Bengisu Münteha Koç

21801974

EEE321 – 02

21.12.2024

EEE 321 Lab Work 5

Introduction

In this assignment, we designed a finite impulse response (FIR) bandpass filter based on parameters derived from student number, 21801974. The goal is to determine filter coefficients analytically, verify the frequency response, and implement the design in MATLAB. We then tested the filter on certain input signals to observe its behavior.

Background

Finite impulse response (FIR) filters are digital filters whose impulse response settles to zero in a finite number of samples. An FIR filter is typically represented as:

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

where L is the filter length, h[k] are the filter's impulse-response coefficients, and x[n] is the input signal. Because all poles of an FIR filter lie at the origin in the z-plane , FIR filters are inherently stable and can be designed to exhibit linear phase characteristics, making them suitable for applications where phase linearity is crucial.

Hence IR filters have a finite impulse response and can be expressed as a polynomial in z^{-1}:

$$H(z) = \sum_{\{n=0\}}^{\{L-1\},} h[n]z^{\{-n\}}$$

The frequency response of the filter is given by

$$H\left(e^{\{j\omega\}}\right) = \sum_{\{n=0\}}^{\{L-1\},} h[n]e^{\{-j\,\omega n\}}$$

In this assignment, a conceptual way is used to design a bandpass filter is to start from two low-pass filters one with a cutoff ω_{c2}, one with ω_{c1}:

$$H_{BP}(z) = H_{LP1}(z) - H_{LP2}(z)$$

Ones the zeros are placed around the stopband frequencies, it can be written:

$$H(z) = c \prod_{\{k=1\}}^{\{L-1\}} (1 - z_k z^{-1})$$

Where $z_k = e^{j\theta k}$ are the zeros on or near the unit circle chosen to carve out the desired band. By expanding this, coefficient h[n] can be found.

The ideal infinite length bandpass filter's impulse response can be written as

$$h_d[n] = \frac{\sin(\omega_{c2}n) - \sin(\omega_{c1}n)}{\pi n} \quad \text{for } n \neq 0,$$

$$h_d[0] = \frac{\omega_{c2} - \omega_{c1}}{\pi}$$

Calculations

While the zero-pole design gives a conceptual placement of zeros, once the polynomial H(z) is formed, the impulse response coefficients h[n] are simply the polynomial's coefficients. For a few sample values:

$$h[0] = \frac{\omega_{c2} - \omega_{c1}}{\pi} = \frac{\frac{\pi}{9} - \frac{\pi}{11}}{\pi} = \frac{2}{99} \approx 0.0202.$$

$$h[1] = \frac{\sin\left(\frac{\pi}{9}\right) - \sin\left(\frac{\pi}{11}\right)}{\pi} \approx \frac{0.3420 - 0.2817}{\pi} = \frac{0.0603}{\pi} \approx 0.0192.$$

Other values are calculated as well, and these values are recorded in an MATLAB array.

Design and Results

The results are recorded in a MATLAB array and code is written by using the background approach. The code is listed in Code Listing 1. The impulse response array h[n] is printed as below:

[0.0202 0.0192 0.0163 0.0117 0.0060 -0.0003 -0.0066 -0.0121 -0.0165 -0.0191 -0.0199 -0.0186 -0.0155 -0.0109 -0.0052 0.0009 0.0069 0.0121]

Code Listing 1: Question 1

```
% Given impulse response array h[n]
h = [0.0202    0.0192    0.0163    0.0117    0.0060   -0.0003   -0.0066 ...
    -0.0121   -0.0165   -0.0191   -0.0199   -0.0186   -0.0155   -0.0109 ...
    -0.0052    0.0009    0.0069    0.0121];


% Print h[n] as an array
disp('h[n] = ');
disp(h);


% Plot h[n]
figure;
stem(0:length(h)-1, h, 'filled');
xlabel('n');
ylabel('h[n]');
title('Impulse Response h[n]');
grid on;


% Compute and plot zeros of H(z)
% H(z) = sum_{n=0}^{L-1} h[n]*z^{-n}
% Zeros of H(z) are roots of the polynomial formed by h in z^{-1},
% i.e. roots of fliplr(h) if we treat h as polynomial coefficients in z^-1.
% To find zeros, treat h as coefficients of a polynomial in z^{-1}:
% H(z) = h[0] + h[1]*z^{-1} + ... + h[L-1]*z^{-(L-1)}
% The zeros in terms of z are roots of: h[0]*z^{L-1} + h[1]*z^{L-2} + ... + h[L-
1] = 0
% That corresponds to roots of flip(h) when considered as a polynomial in z.
r = roots(flip(h));
```

```
% Plot zeros on the z-plane
figure;
plot(real(r), imag(r), 'o', 'MarkerSize', 8, 'LineWidth', 1.5);
hold on;
% Draw unit circle
theta = linspace(0, 2*pi, 200);
plot(cos(theta), sin(theta), 'r--');
hold off;
xlabel('Real');
ylabel('Imag');
title('Zeros of H(z)');
axis equal; grid on;


% Frequency response
% Compute H(e^{j\omega}) on a dense grid of frequencies
Nfft = 1024;
Hf = fft(h, Nfft);       % Discrete-time Fourier transform approximation
omega = linspace(0, 2*pi, Nfft);  % Frequencies from 0 to 2*pi


% Magnitude Response |H(e^{j\omega})|
figure;
plot(omega, abs(Hf));
xlabel('\omega (rad/sample)');
ylabel('|H(e^{j\omega})|');
title('Magnitude Response');
grid on;


% Phase Response Φ(e^{j\omega})
figure;
plot(omega, angle(Hf));
xlabel('\omega (rad/sample)');
ylabel('Phase (radians)');
title('Phase Response');
```

```
grid on;
```

In this code block, impulse response h[n], zeros of H(z), magnitude response and phase response are plotted as shown in Figures 1,2,3 and 4, respectively.

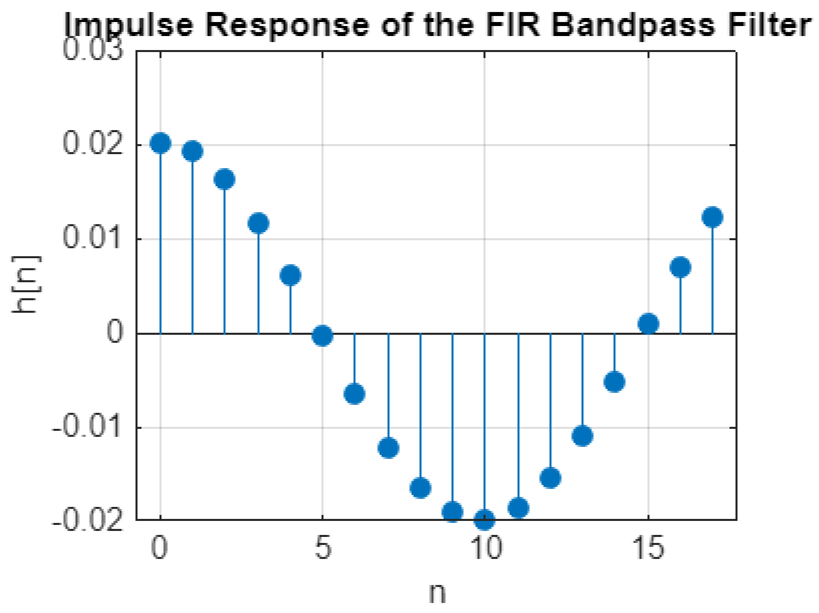


Figure 1: Impulse Response of FIR

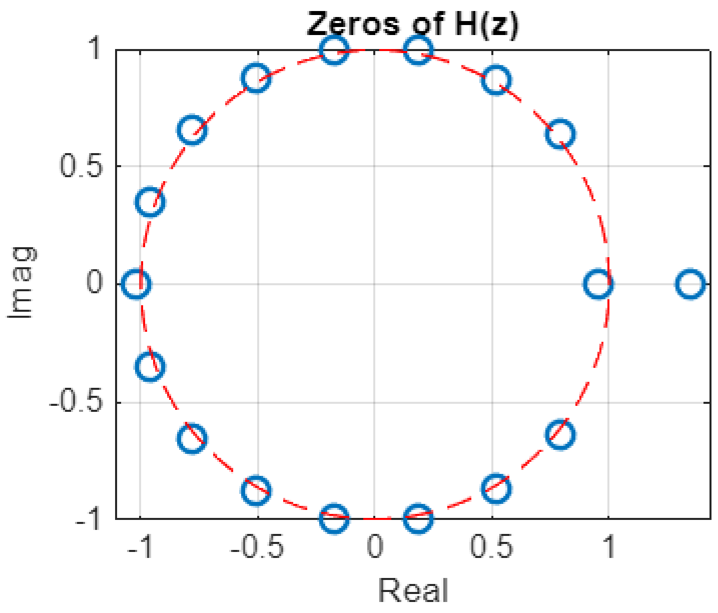


Figure 2: Zeros of H(z)

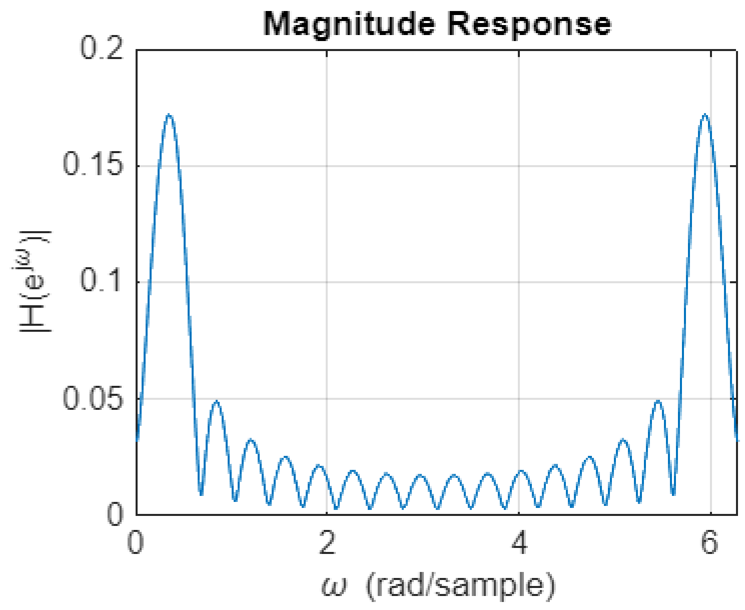


Figure 3: Magnitude Response

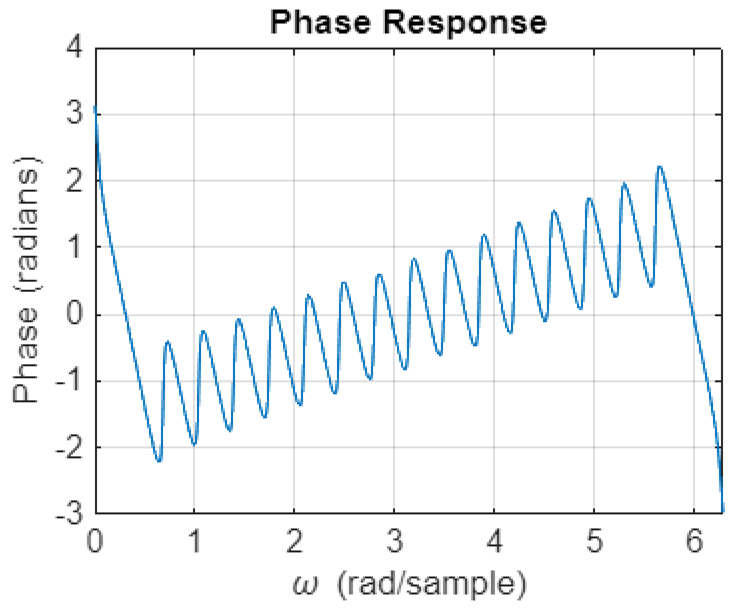


Figure 4: Phase Response

To test the frequency response of the designed filter is to apply a chirp signal as an input and observe the corresponding output. The given chirp signal is $x_a(t) = \cos(\alpha t^2)$. The frequency of the signal changes continuously with time. The instantaneous frequency of the signal is $\omega_i(t) = \frac{d}{dt}(\alpha t^2) = 2\alpha t$. This is not constant, and it does not return to a previous frequency after some finite period. Hence the given chirp signal is not periodic.

At t = 0, the instantaneous frequency is 0, and it is increasing linearly in time.

When the given sampling period is substituted to the chirp signal, $x_1[n]$ is found as

$$x_1[n] = \cos\left(\alpha n^2 \frac{\pi}{a \cdot 512}\right)$$

A discrete-time signal with a quadratic argument is not periodic, because the phase does not increase linearly with n; it increases quadratically, and does not repeat in a simple, integer multiple of 2π after a fixed number of samples. Thus $x_1[n]$ is not periodic.

To form a finite sequence $x_f[n]$; it is defined $x_f[n] = x_1[n]$ for $n \in [0,1023]$. It is 0 elsewhere.

The code for this part is written using mentioned approach and the plots are observed. The code is given in Code Listing 2.

Code Listing 2: Question 2

```
% Given parameters
alpha = 100;    % Alpha parameter for the chirp


% Sampling period
Ts = sqrt(pi/(alpha*512));


% Continuous-time signal: x_a(t) = cos(alpha * t^2)
% Not periodic. Instantaneous frequency: w_i(t) = 2 * alpha * t.


% Discrete-time signal:
% x_1[n] = x_a(n*Ts) = cos(alpha * (n*Ts)^2)
N = 1024;  % Number of samples for our finite sequence
```

```
n = 0:N-1;
x1 = cos(alpha * (n.*Ts).^2);


disp('The continuous-time signal x_a(t) is not periodic.');
disp('The discrete-time signal x_1[n] is also not periodic.');


% Form x_f[n]:
xf = zeros(1, 5*N);    % Example longer sequence with zeros outside [0,1023]
xf(1:N) = x1;          % x_f[n] = x_1[n] for n=0...1023, rest are zero


% Save x_f[n] to a file
% We can save in a .mat file or a text file
save('xf_data.mat', 'xf');


% Plot x_f[n]
figure;
stem(0:length(xf)-1, xf, 'filled');
xlabel('n');
ylabel('x_f[n]');
title('Finite Input Sequence x_f[n]');
grid on;


t = n*Ts;   % Continuous-time samples corresponding to n
xa = cos(alpha * t.^2);


figure;
plot(t, xa, 'b', 'LineWidth', 1.5);
hold on;
stem(t, x1, 'r');
```

```
hold off;
xlabel('t (seconds)');
ylabel('Amplitude');
title('x_a(t) and Samples x_1[n]');
legend('x_a(t)','x_1[n]');
grid on;
```

In this code block, the value of alpha is chosen arbitrarily.

The plot of $x_f[n]$ is shown in Figure 5. It can be observed from the plot that $x_1[n]$, accordingly $x_f[n]$, is not periodic.

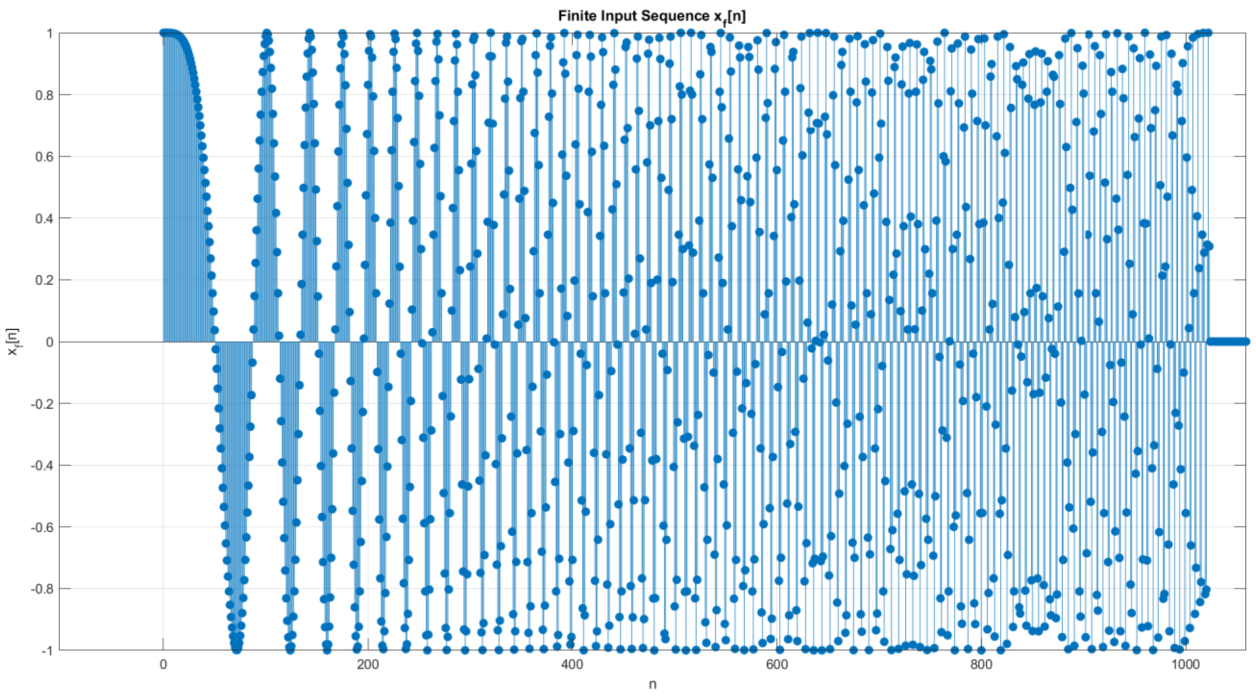


Figure 5: Finite Input Sequence of $x_f[n]$

To compare $x_1[n]$ with $x_a(t)$, they are plotted on the same graph. The result is shown in Figure 6. To be able to see clearer view of the result, the close up of the plot is given in Figure 7.

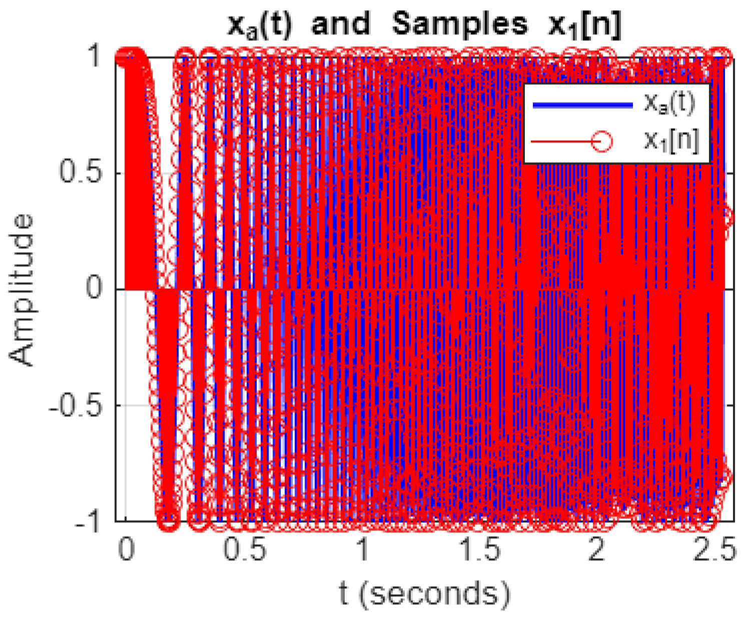


Figure 6: Comparison Plot of $x_a(t)$ and $x_1[n]$

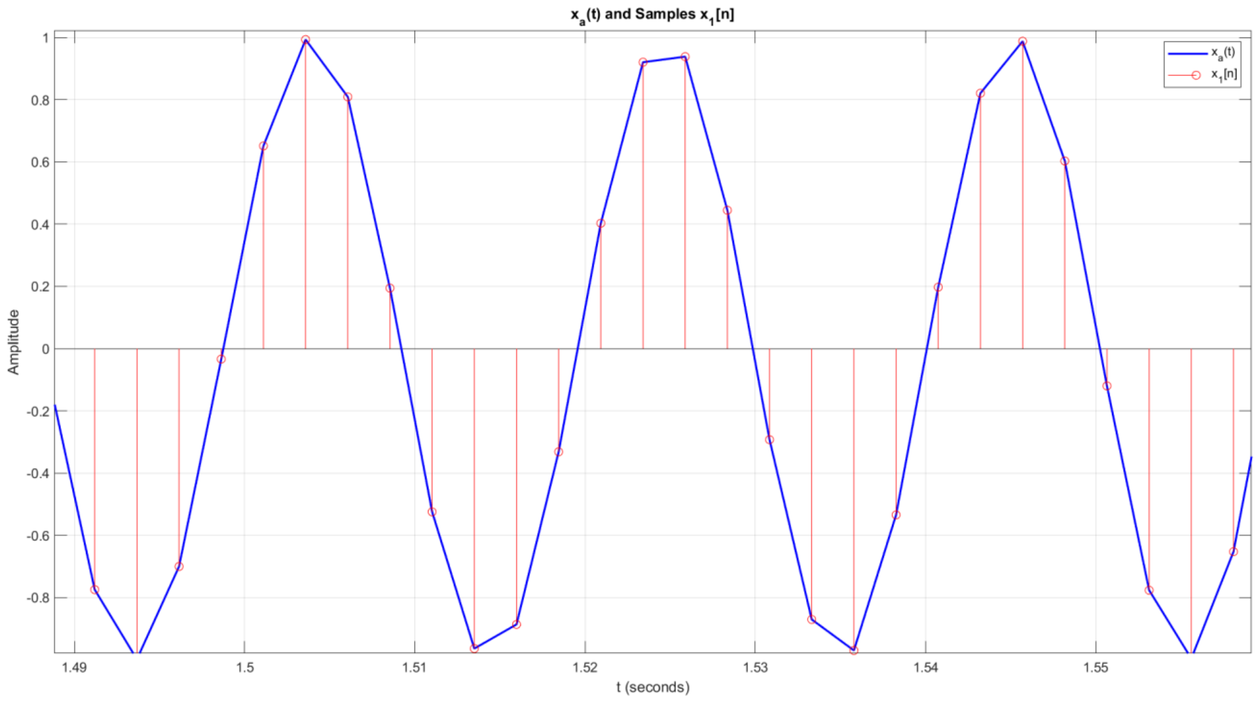


Figure 7: Close up View of Comparison Plot of $x_a(t)$ and $x_1[n]$

From the results, it can be concluded that $x_1[n]$ looks like $x_a(t)$.

With given alpha and sampling rate $T_s = \sqrt{\frac{\pi}{\alpha 8192}}$ s, x2[n] is generated. Then $x_g[n]$ finite sequence is formed with given N, where $x_g[n] = x_2[n]$ for n ∈ [0,8191]. It is 0 elsewhere. The code of this part is shown in Code Listing 3.

Code Listing 3: Question 3

```
% Given parameters
alpha = 1000;  % (radians^-2)
N = 8192;      % Number of samples from n=0 to n=8191


% Sampling period
Ts = sqrt(pi/(alpha*N));  % Ts = sqrt(pi/(alpha*8192))


% Generate x_2[n]
n = 0:N-1;
x2 = cos(alpha*(n*Ts).^2);
xg = x2;  % For n<0 or n>=N, x_g is not defined or zero, but we only consider
[0..8191]


% Store x_g[n] in a file


save('xg_data.mat','xg');


% Plot x_g[n] to visualize
figure;
plot(n, xg);
xlabel('n');
```

```
ylabel('x_g[n]');
title('Finite Input Sequence x_g[n]');
grid on;


% Just for clarity, display some info
disp('Parameters:');
disp(['alpha = ', num2str(alpha),' rad/s^2']);
disp(['Ts = ', num2str(Ts), ' s']);
disp(['Number of samples (N) = ', num2str(N)]);


disp('x_g[n] has been generated and saved to xg_data.mat');
```

The plot of $x_g[n]$ is shown in Figure 8.

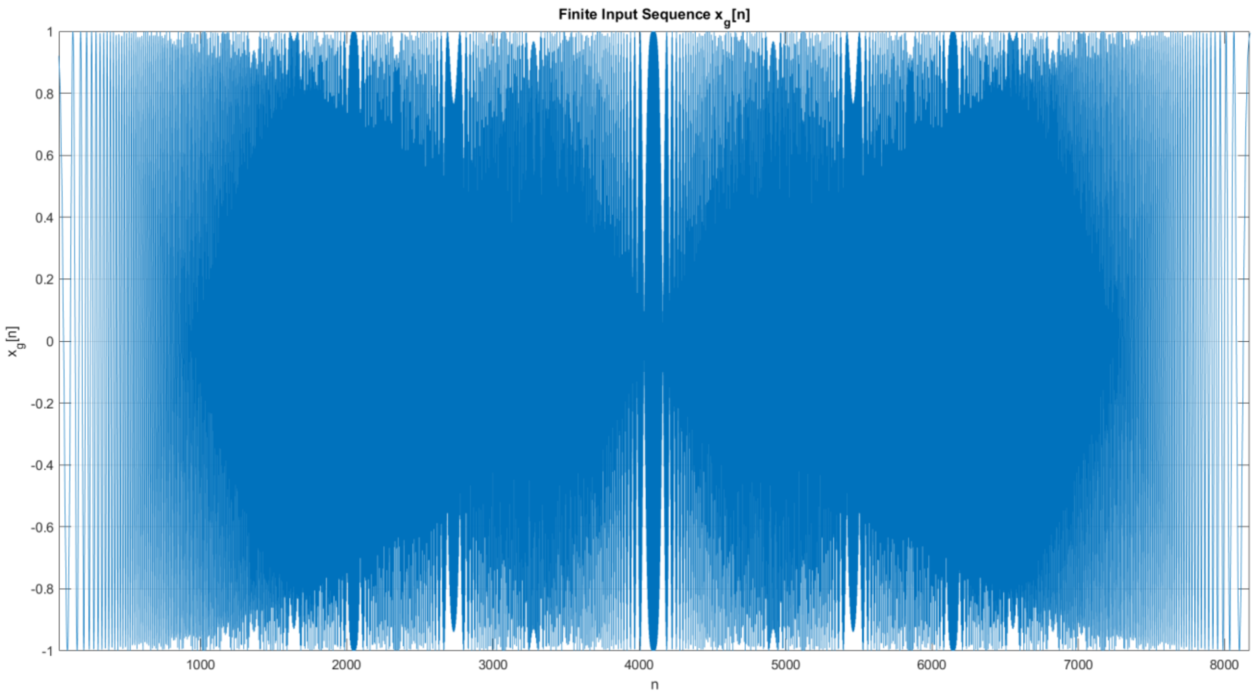


Figure 8: Finite Input Sequence of $x_g[n]$

Obtained signals $x_f[n]$ and $x_g[n]$ are passed through the filter with convolution property and the results are plotted and saved by the code listed in Code Listing 4.

Code Listing 4: Question 4

```
% Load x_f[n] and x_g[n] from files (previously saved)
load('xf_data.mat','xf');
load('xg_data.mat','xg');


% Check lengths
Nf = length(xf);
Ng = length(xg);


% Pass x_f[n] through filter
y1 = conv(xf, h);
N_y1 = 1022 + L;
if N_y1 > length(y1)
    error('Requested range exceeds output length. Adjust indexing.');
end


y1_extract = y1(1:N_y1+1);




% Plot y1[n]
figure;
plot(0:N_y1, y1_extract);
xlabel('n');
ylabel('y_1[n]');
title('Output y_1[n] for x_f[n] through the FIR filter');
grid on;
```

```
y2 = conv(xg, h);


N_y2 = 8190 + L;
if N_y2 > length(y2)
    error('Requested range for y2 exceeds output length. Adjust indexing.');
end


y2_extract = y2(1:N_y2+1);


% Save y1 and y2
save('y1_data.mat','y1_extract');
save('y2_data.mat','y2_extract');


% Plot y2[n]
figure;
plot(0:N_y2, y2_extract);
xlabel('n');
ylabel('y_2[n]');
title('Output y_2[n] for x_g[n] through the FIR filter');
grid on;
```

For a FIR filter, the output of the filter can be found by convolution. Hence the output is y[n] = x[n] * h[n].

Plot of $y_1[n]$ is shown in Figure 9.

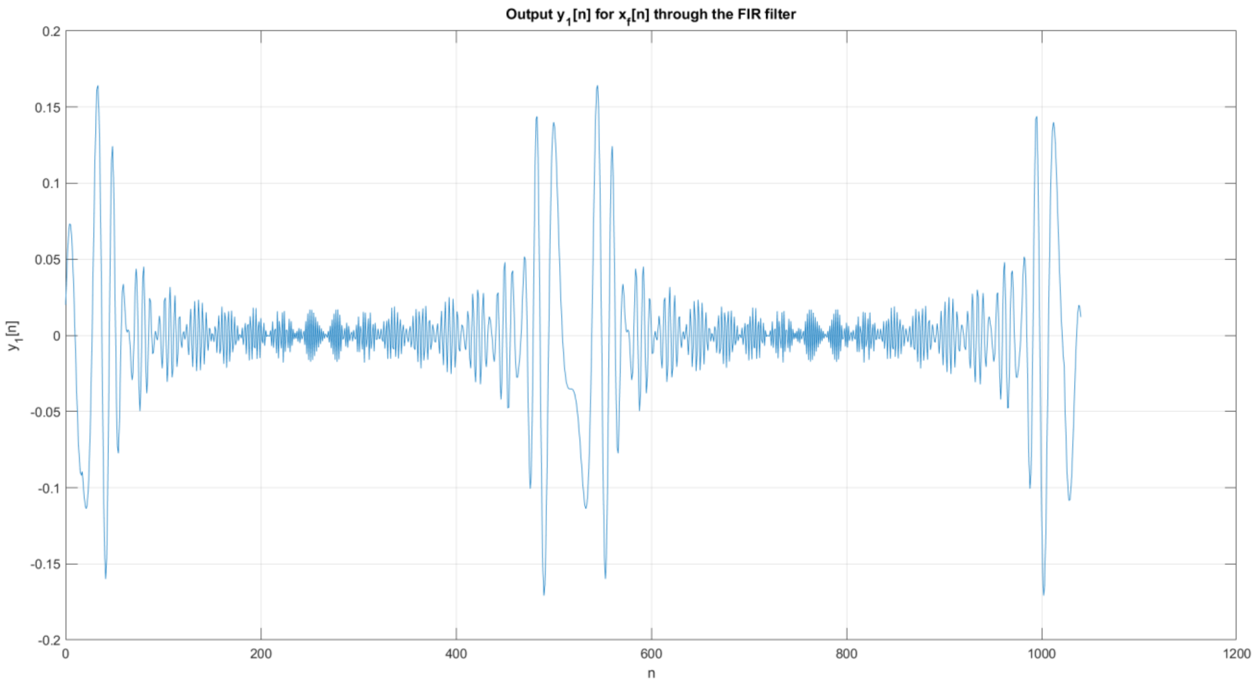


Figure 9: Plot of $y_1[n]$

The transient part of the output is shown in Figure 10, while the steady part is shown in Figure 11.

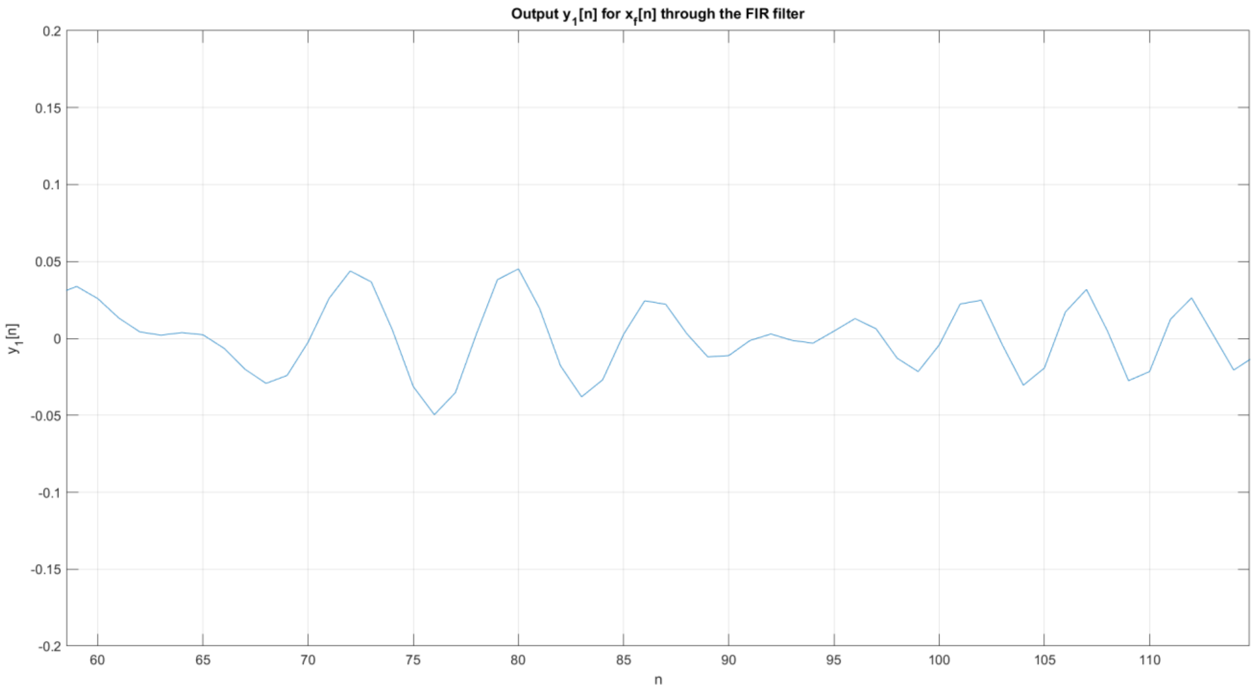


Figure 10: Transient Part of the Output

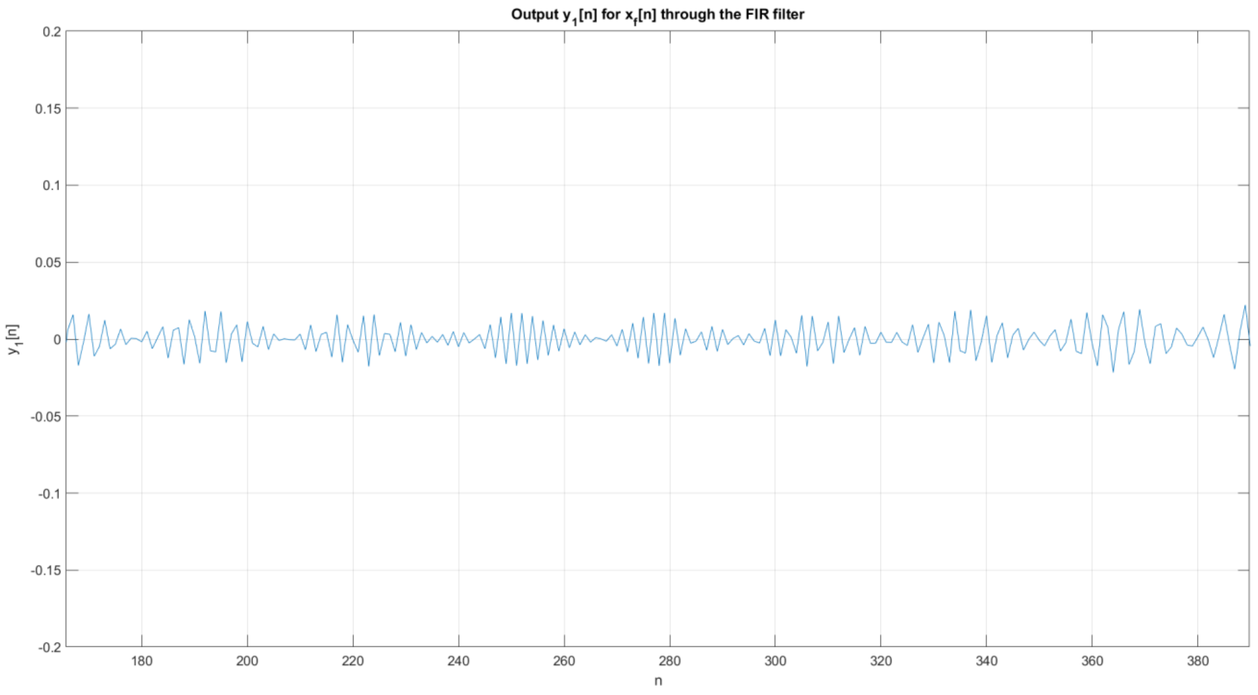


Figure 11: Steady Part of the Output

When the input signal is feed into the filter at n = 0, initially the output depends on fewer input samples. This initial portion where the filter is built up from zero input conditions is the transient.

After some time, once enough input samples have been processed, the output enters a steady-state region that fully reflects the filter's operation on the ongoing input signal.

Plot of $y_2[n]$ is shown in Figure 12. This plot's transient and steady state parts can be observed with the same manner.

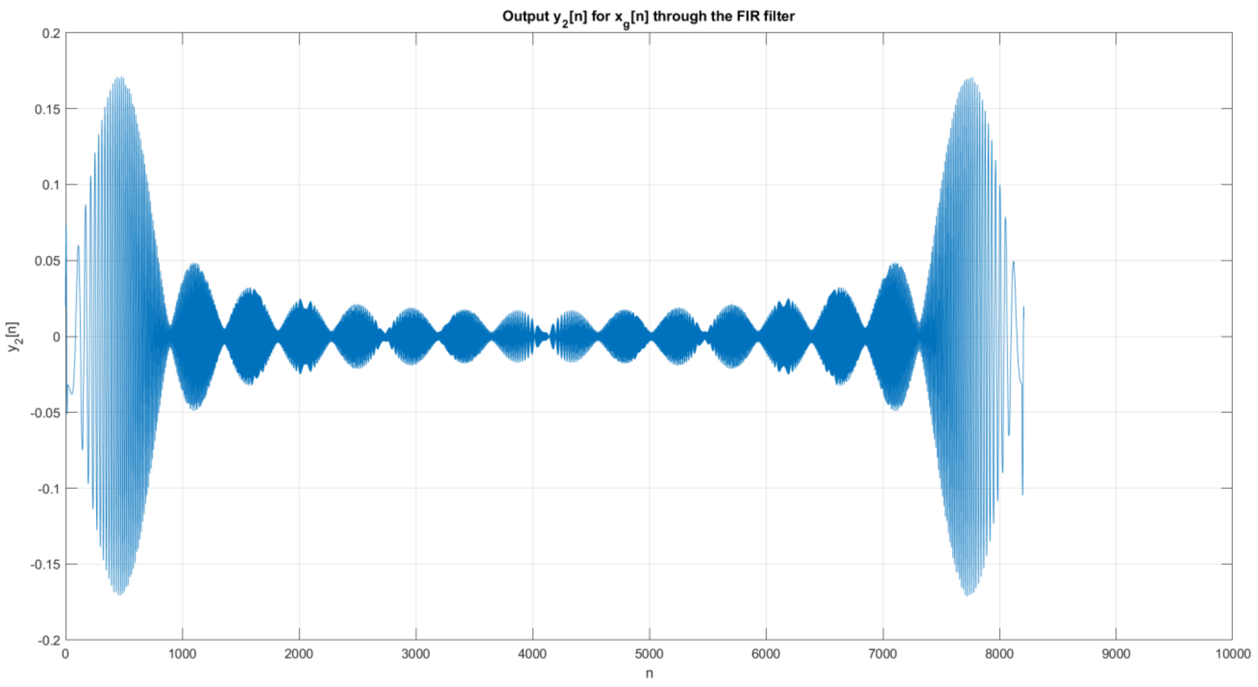


Figure 12: Plot of $y_2[n]$

The code is written to listen x2[n], is listed in Code Listing 5.

Code Listing 5: Question 5

```
alpha = 1000;
N = 8192;
Ts = sqrt(pi/(alpha*N));
Fs = 1/Ts;
n = 0:N-1;
x2 = cos(alpha * (n*Ts).^2);


player = audioplayer(x2, Fs);


% Press Ctrl+C in the Command Window to stop.
while true
    playblocking(player);
end
```

The played-back signal $x_r(t)$ is a time-scaled, band-limited analog version of $x_2[n]$. While $x_2[n]$ was obtained by sampling $x_a(t)$ at intervals Ts, the reconstruction via the sound card's DAC and zero-order hold does not perfectly recreate the original $x_a(t)$. In practice, $x_r(t)$ will approximate $x_a(t)$ if the sampling rate is sufficiently high and the bandwidth of $x_a(t)$ is within the audible range.

Since $x_a(t)$ is not periodic, the output will sound like a chirp sweep. If the sequence is repeated, only the chirp pattern sound from start to finish can be heard repeatedly, not a truly continuous chirp.

The code written for listen to analog version of y2[n], yr(t), is listed in Code Listing 6.

Code Listing 6: Question 6

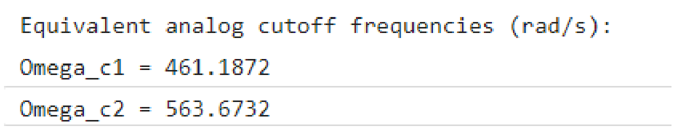
```
load('y2_data.mat','y2_extract');
y2 = y2_extract; % Rename if needed
Fs = 1/Ts;       % Ts from previous step


% Play the sound of y2[n] as y_r(t)
sound(y2, Fs);
```

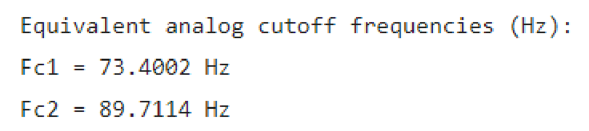
The heard sound is heard quite weak and slightly more wavey.

The analog cutoff frequencies of this signal are found as follows

$$\Omega_{c1} = \frac{\omega_{c1}}{T_s}, \quad \Omega_{c2} = \frac{\omega_{c2}}{T_s}$$



```
Equivalent analog cutoff frequencies (rad/s):
Omega_c1 = 461.1872
Omega_c2 = 563.6732
```



```
Equivalent analog cutoff frequencies (Hz):
Fc1 = 73.4002 Hz
Fc2 = 89.7114 Hz
```

The code is listed in Code Listing 7. Plot of $|H_{eq}(j\omega)|$ is shown in Figure 13 and in Hz is shown in Figure 14.

Code Listing 7: Question 7

```
omega_c1 = pi/11;
omega_c2 = pi/9;


% Compute equivalent analog cutoff frequencies
Omega_c1 = omega_c1 / Ts;
Omega_c2 = omega_c2 / Ts;


disp('Equivalent analog cutoff frequencies (rad/s):');
disp(['Omega_c1 = ', num2str(Omega_c1)]);
disp(['Omega_c2 = ', num2str(Omega_c2)]);


% If you prefer Hz:
Fc1 = Omega_c1/(2*pi);
Fc2 = Omega_c2/(2*pi);
disp('Equivalent analog cutoff frequencies (Hz):');
disp(['Fc1 = ', num2str(Fc1), ' Hz']);
disp(['Fc2 = ', num2str(Fc2), ' Hz']);


% Now, to plot |H_eq(jOmega)|, we start from H(e^{jω}) for ω in [0, π].
```

```
Nfft = 1024;
Hf = fft(h, Nfft);  % frequency response in digital domain
omega = 2*pi*(0:Nfft-1)/Nfft;  % digital frequencies [0, 2π)


% Restrict to [0, π] since FIR freq response is often symmetric
half = 1:(Nfft/2);
omega_half = omega(half);
Hf_half = Hf(half);


% Convert to analog frequencies:
Omega = omega_half / Ts;


% Plot |H_eq(jΩ)|
figure;
plot(Omega, abs(Hf_half));
xlabel('\Omega (rad/s)');
ylabel('|H_{eq}(j\Omega)|');
title('Equivalent Analog Frequency Response');
grid on;


% You can also plot in Hz:
F = Omega/(2*pi);
figure;
plot(F, abs(Hf_half));
xlabel('Frequency (Hz)');
ylabel('|H_{eq}(j\Omega)|');
title('Equivalent Analog Frequency Response in Hz');
grid on;
```

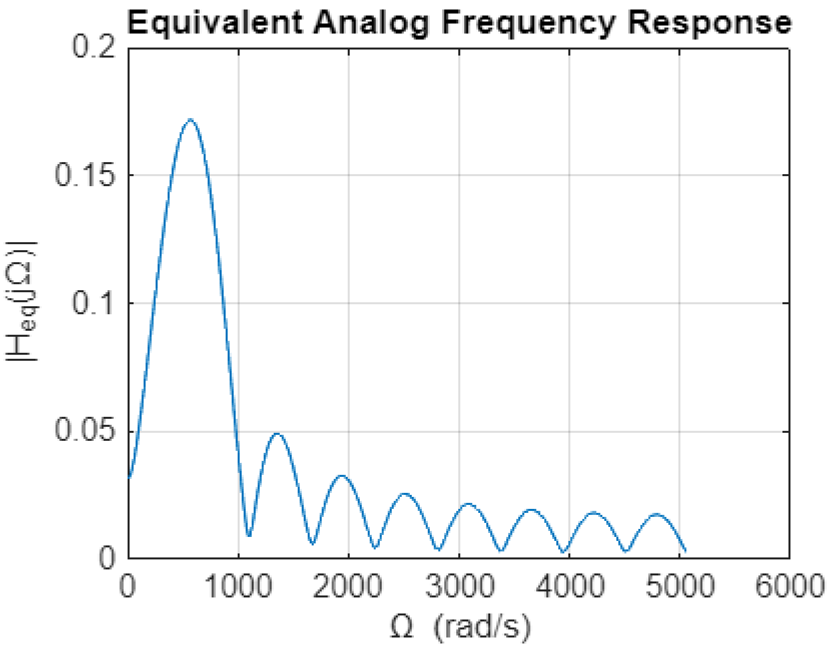


Figure 13: Equivalent Analog Frequency Response

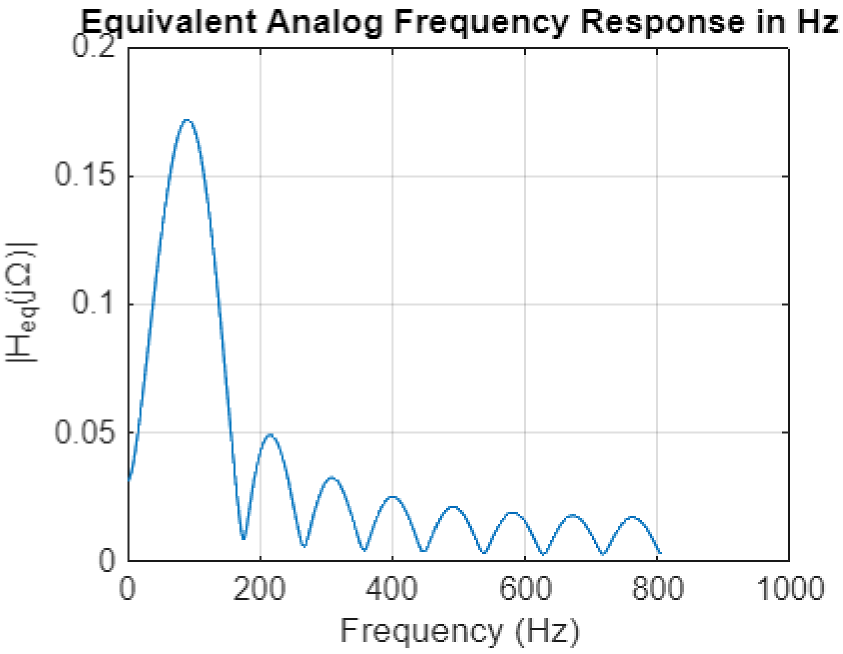


Figure 14: Equivalent Analog Frequency Response in Hz

Written code for Question 8 is given in Code Listing 8.

Code Listing 8: Question 8

```
% Load the music file
[x, Fs] = audioread('sample_music.wav');


% Display sampling rate
disp(['Sampling Rate of the audio file: ', num2str(Fs), ' Hz']);


% If the audio is stereo, take one channel for simplicity
if size(x,2) > 1
    x = x(:,1);  % take the left channel
end


duration = 5;
N = min(length(x), round(duration * Fs));
x_segment = x(1:N);


% Listen to the original segment
disp('Playing original segment...');
sound(x_segment, Fs);  % Listen to the original segment




% Filter the music segment:
y = conv(x_segment, h);


% Listen to the filtered segment
disp('Playing filtered segment...');
sound(y, Fs);
```

```
t_x = (0:N-1)/Fs;           % time vector for original segment
t_y = (0:length(y)-1)/Fs;   % time vector for filtered segment


figure;
subplot(2,1,1);
plot(t_x, x_segment);
xlabel('Time (s)');
ylabel('Amplitude');
title('Original Music Segment');
grid on;


subplot(2,1,2);
plot(t_y, y);
xlabel('Time (s)');
ylabel('Amplitude');
title('Filtered Music Segment');
grid on;
```

The plot of the original sound is shown in Figure 15 and the plot of the filtered sound is shown in Figure 16.

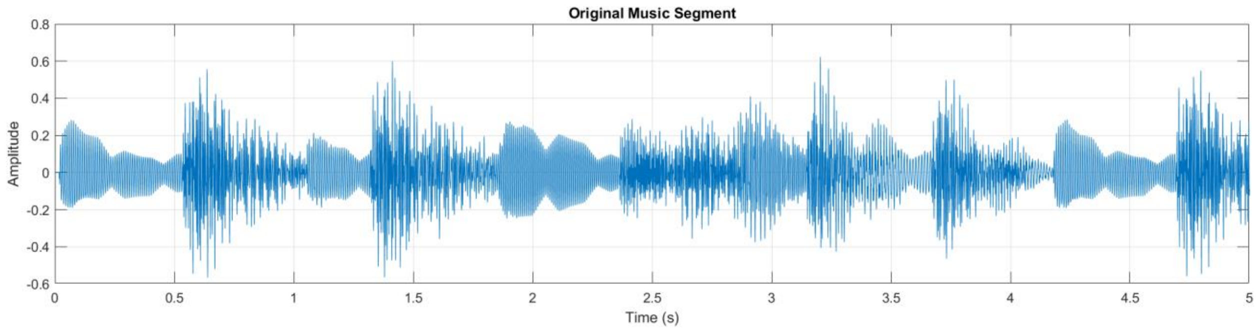


Figure 15: The Plot of the Original Music Segment

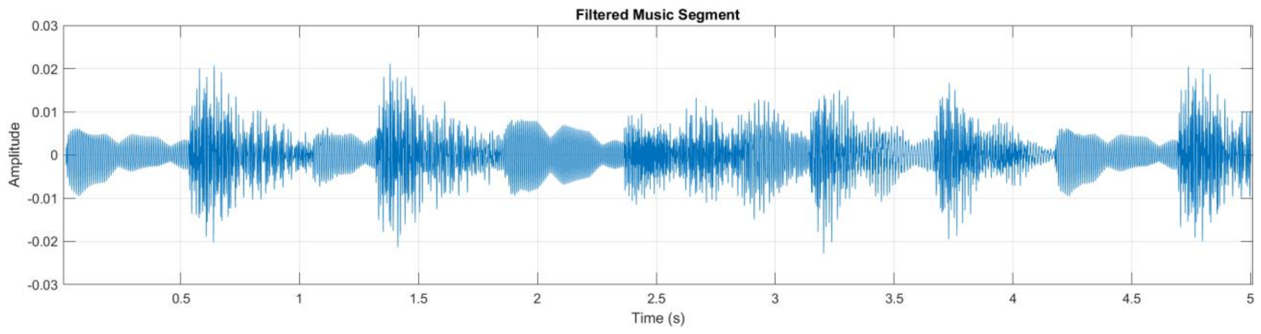


Figure 16: The Plot of the Filtered Music Segment

It can be seen from the figures; the filter decreased the amplitude of the signal while blocking some frequencies.

The code for the question 9 is listed in Code Listing 9.

Code Listing 9: Question 9

```
% Load the recorded voice file
[x, Fs] = audioread('my_voice.m4a');


% Display sampling rate
disp(['Sampling Rate of the voice recording: ', num2str(Fs), ' Hz']);


% If stereo, just use one channel
if size(x,2) > 1
    x = x(:,1);
end


% Select a finite portion of the voice, e.g., first 3 seconds
duration = 3; % seconds
N = min(length(x), round(duration * Fs));
x_segment = x(1:N);
```

```matlab
% Play the original voice segment
disp('Playing original voice segment...');
sound(x_segment, Fs);
pause(duration + 1); % wait until the original finishes playing

% Filter the voice segment
y = conv(x_segment, h);

% Play the filtered voice segment
disp('Playing filtered voice segment...');
sound(y, Fs);

% Save the original and filtered segments
audiowrite('original_voice_segment.wav', x_segment, Fs);
audiowrite('filtered_voice_segment.wav', y, Fs);

% Plot original and filtered signals
t_x = (0:N-1)/Fs;
t_y = (0:length(y)-1)/Fs;

figure;
subplot(2,1,1);
plot(t_x, x_segment);
xlabel('Time (s)');
ylabel('Amplitude');
title('Original Voice Segment');
grid on;

subplot(2,1,2);
```

```
plot(t_y, y);
xlabel('Time (s)');
ylabel('Amplitude');
title('Filtered Voice Segment');
grid on;
```

The recorded voice plot is shown in Figure 17, and the filtered voice plot is shown in Figure 18.

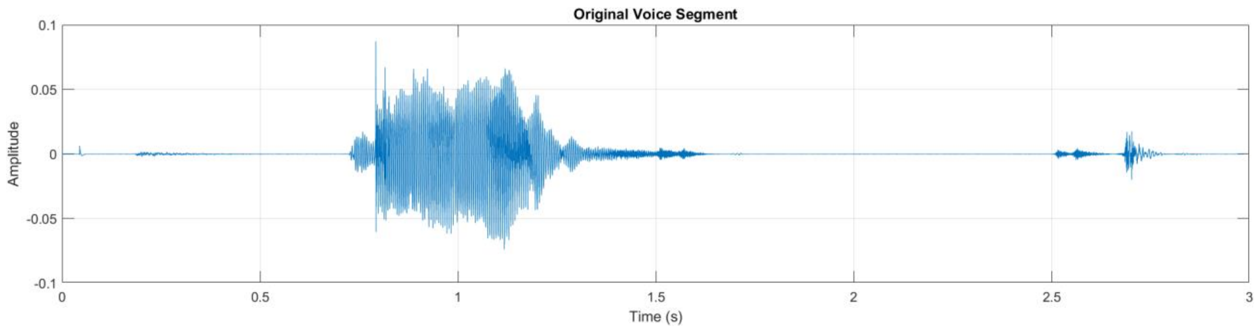


Figure 17: Plot of the Recorded Voice

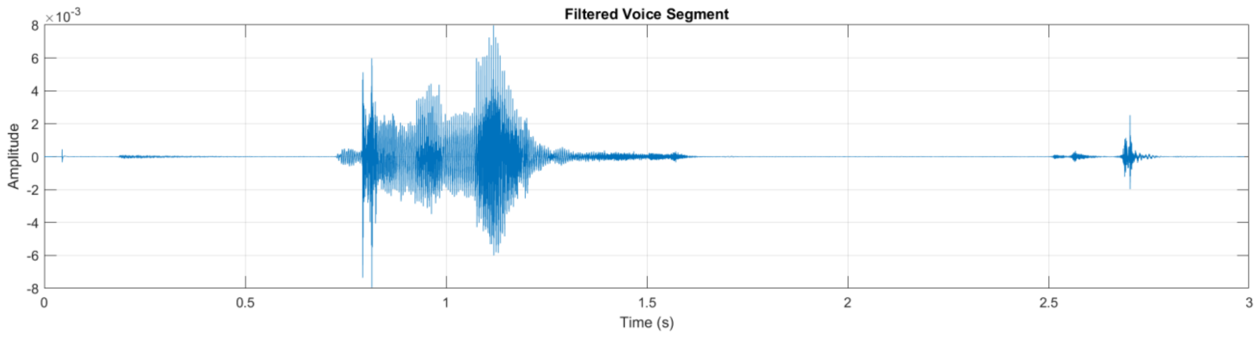


Figure 18: Plot of the Filtered Voice

Conclusion

A bandpass FIR filter was designed, implemented, and tested using parameters derived from the student number. The filter was realized by truncating the ideal infinite impulse response in the time domain, with optional windowing to manage passband and stopband ripples. Analytical computations and MATLAB plots verified that the impulse response matched the desired specifications. Tests with various input signals, including chirps and music clips, demonstrated that the filter passes frequencies within the specified band while attenuating those outside it, confirming compliance with the stability, causality, and bandpass criteria.