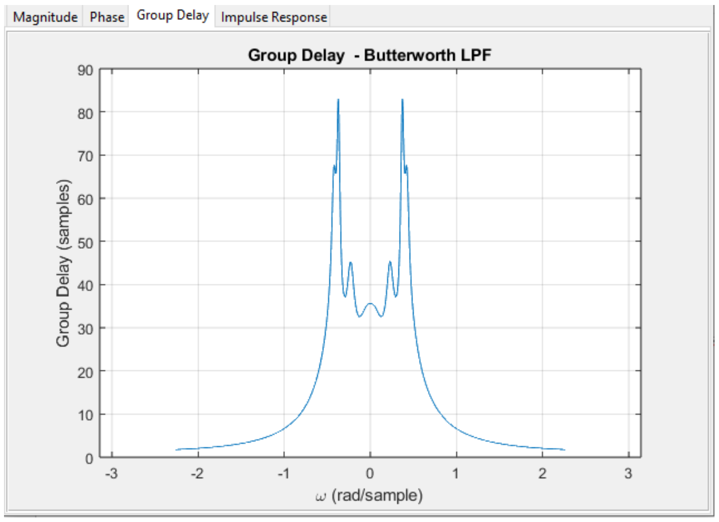
The rest of the results are displayed in the following pages.

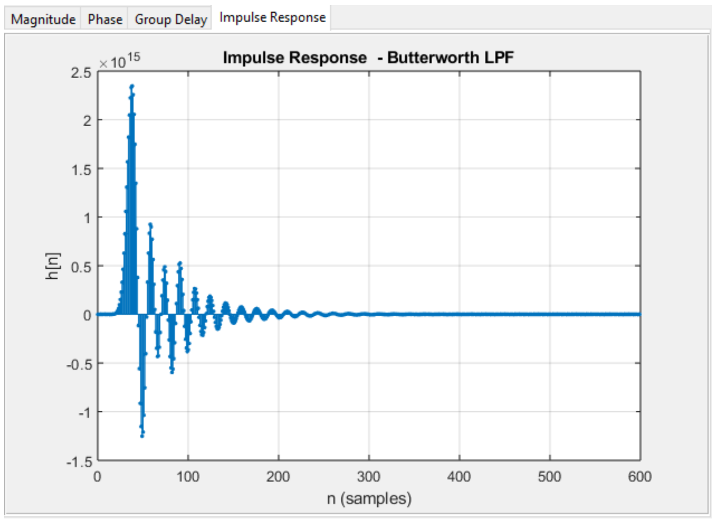
Butterworth



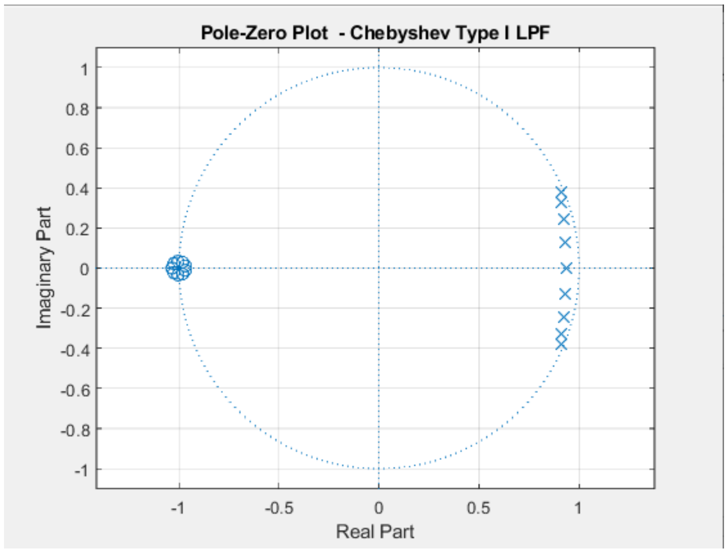


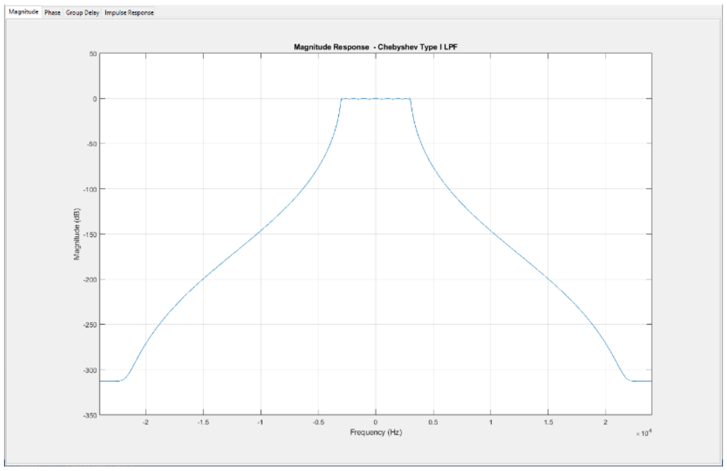


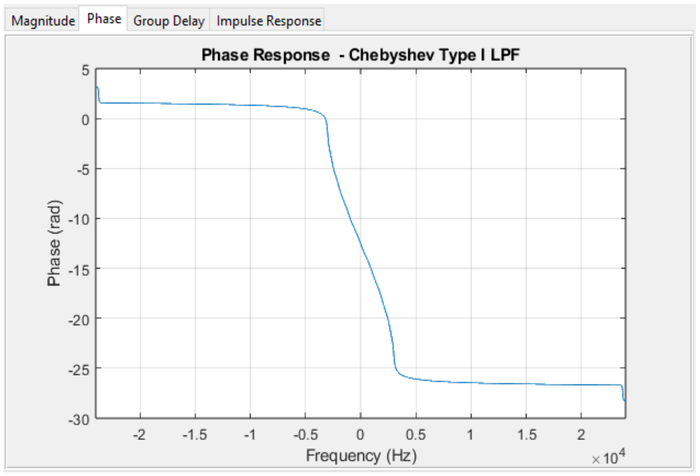


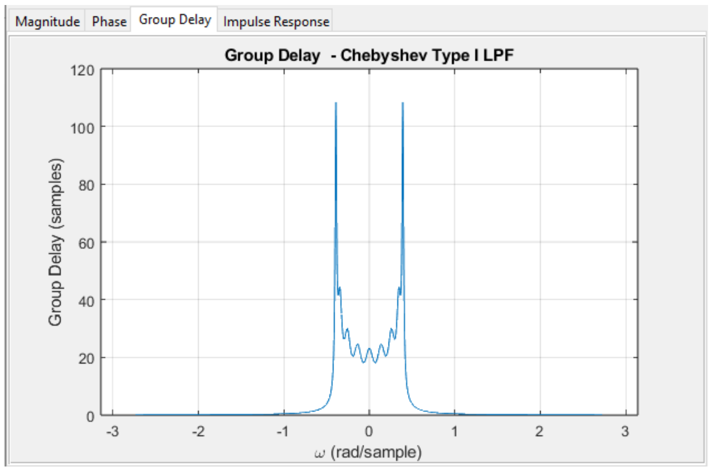


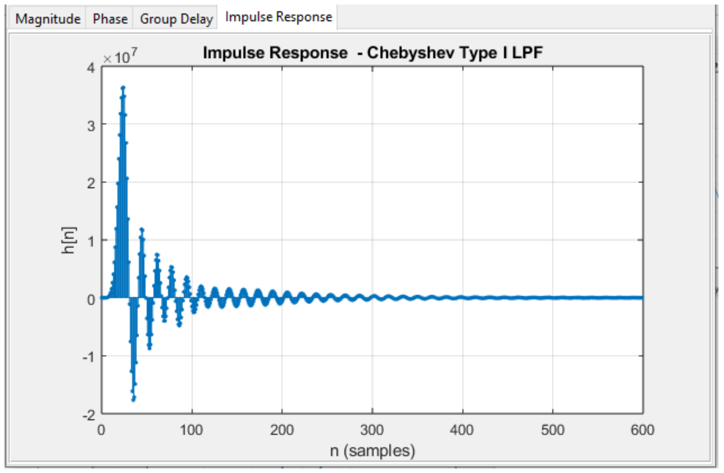
Chebyshev I


Pole-Zero Plot - Chebyshev Type I LPF
Imaginary Part
Real Part

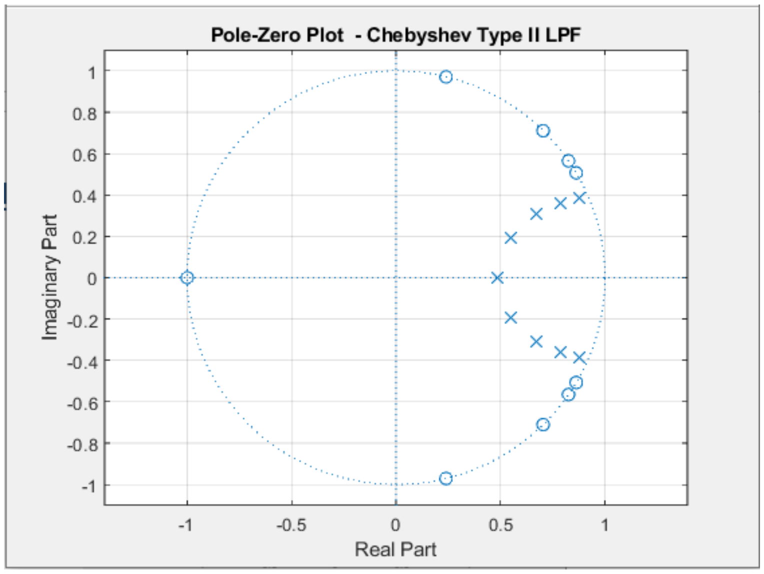

Magnitude
Phase
Group Delay
Impulse Response
Magnitude Response - Chebyshev Type I LPF
Magnitude (dB)
Frequency (Hz)

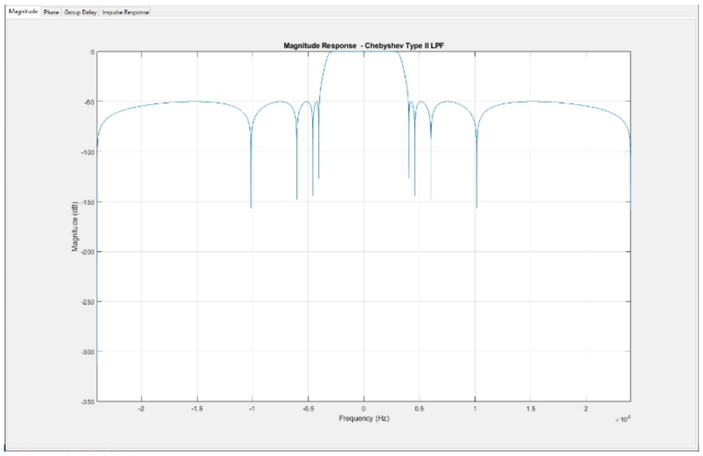

Magnitude
Phase
Group Delay
Impulse Response
Phase Response - Chebyshev Type I LPF
Phase (rad)
Frequency (Hz)

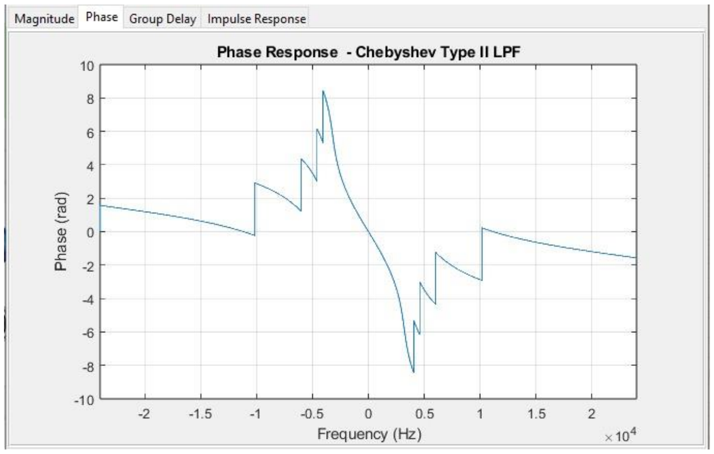

Magnitude
Phase
Group Delay
Impulse Response
Group Delay - Chebyshev Type I LPF
Group Delay (samples)
ω (rad/sample)

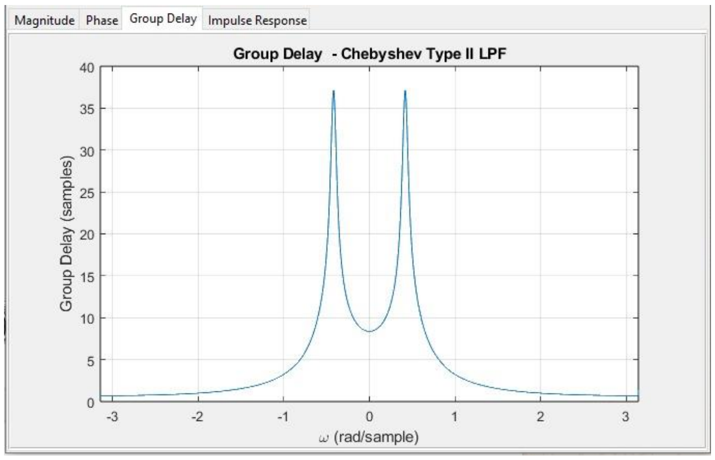

Magnitude
Phase
Group Delay
Impulse Response
Impulse Response - Chebyshev Type I LPF
h[n]
n (samples)

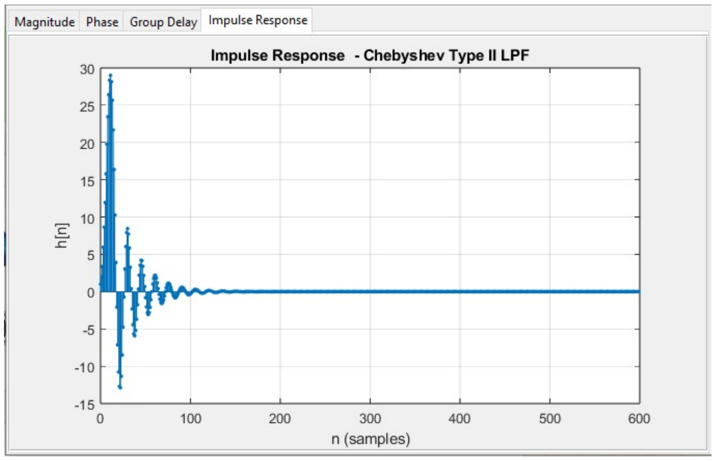
Chebyshev II



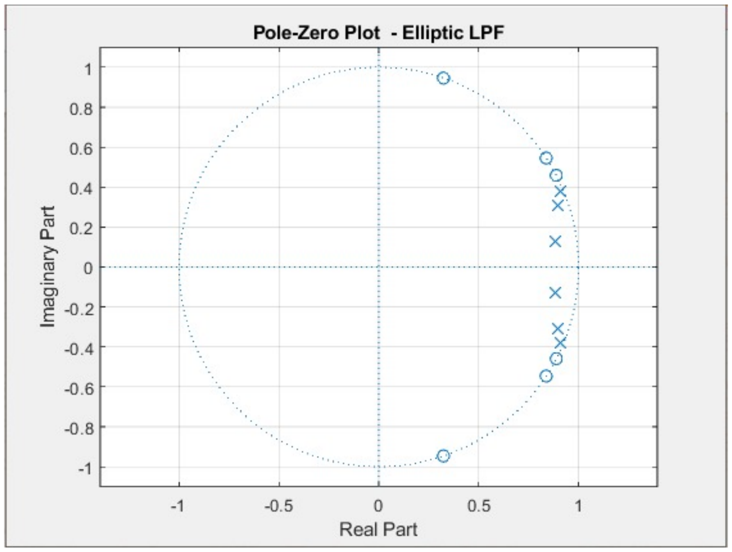


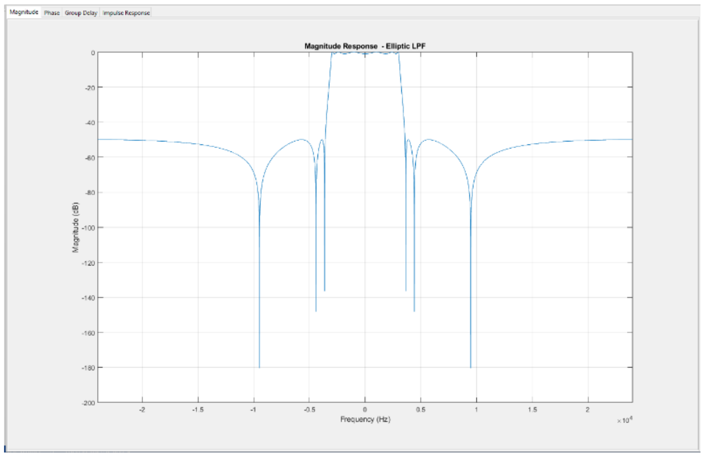


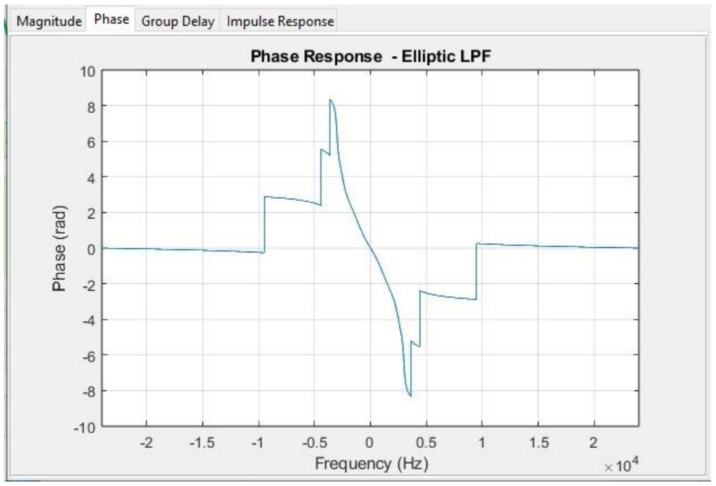


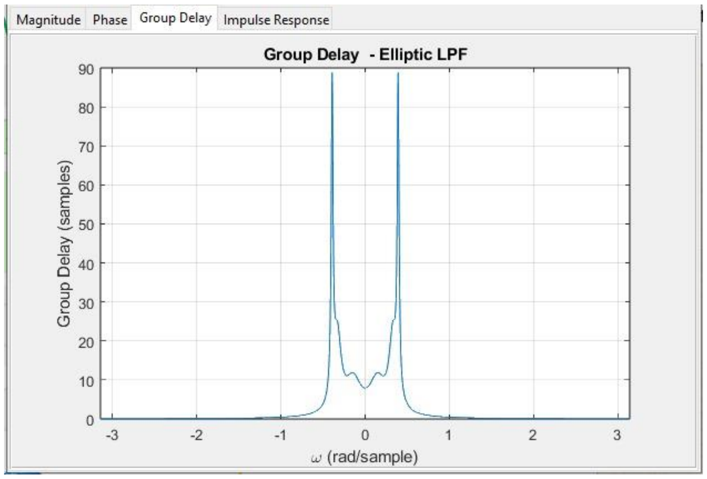


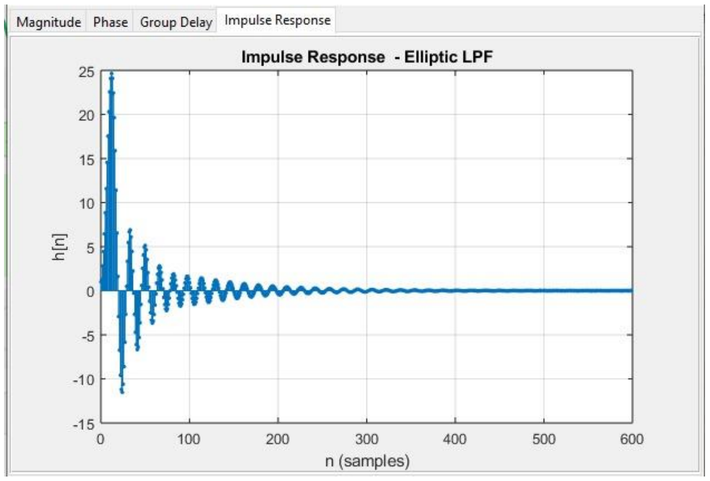
Elliptic


Pole-Zero Plot - Elliptic LPF
Imaginary Part
Real Part


Magnitude
Phase
Group Delay
Impulse Response
Magnitude Response - Elliptic LPF
Magnitude (dB)
Frequency (Hz)


Magnitude
Phase
Group Delay
Impulse Response
Phase Response - Elliptic LPF
Phase (rad)
Frequency (Hz)


Magnitude
Phase
Group Delay
Impulse Response
Group Delay - Elliptic LPF
Group Delay (samples)
ω (rad/sample)


Magnitude
Phase
Group Delay
Impulse Response
Impulse Response - Elliptic LPF
h[n]
n (samples)

Observations

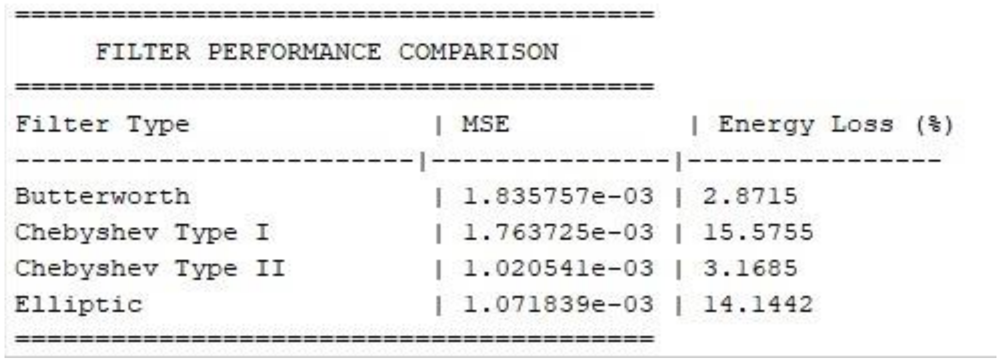
We can notice in the butterworth filter that it provides a smooth magnitude response without any ripples in the passband and stopband. However, it requires a very high order to meet required specs, which results in a wider transition band and slightly higher signal distortion than other filter types.

The Chebyshev I filter provides a sharper transition at the cost of ripples in the passband. This improves frequency selectivity but causes signal distortion due to amplitude variations in the passband.

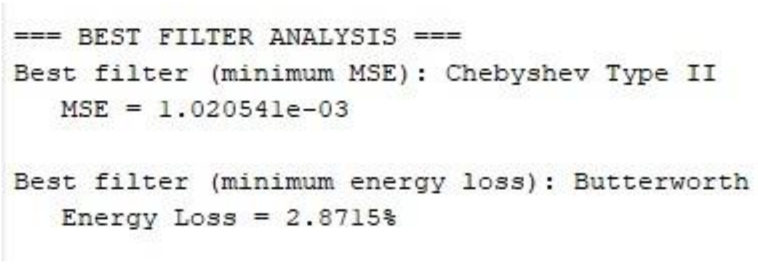
Unlike the ChebyI filter, the Chebyshev II has no ripples in the passband but we notice some in the stopband, which achieves better amplitude accuracy. However, its non-linear phase alongside the stopband ripples affect signal quality.

In the elliptic filter, we observe that it has the sharpest transition with the lowest order among all other types but allows ripples in the pass and stop bands. This results in excellent frequency selectivity, but at the expense of more complex phase behavior and potential signal distortion.

The mean square error and energy loss percentage are computed and given as follows:



```
========================================
     FILTER PERFORMANCE COMPARISON
========================================
Filter Type              | MSE           | Energy Loss (%)
-------------------------|---------------|----------------
Butterworth              | 1.835757e-03 | 2.8715
Chebyshev Type I         | 1.763725e-03 | 15.5755
Chebyshev Type II        | 1.020541e-03 | 3.1685
Elliptic                 | 1.071839e-03 | 14.1442
========================================
```



```
=== BEST FILTER ANALYSIS ===
Best filter (minimum MSE): Chebyshev Type II
   MSE = 1.020541e-03

Best filter (minimum energy loss): Butterworth
   Energy Loss = 2.8715%
```

The Chebyshev II achieves the lowest MSE due to its flat passband which preserves the signal as is, and its stopband ripples don't affect the original waveform.
The lowest energy loss % was observed in the butterworth filter, which is expected behavior because it has extremely low ripples in both of its bands.

The results show that filters with higher frequency selectivity, such as Chebyshev I and elliptic filters, introduce greater energy loss due to stronger attenuation of signal components. In contrast, Chebyshev II achieves the lowest mean square error by preserving passband amplitude, highlighting the trade-off between waveform preservation and filtering strength.

Task 3

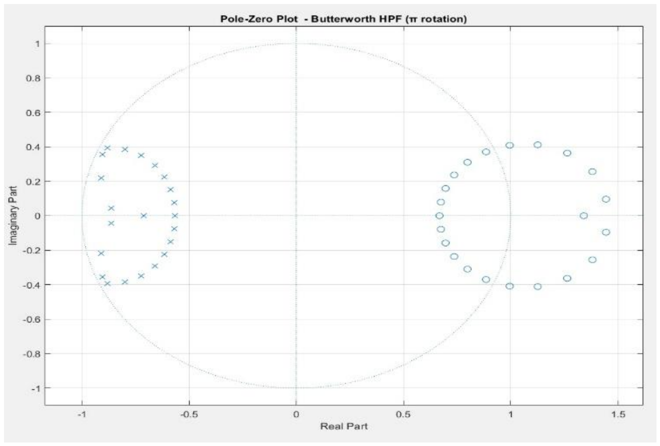
In this task, the concept of rotating the pole–zero pattern is used to transform the low-pass filter designed in task 2 into high-pass and band-pass filters. This transformation is achieved by applying the frequency-shifting property of the Discrete-Time Fourier Transform (DTFT).

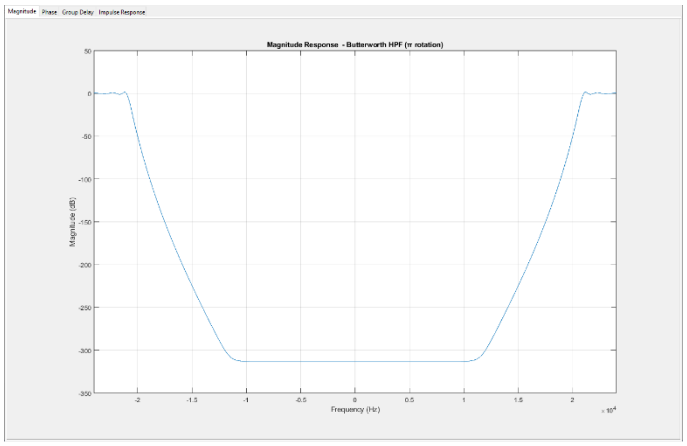
Part 1

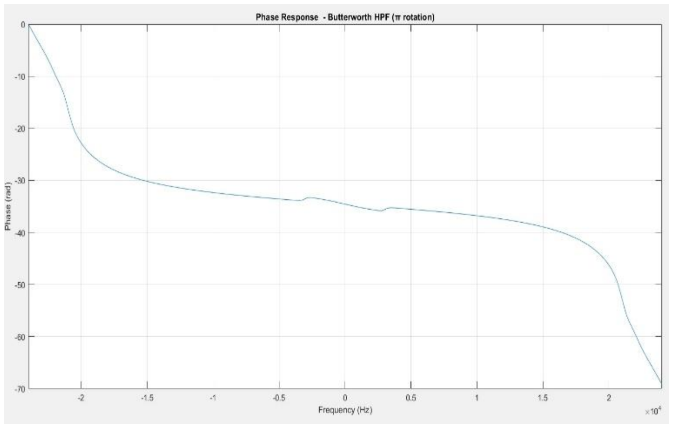
In this part, the previously designed Butterworth low-pass filter (LPF) is transformed into a high-pass filter (HPF). First, the zeros, poles, and gain are obtained from the Butterworth filter coefficients using the *tf2zp* function. The poles and zeros are then multiplied by -1 (i.e., $e^{j\pi}$), which corresponds to a 180° rotation in the z-plane and converts the filter into a high-pass filter.

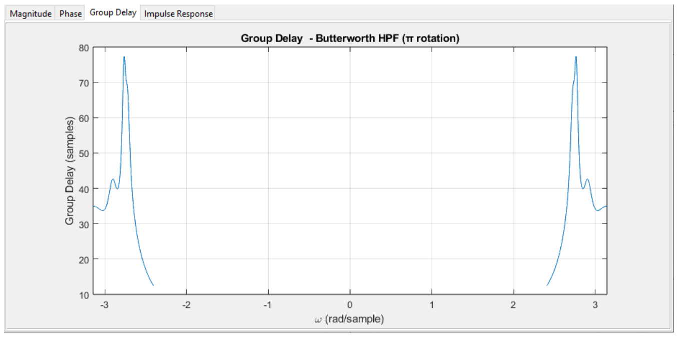
After that, the modified poles, zeros, and gain are converted back to Butterworth filter coefficients using the *zp2tf* function. Finally, all the required plots are generated using the previously explained function (*Sec3Tasks_2_1*).

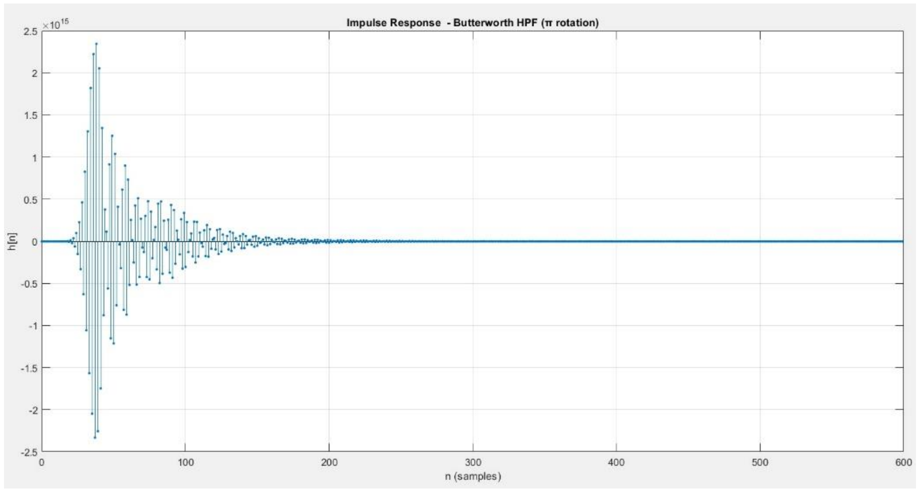
```
fprintf('\n=== TASK 3: Frequency Transformation using Pole-Zero Rotation ===\n\n');

%% Part A: Transform Butterworth LPF to HPF
fprintf('=== Part A: Highpass Filter (HPF) via pi rotation ===\n');

[z_lpf, p_lpf, k_lpf] = tf2zp(b_butter, a_butter);

% multiply by e^(j*pi) = -1
z_hpf = -z_lpf;
p_hpf = -p_lpf;
k_hpf = k_lpf;

[b_hpf, a_hpf] = zp2tf(z_hpf, p_hpf, k_hpf);

fprintf('Generating plots for HPF...\n');
[z, p, k] = tf2zpk(b_hpf, a_hpf);
[sos, g] = zp2sos(z, p, k);
Sec3Tasks_2_1(' - Butterworth HPF ', sos, 1, b_hpf, a_hpf, f, Fs, N, 600, false, g);
```


Pole-Zero Plot - Butterworth HPF (π rotation)
Imaginary Part
Real Part


Magnitude
Phase
Group Delay
Impulse Response
Magnitude Response - Butterworth HPF (π rotation)
Magnitude (dB)
Frequency (Hz)
×10⁴


Phase Response - Butterworth HPF (π rotation)
Phase (rad)
Frequency (Hz)
×10⁴


Magnitude
Phase
Group Delay
Impulse Response
Group Delay - Butterworth HPF (π rotation)
Group Delay (samples)
ω (rad/sample)

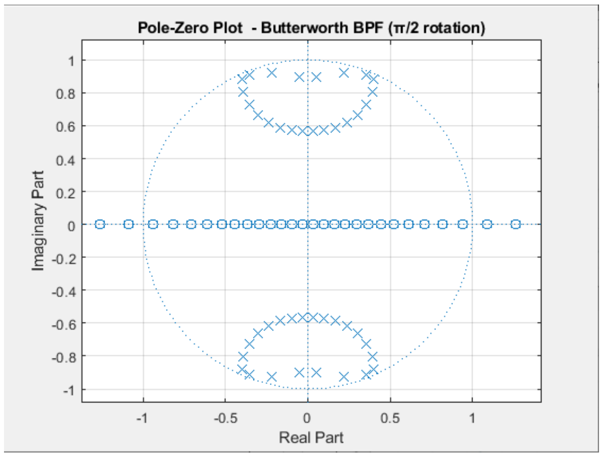

Impulse Response - Butterworth HPF (π rotation)
×10¹⁵
h[n]
n (samples)

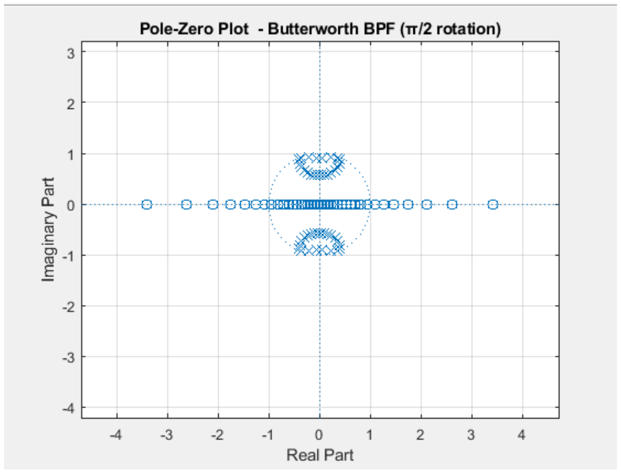
Part 2

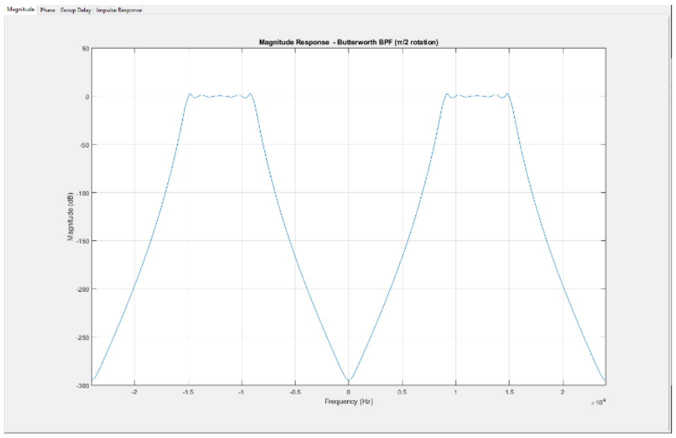
In this part, a band-pass filter (BPF) is obtained by transforming a previously designed Butterworth low-pass filter (LPF) using pole–zero manipulation in the *z*-domain. To achieve a BPF centered at ω=π/2, the poles and zeros of the LPF are rotated by +π/2 and −π/2. This is done by multiplying the LPF poles and zeros by *j* and dividing them by *j*, which corresponds to rotation by ±π/2 in the complex plane.

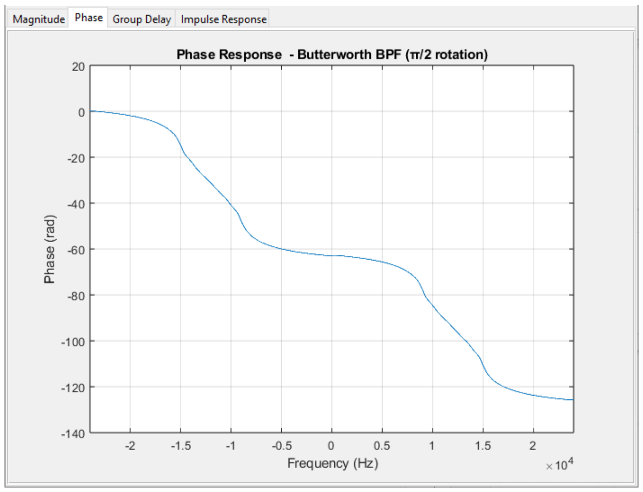
Two frequency-shifted versions of the original LPF are therefore created: one rotated in the positive direction and one in the negative direction. These two filters are then combined in parallel to ensure conjugate symmetry, which guarantees real-valued filter coefficients. The resulting transfer function is obtained by algebraically adding the two filters, and any small imaginary parts caused by numerical round-off are removed using the *real* function. Finally, the combined filter is converted to second-order sections (SOS) for numerical stability and analyzed using the *Sec3Tasks* function to generate the required plots.

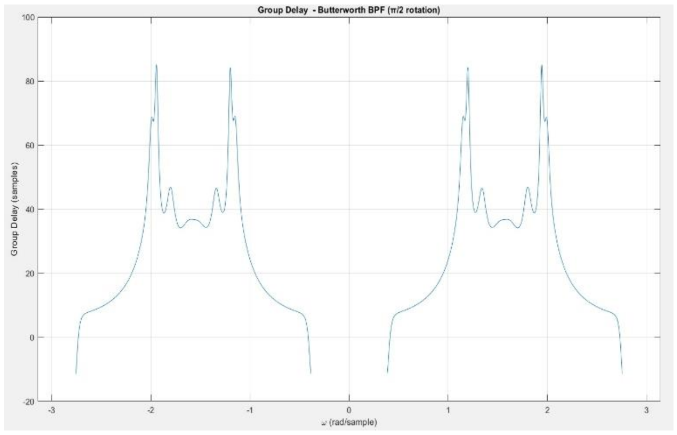
```
%% Part B: Transform Butterworth LPF to BPF (rotate by pi/2)
fprintf('\n=== Part B: Bandpass Filter (BPF) centered at pi/2 ===\n');

% Manual pole-zero rotation for digital BPF
z_bpf_pos = z_lpf.*j;
z_bpf_neg = z_lpf./j;

p_bpf_pos = p_lpf.*j;     % Rotate by +pi/2
p_bpf_neg = p_lpf./j;     % Rotate by -pi/2

% Create two separate filters
z_bpf_1 = z_bpf_pos ;
p_bpf_1 = p_bpf_pos ;
k_bpf_1 = k_lpf;

z_bpf_2 = z_bpf_neg;
p_bpf_2 = p_bpf_neg;
k_bpf_2 = k_lpf;

b_bpf_1 = k_bpf_1*poly(z_bpf_1);
a_bpf_1 = poly(p_bpf_1);
b_bpf_2 = k_bpf_2*poly(z_bpf_2);
a_bpf_2 = poly(p_bpf_2);

% Parallel combination: H(z) = H1(z) + H2(z)
% This means: (b1/a1) + (b2/a2) = (b1*a2 + b2*a1) / (a1*a2)
% Parallel combination: H(z) = H1(z) + H2(z)
% (b1/a1) + (b2/a2) = (b1*a2 + a1*b2) / (a1*a2)
b_bpf = conv(b_bpf_1, a_bpf_2) + conv(a_bpf_1, b_bpf_2);
a_bpf = conv(a_bpf_1, a_bpf_2);

b_bpf = real(b_bpf);
a_bpf = real(a_bpf);

[z, p, k] = tf2zpk(b_bpf, a_bpf);
[sos, g] = zp2sos(z,p,k);
fprintf('Generating plots for BPF...\n');
Sec3Tasks_2_1(' - Butterworth BPF (pi/2 rotation)', sos, 1, b_bpf, a_bpf, f, Fs, N, 600, false, g);
```

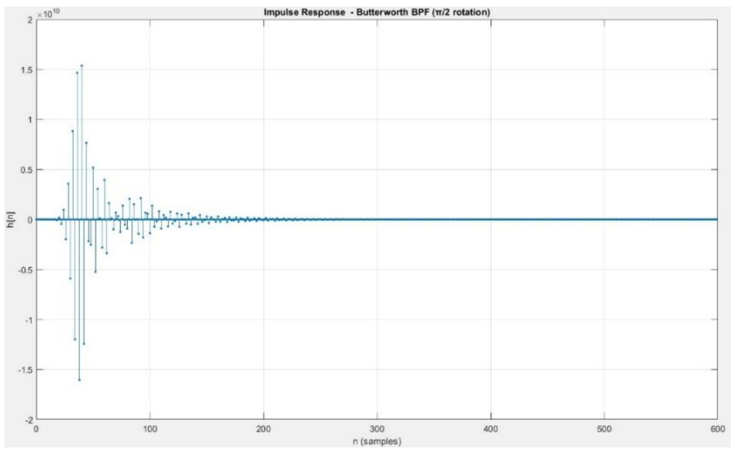

Pole-Zero Plot - Butterworth BPF (π/2 rotation)
Imaginary Part
Real Part


Pole-Zero Plot - Butterworth BPF (π/2 rotation)
Imaginary Part
Real Part


Magnitude
Phase
Group Delay
Impulse Response
Magnitude Response - Butterworth BPF (π/2 rotation)
Magnitude (dB)
Frequency (Hz)
$\times 10^4$


Magnitude
Phase
Group Delay
Impulse Response
Phase Response - Butterworth BPF (π/2 rotation)
Phase (rad)
Frequency (Hz)
$\times 10^4$

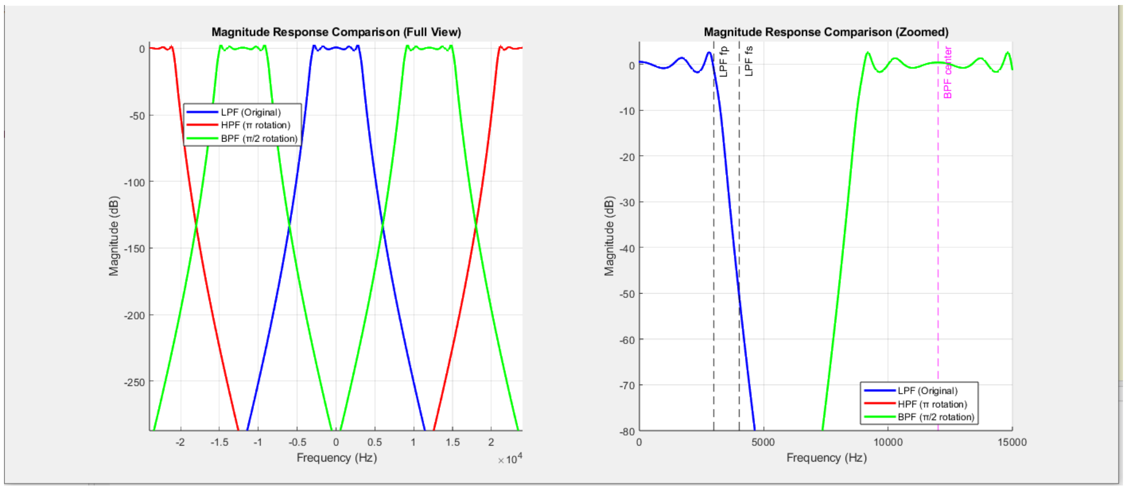

Group Delay - Butterworth BPF (π/2 rotation)
Group Delay (samples)
ω (rad/sample)


Impulse Response - Butterworth BPF (π/2 rotation)
$\times 10^{10}$
h[n]
n (samples)

Part 3

In this part, we plot the frequency responses of all the designed filters (LPF, HPF, and BPF) in order to compare them.

```
%% ----------EXTRAS Comparison Plot: LPF vs HPF vs BPF------------------
fprintf('\n=== Generating comparison plots ===\n');
figure('Name', 'Task 3: Filter Comparison (LPF, HPF, BPF)', 'Position', [150, 150, 1400, 600]);

[z, p, k] = tf2zpk(b_butter, a_butter);
[sos_lpf, g_lpf] = zp2sos(z,p,k);

[z, p, k] = tf2zpk(b_hpf, a_hpf);
[sos_hpf, g_hpf] = zp2sos(z,p,k);

[z, p, k] = tf2zpk(b_bpf, a_bpf);
[sos_bpf, g_bpf] = zp2sos(z,p,k);

H_lpf_t3 = g_lpf * freqz(sos_lpf, N, Fs, 'whole');
H_hpf_t3 = g_hpf*freqz(sos_hpf, N, Fs, 'whole');
H_bpf_t3 = g_bpf*freqz(sos_bpf, N, Fs, 'whole');

H_lpf_shift = fftshift(H_lpf_t3);
H_hpf_shift = fftshift(H_hpf_t3);
H_bpf_shift = fftshift(H_bpf_t3);

subplot(1,2,1);
hold on;
plot(f, 20*log10(abs(H_lpf_shift) + eps), 'b-', 'LineWidth', 2);
plot(f, 20*log10(abs(H_hpf_shift) + eps), 'r-', 'LineWidth', 2);
plot(f, 20*log10(abs(H_bpf_shift) + eps), 'g-', 'LineWidth', 2);
hold off;
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
title('Magnitude Response Comparison (Full View)');
legend('LPF (Original)', 'HPF (? rotation)', 'BPF (?/2 rotation)', 'Location', 'best');
grid on;
xlim([0, Fs/2]);
ylim([-100, 5]);

subplot(1,2,2);
hold on;
plot(f, 20*log10(abs(H_lpf_shift) + eps), 'b-', 'LineWidth', 2);
plot(f, 20*log10(abs(H_hpf_shift) + eps), 'r-', 'LineWidth', 2);
plot(f, 20*log10(abs(H_bpf_shift) + eps), 'g-', 'LineWidth', 2);
xline(3000, '--k', 'LPF fp', 'LineWidth', 1);
xline(4000, '--k', 'LPF fs', 'LineWidth', 1);
xline(Fs/4, '--m', 'BPF center', 'LineWidth', 1);
hold off;
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
title('Magnitude Response Comparison (Zoomed)');
legend('LPF (Original)', 'HPF (? rotation)', 'BPF (?/2 rotation)', 'Location', 'best');
grid on;
xlim([0, 15000]);
ylim([-80, 5]);
fprintf('\n=== End Of Project ===\n');
```


Magnitude Response Comparison (Full View)
Magnitude (dB)
Frequency (Hz)
×10^4
LPF (Original)
HPF (π rotation)
BPF (π/2 rotation)
Magnitude Response Comparison (Zoomed)
LPF fp
LPF fs
BPF center
Magnitude (dB)
Frequency (Hz)
LPF (Original)
HPF (π rotation)
BPF (π/2 rotation)