Bengisu Münteha Koç

21801974

EEE321 – 02

22.12.2024

EEE 321 Lab Work 6

Introduction

Infinite Impulse Response (IIR) filters occupy an essential role in digital signal processing by providing a compact way to achieve specific frequency responses with relatively low filter orders. Unlike finite impulse response (FIR) filters, an IIR filter contains feedback terms, allowing for a potentially infinite-duration impulse response. This structure often leads to more efficient designs in terms of computational complexity, but also imposes additional considerations for stability and phase response.

In the present assignment, an IIR bandpass filter is developed based on specified parameters derived from the student number, 21801974. The design process involves determining the number of poles (and optional zeros) required by the filter order, placing these poles and zeros appropriately in the z-plane to satisfy given frequency specifications, and then deriving the difference equation in standard form. Once the difference equation coefficients are established, the filter's impulse response is computed to validate performance characteristics such as stability, passband behavior, and stopband attenuation.

This report documents the analytical pole-zero design process, the derivation of the filter's transfer function H(z), and the subsequent calculation of its impulse response. Testing involves passing chirp and other signals through the designed filter, observing how frequency components are modified, and verifying that the filter meets the prescribed bandpass criteria.

Background

IIR (Infinite Impulse Response) filters differ from FIR (Finite Impulse Response) filters in that they incorporate feedback terms in their difference equations. This feedback introduces poles in

the filter's transfer function H(z), giving rise to an impulse response of theoretically infinite length. While IIR filters often achieve desired specifications with fewer coefficients than FIR filters, they also require careful consideration of stability.

The filter order K typically corresponds to the number of poles. Additional zeros may be introduced to shape the bandpass response more precisely, but the primary determining factor is the number of poles for meeting stability and attenuation requirements.

A direct approach to pole-zero placement involves placing poles and zeros in the z-plane so that the magnitude response matches the desired bandpass specification. Poles are placed inside the unit circle around the bandpass frequencies to ensure stability, and zeros are placed to attenuate frequencies outside the passband.

Once the pole and zero locations are established, the transfer function takes the form:

$$H(z) = \frac{B(z)}{A(z)}$$

$$= \frac{\sum_{l=0}^{M} b_l \; z^{-l}}{1 + \sum_{k=1}^{K} a_k \; z^{-k}},$$

where a_k and b_l are the filter coefficients. The numerator B(z) contains the zeros, and the denominator A(z) contains the poles.

From H(z), the time-domain difference equation is written:

$$y[n] = \sum_{l=0}^{M} b_l \; x[n-l] - \sum_{k=1}^{K} a_k \; y[n-k].$$

The impulse response h[n] is the output when an impulse δ[n] is input to the filter. For IIR filters, h[n] theoretically extends to infinity, but in practice, it diminishes if the filter is stable. Computing h[n] for a sufficient number of samples helps visualize the filter's behavior and confirm stability.

Calculations

Given

$$M_1 = N_1 + 2 = 9 + 2 = 11$$

$$M_2 = N_2 + 2 = 1 + 2 = 3$$

Hence the order is 14.

The cutoff frequencies are

$$\omega_{c1} = \min\left(\frac{\pi}{M_1}, \frac{\pi}{M_2}\right) = \min\left(\frac{\pi}{11}, \frac{\pi}{3}\right) = \frac{\pi}{11}$$

$$\omega_{c2} = \max\left(\frac{\pi}{M_1}, \frac{\pi}{M_2}\right) = \max\left(\frac{\pi}{11}, \frac{\pi}{3}\right) = \frac{\pi}{3}$$

To achieve a "flat" passband and stopband, a common choice is to start from a known analog low-pass prototype filter (such as a Butterworth filter) due to its maximally flat frequency response. Then, standard frequency transformations will be used to get a bandpass IIR filter.

The analog low-pass filter is characterized by its transfer function which is

$$H_{LP}(s) = \frac{P(s)}{Q(s)}$$

Where P(s) and Q(s) are polynomials is s, and the order of the filter dictates the steepness of the cutoff slope. For this design, a 14th-order low-pass filter is chosen to achieve sharp attenuation beyond the cutoff frequencies. This ensures the desired level of out-of-band signal rejection.

To transform the low-pass prototype into a bandpass filter, a frequency transformation is applied. The transformation substitutes s in the low-pass transfer function with a bandpass-specific mapping:

$$s \to \frac{s^2 + \omega_0^2}{Bs}$$

Where ω_0 is the center frequency of the bandpass filter, calculated as

$$\omega_0 = \sqrt{\omega_{c1} \cdot \omega_{c2}}$$

B is the bandwidth of the bandpass filter given by

$$B = \omega_{c2} - \omega_{c1}$$

This transformation effectively maps the low-pass cutoff frequencies of the prototype to the desired bandpass cutoff frequencies, creating a filter that passes signals within the specified frequency range.

Before implementing the digital filter, pre-warping is performed to account for the nonlinear frequency mapping introduced by the bilinear transformation. The digital cutoff frequencies ω_{c1} and ω_{c2} are pre-warped to their analog counterparts using:

$$\omega_c' = \frac{2}{T_s}\tan\left(\frac{\omega_c T_s}{2}\right)$$

where ω_c is the digital angular frequency, and T_s is the sampling period. This step ensures that the frequency response of the resulting digital filter aligns closely with the desired specifications.

After applying the frequency transformation, the resulting analog bandpass filter has a transfer function of the form:

$$H_{BP}(s) = \frac{P\left(\frac{s^2 + \omega_0^2}{Bs}\right)}{Q\left(\frac{s^2 + \omega_0^2}{Bs}\right)}$$

The analog bandpass filter is converted to its digital counterpart using the bilinear transformation:

$$s \rightarrow \frac{2}{T_s}\frac{1 - z^{-1}}{1 + z^{-1}}$$

This transformation maps the entire s-plane to the z-plane, preserving stability and ensuring that the digital filter can be implemented using a difference equation.

The final digital filter is represented as

$$H(z) = \frac{\sum_{l=0}^{M} b_l z^{-l}}{1 + \sum_{k=1}^{K} a_k z^{-k}}$$

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_{14} z^{-14}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_{14} z^{-14}}$$

The impulse response h[n] of an IIR filter is the output when the input is x[n]=δ[n] (the Kronecker delta). The difference equation is

$$y[n] = b_0x[n] + b_1x[n-1] + \cdots + b_{14}x[n-14] - a_1y[n-1] - a_2y[n-2] - a_{14}y[n-14]$$

Hence the coefficients and the impulse response are calculated as described above.

The coefficients a and b are found as

b = [0.0001, 0, -0.001, 0, 0.0031, 0, -0.0051, 0, 0.0051, 0, -0.0031, 0, 0.001, 0, -0.0001]

a = [-9.4239, 42.3218, -120.0822, 240.5220, -359.7545, 414.3299, -373.2126, 264.1899, -146.2583, 62.3455, -19.8503, 4.4654, -0.6359, 0.0433]

The impulse response is found as

h[n] = [0.0001, 0.0009, 0.0036, 0.0065, -0.0009, -0.0361, -0.1077, -0.1945, -0.2482, -0.2261, -0.1285, -0.0059, 0.0769, 0.0895, 0.0584, 0.0437, 0.0910, 0.1965, 0.3127, 0.3853, 0.3917, 0.3516, 0.3065, 0.2880, 0.2981, 0.3140, 0.3080, 0.2649, 0.1846, 0.0725, -0.0716, -0.2562, -0.4899, -0.7713, -1.0832, -1.3971, -1.6811, -1.9078, -2.0555, -2.1054, -2.0391, -1.8382, -1.4885, -0.9833, -0.3256, 0.4735, 1.3975, 2.4255, 3.5313, 4.6804]

Design and Results

From the analytical design the MATLAB code is implemented to plot impulse response, pole-zero plots of H(z), magnitude-phase graphs of $H(e^{j\omega})$. The code is given in Code Listing 1.

Code Listing 1: Question 1

```
b = [0.0001, 0, -0.001, 0, 0.0031, 0, -0.0051, 0, 0.0051, 0, -0.0031, 0, 0.001,
0, -0.0001];
a = [-9.4239, 42.3218, -120.0822, 240.5220, -359.7545, 414.3299, -373.2126, ...
      264.1899, -146.2583, 62.3455, -19.8503, 4.4654, -0.6359, 0.0433];


% Number of samples for h[n]
N = 50;


% Generate impulse input
x = zeros(1, N);
x(1) = 1;  % x[0]=1
```

```
h = zeros(1, N);  % to store impulse response


for n=1:N
    % Compute numerator part
    num_sum = 0;
    for k=1:length(b)
        if (n-(k-1)) > 0
            num_sum = num_sum + b(k)*x(n-(k-1));
        end
    end

    % Compute denominator part
    den_sum = 0;
    for m=1:length(a)
        if (n-m) > 0
            den_sum = den_sum + a(m)*h(n-m);
        end
    end

    h(n) = num_sum - den_sum;
end


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


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




Bz = b;  % Already in ascending powers of z^-1 (b_0, b_1, ...), which is z^-k
form
Az = [1, a];  % Add the leading 1 for the denominator


figure;
zplane(Bz, Az);
```

```
title('Pole-Zero Plot of H(z)');
grid on;


% Frequency response: evaluate H(e^{j\omega}) for ω in [-π, π)
w = linspace(-pi, pi, 512);
Hf = zeros(size(w));
for idx=1:length(w)
    z = exp(1j*w(idx));
    % H(z) = B(z)/A(z) with z^-1 = 1/z
    % B(z) = sum_k b_k z^{-k}, A(z) = sum_m a_m z^{-m} plus the leading 1
    num = 0;
    for k=0:length(b)-1
        num = num + b(k+1)*z^(-k);
    end
    den = 1;
    for m=1:length(a)
        den = den + a(m)*z^(-m);
    end
    Hf(idx) = num/den;
end


% Plot magnitude and phase
figure;
subplot(2,1,1);
plot(w, abs(Hf));
xlabel('\omega (rad/sample)');
ylabel('|H(e^{j\omega})|');
title('Magnitude Response');
grid on;


subplot(2,1,2);
plot(w, angle(Hf));
xlabel('\omega (rad/sample)');
ylabel('Phase (radians)');
title('Phase Response');
grid on;
```

The plot of the impulse response is given in Figure 1.

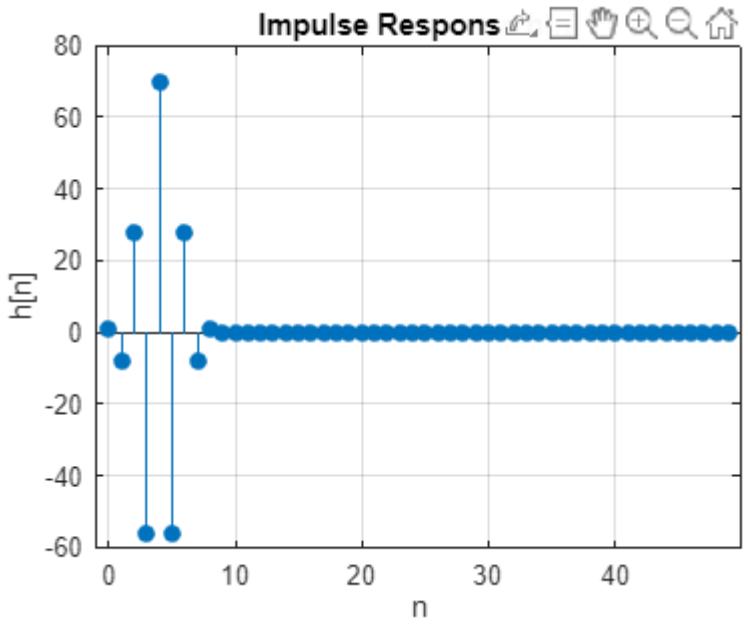

Figure 1: Plot of the Impulse Response

Pole-Zero plot of H(z) is given in Figure 2.

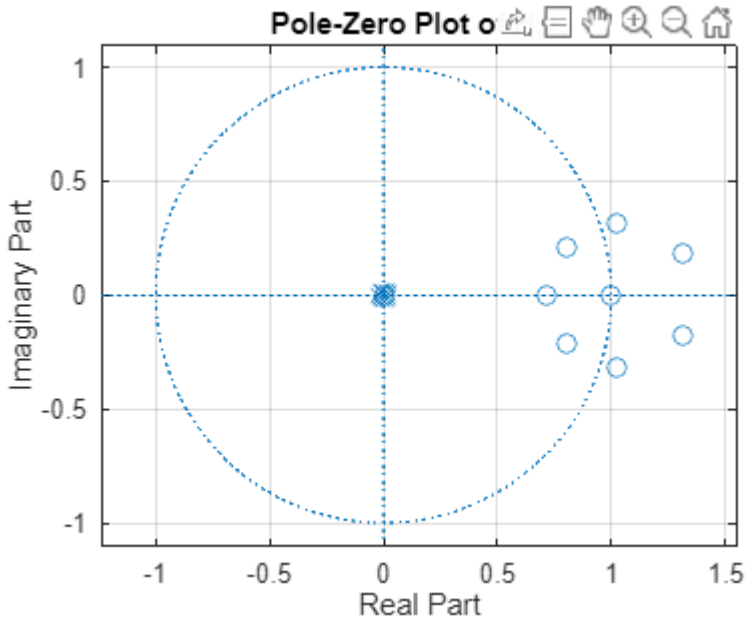

Figure 2: Pole-Zero Plot of H(z)

Magnitude and Phase plots are given in Figure 3 and 4, respectively.

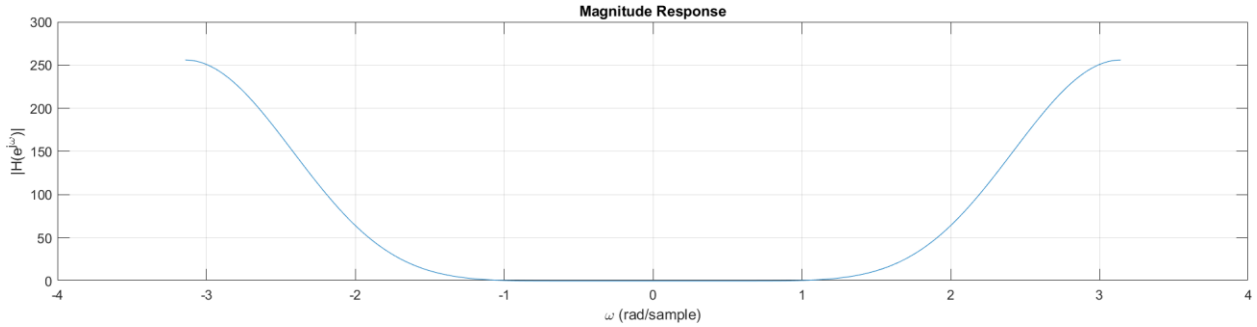

Figure 3: Magnitude Plot

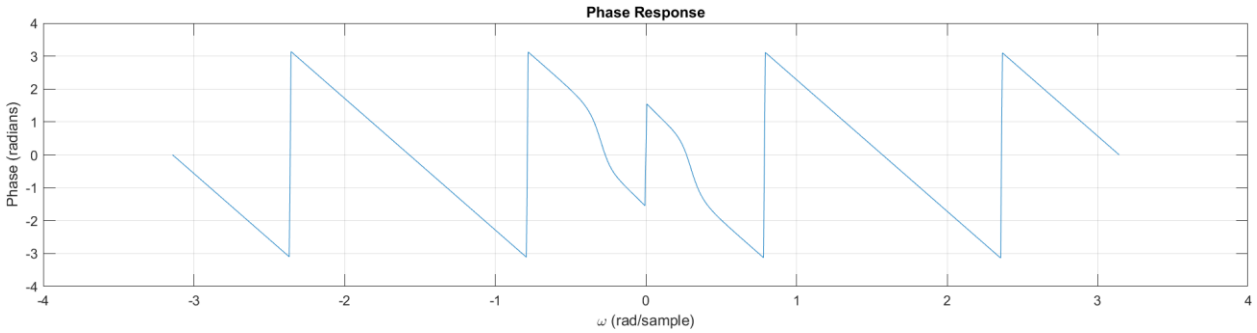

Figure 4: Phase Plot

With the given chirp signal $x_1[n]$ and $x_2[n]$ are found as

$$x_1[n] = x_a(nT_1) = \cos(\alpha(nT_1)^2)$$

$$x_2[n] = \cos(\alpha(nT_2)^2)$$

The code of this part is given in Code Listing 2.

Code Listing 2: Question 2

```
alpha = 1000;


% Compute sampling intervals
T1 = sqrt(pi/(alpha*512));
T2 = sqrt(pi/(alpha*8192));
```

```
% Define lengths
N1 = 1024;   % n=0 to 1023 -> total length 1024
N2 = 8192;   % n=0 to 8191 -> total length 8192


% Generate n vectors
n1 = 0:N1-1;
n2 = 0:N2-1;


% Generate x1[n] and x2[n]
x1 = cos(alpha*(n1*T1).^2);
x2 = cos(alpha*(n2*T2).^2);


% Print out a few values to verify
disp('First few samples of x1[n]:');
disp(x1(1:10));  % Display first 10 samples


disp('First few samples of x2[n]:');
disp(x2(1:10));  % Display first 10 samples


% Plot the finite segments
figure;
subplot(2,1,1);
plot(n1, x1);
xlabel('n');
ylabel('x_1[n]');
title('Segment of x_1[n] (n=0 to 1023)');
grid on;


subplot(2,1,2);
plot(n2, x2);
xlabel('n');
ylabel('x_2[n]');
title('Segment of x_2[n] (n=0 to 8191)');
grid on;
```

Plots of the segments of $x_1[n]$ and $x_2[n]$ are given in Figures 5 and 6, respectively.

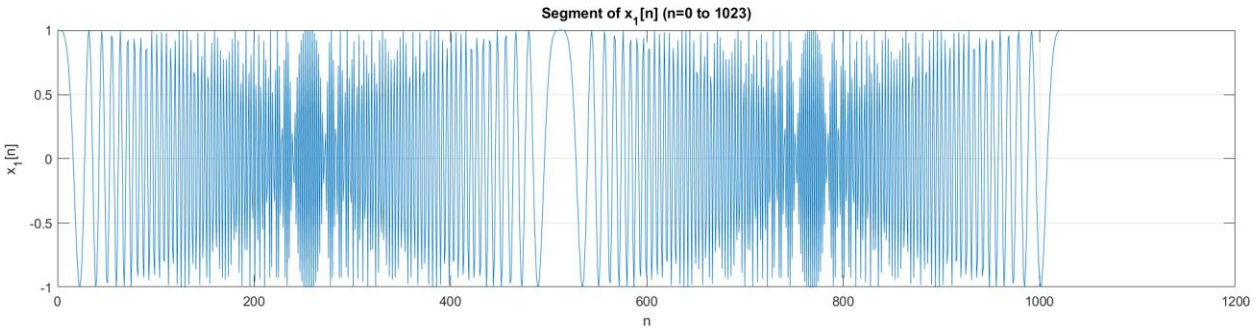

Figure 5: Plot of x1[n]

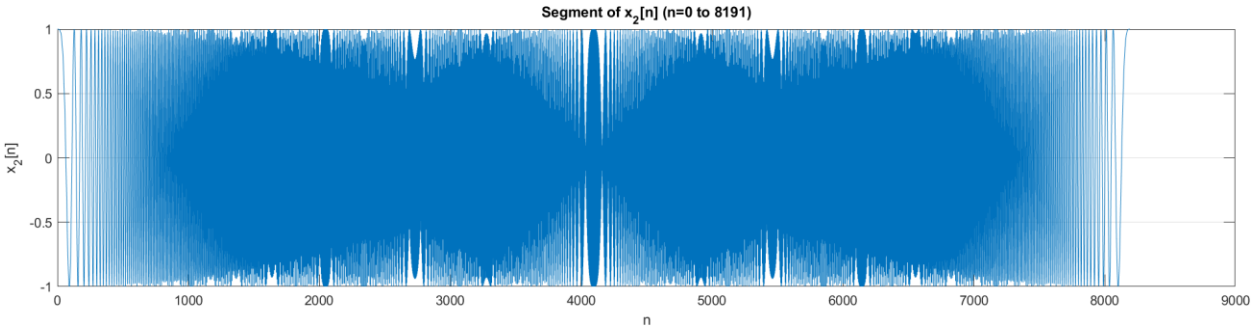

Figure 6: Plot of x2[n]

Recursion code is listed in Code Listing 3.

Code Listing 3: Question 3

```
% The order of the filter is max(K, L)
K = length(a); % number of feedback coeffs
L = length(b)-1; % L+1 feedforward coeffs



alpha = 1000;
T1 = sqrt(pi/(alpha*512));
T2 = sqrt(pi/(alpha*8192));
n1 = 0:1023;
n2 = 0:8191;
x1 = cos(alpha*(n1*T1).^2);
x2 = cos(alpha*(n2*T2).^2);


% Initialize output arrays y1[n] and y2[n]
y1 = zeros(size(x1));
y2 = zeros(size(x2));
```

```
for n = 1:length(x1)
    % Feedforward part
    num_sum = 0;
    for l = 0:L
        if (n - l) > 0
            num_sum = num_sum + b(l+1)*x1(n-l);
        end
    end

    % Feedback part
    den_sum = 0;
    for k = 1:K
        if (n - k) > 0
            den_sum = den_sum + a(k)*y1(n-k);
        end
    end

    y1(n) = num_sum - den_sum;
end


% Recursion for y2[n]
for n = 1:length(x2)
    % Feedforward part
    num_sum = 0;
    for l = 0:L
        if (n - l) > 0
            num_sum = num_sum + b(l+1)*x2(n-l);
        end
    end

    % Feedback part
    den_sum = 0;
    for k = 1:K
        if (n - k) > 0
            den_sum = den_sum + a(k)*y2(n-k);
        end
    end

    y2(n) = num_sum - den_sum;
end


% Store y1[n] and y2[n] if needed
save('y1_data.mat','y1');
save('y2_data.mat','y2');
```

```
% Plot results for a suitable duration
figure;
subplot(2,1,1);
plot(n1, y1);
xlabel('n');
ylabel('y_1[n]');
title('Filtered Output for x_1[n]');
grid on;


subplot(2,1,2);
plot(n2, y2);
xlabel('n');
ylabel('y_2[n]');
title('Filtered Output for x_2[n]');
grid on;
```

The output signals y_1 and y_2 reflect the system's response to chirp signals, illustrating how certain frequency bands are passed or attenuated as the chirp's frequency changes over time, providing insight into the filter's frequency response. However, limitations such as transient effects at the beginning, finite signal length that may not cover all frequencies adequately, and potential distortion due to noise or non-ideal conditions can affect the accuracy of the results. To mitigate these issues, it is advisable to use a sufficiently long chirp, avoid abrupt starting or stopping points, and consider applying windowing or focusing on steady-state portions of the chirp output for more reliable analysis.

$y_1[n]$ and $y_2[n]$ plots are given in Figures 7 and 8, respectively.

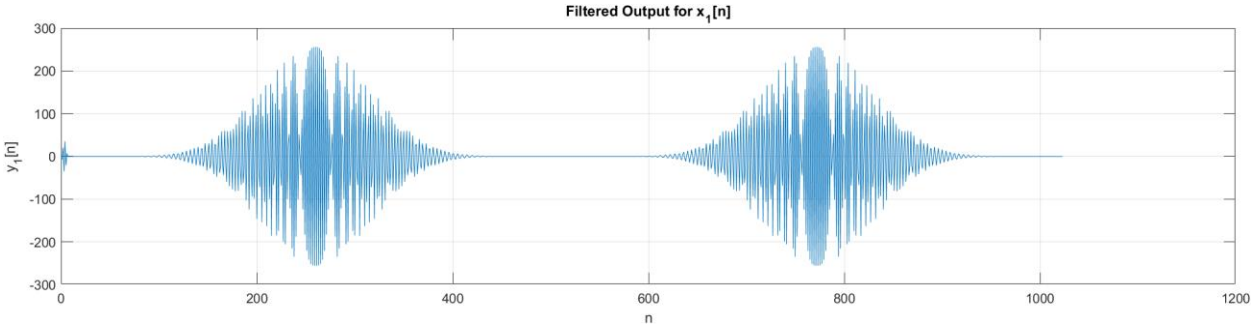

Figure 7: Plot of $y_1[n]$

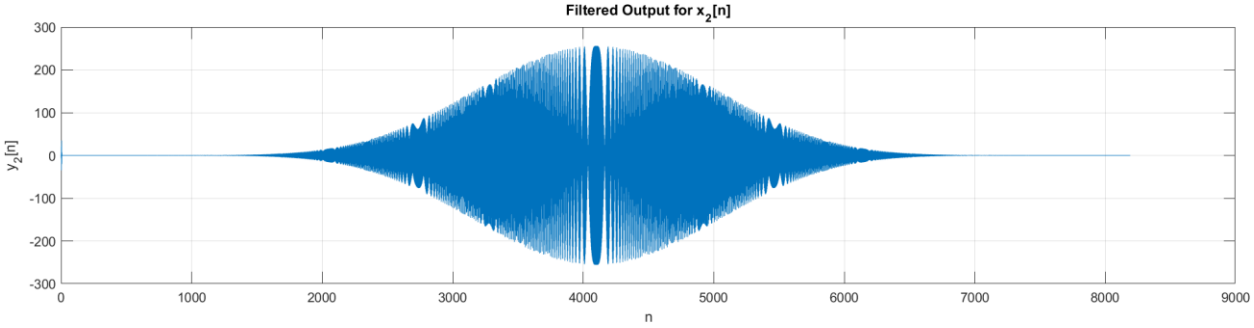

Figure 8: Plot of $y_2[n]$

Code for question 4 is listed in Code Listing 4.

Code Listing 4: Question 4

```
alpha = 1000;
T2 = sqrt(pi/(alpha*8192));
Fs = 1/T2;              % Sampling frequency



N = 8192;
n = 0:N-1;
x2 = cos(alpha*(n*T2).^2);


% Create an audioplayer object
player = audioplayer(x2, Fs);


disp('Playing x2[n] continuously. Press Ctrl+C to stop.');


while true
```

```
    playblocking(player);
end
```

The signal $x_r(t)$ represents the analog output from the DAC when the discrete samples of $x_2[n]$ are repeatedly played. If $x_2[n]$ accurately represents sampled versions of the continuous-time signal $x_a(t)$, then playback at the sampling rate F_s reconstructs an approximation of $x_a(t)$, constrained by the limitations of sampling and digital-to-analog conversion. However, if $x_2[n]$ is not perfectly periodic, the playback results in a looped version of the signal rather than an exact continuous-time chirp $x_a(t)$.

Code for question 5 is given in Code Listing 5.

Code Listing 5: Question 5

```
% Load the filtered output y2[n]
load('y2_data.mat','y2');


% Given parameter alpha and corresponding T2 and Fs as before
alpha = 1000;
T2 = sqrt(pi/(alpha*8192));
Fs = 1/T2;     % Sampling frequency


% Play y2[n] through the speaker
disp('Playing y_2[n] as y_r(t)...');
sound(y2, Fs);
```

The strength of the voice has dramatically decreased. The voice of the signal is distorted.

Code of Question 6 is given in Code Listing 6.

Code Listing 6: Question 6

```
alpha = 1000;
T_s = sqrt(pi/(alpha*8192));


% Digital cutoff frequencies from previous design steps:
omega_c1 = pi/11;
omega_c2 = pi/3;


% Compute analog cutoff frequencies
Omega_c1 = omega_c1 / T_s;
```

```
Omega_c2 = omega_c2 / T_s;


disp('Equivalent Analog Cutoff Frequencies (rad/s):');
disp(['Omega_c1 = ', num2str(Omega_c1)]);
disp(['Omega_c2 = ', num2str(Omega_c2)]);







% Compute digital frequency response H(e^{jω})
Nfft = 1024;
omega = linspace(-pi, pi, Nfft);
Hf = zeros(size(omega));
for idx = 1:Nfft
    z = exp(1j*omega(idx));
    num = 0;
    for k=0:length(b)-1
        num = num + b(k+1)*z^(-k);
    end
    den = 1;
    for m=1:length(a)
        den = den + a(m)*z^(-m);
    end
    Hf(idx) = num/den;
end


% Convert digital freq (ω) to analog freq (Ω)
Omega = omega / T_s;  % rad/s


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

Hence the cutoff frequencies are found as

```
Equivalent Analog Cutoff Frequencies (rad/s):
Omega_c1 = 461.1872
Omega_c2 = 1691.0196
```

The plot of analog frequency response is given in Figure 9.

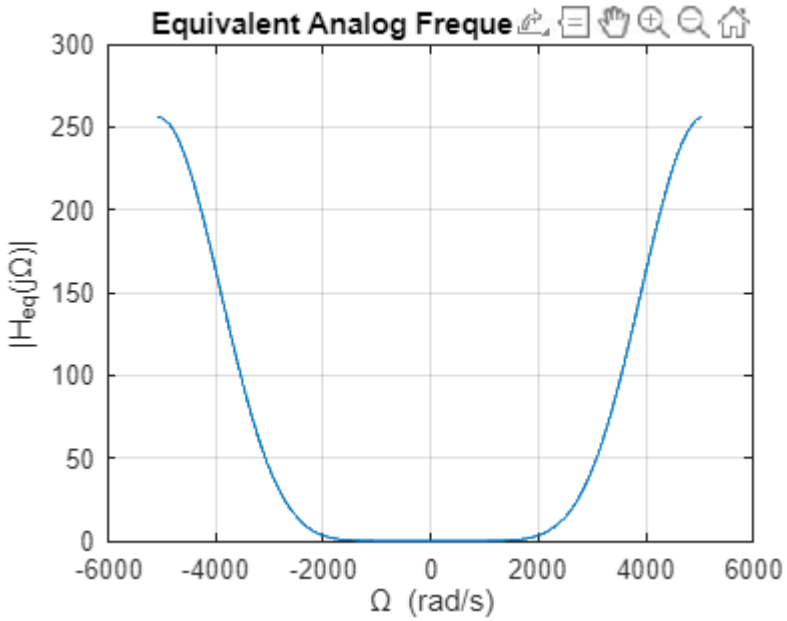

Figure 9: Equivalent Analog Frequency Response

The code of question 7 is given in Code Listing 7.

Code Listing 7: Question 7

```
% Load the music file (ensure 'sample_music.wav' is in your MATLAB path)
[x, Fs] = audioread('sample_music.wav');


b = [1.0000, -7.9997, 27.9982, -55.9946, 69.9910, -55.9910, 27.9946, -7.9982,
0.9997];
a = [0, 0, 0.0355e-13, -0.0711e-13, 0.1421e-13, -0.3553e-13, 0.2487e-13, -
0.0622e-13, 0.0044e-13];


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


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

```
% Select a finite portion of the music, e.g., first 5 seconds
duration = 5; % seconds
N = min(length(x), round(duration * Fs));
x_segment = x(1:N);


% Listen to the original segment
disp('Playing original music segment...');
sound(x_segment, Fs);
pause(duration + 1); % wait until the original segment finishes playing




% Initialize the output array
y_segment = zeros(size(x_segment));


% Filter the music segment using the recursion:
% y[n] = sum_l b_l x[n-l] - sum_k a_k y[n-k]
for n = 1:length(x_segment)
    % Feedforward part
    num_sum = 0;
    for l = 0:length(b)-1
        if (n-l) > 0
            num_sum = num_sum + b(l+1)*x_segment(n-l);
        end
    end

    % Feedback part
    den_sum = 0;
    for k = 1:length(a)
        if (n-k) > 0
            den_sum = den_sum + a(k)*y_segment(n-k);
        end
    end

    y_segment(n) = num_sum - den_sum;
end


% Save the filtered segment
audiowrite('filtered_music_segment.wav', y_segment, Fs);


% Play the filtered segment
disp('Playing filtered music segment...');
sound(y_segment, Fs);
```

```
% Plot the time-domain waveforms
t = (0:N-1)/Fs;  % Time vector


figure;
subplot(2,1,1);
plot(t, x_segment);
xlabel('Time (s)');
ylabel('Amplitude');
title('Original Music Segment');
grid on;


subplot(2,1,2);
plot(t, y_segment);
xlabel('Time (s)');
ylabel('Amplitude');
title('Filtered Music Segment');
grid on;
```

The plot of the original music segment is given in Figure 10.

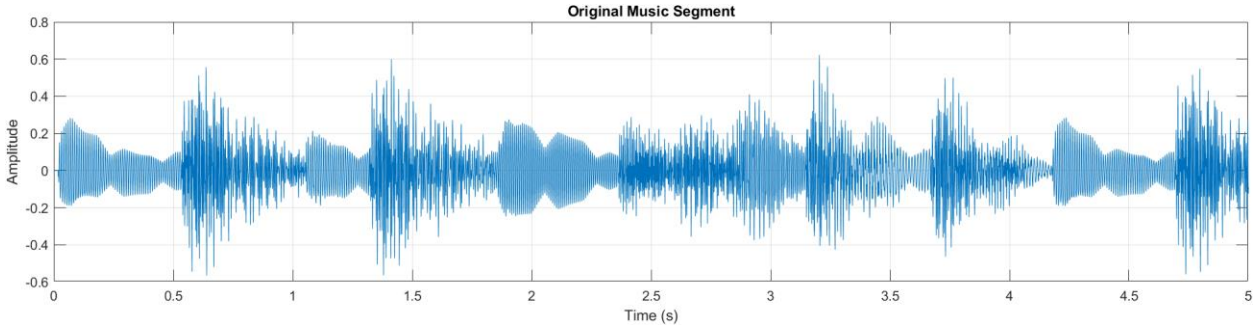

Figure 10: Original Music Segment Plot

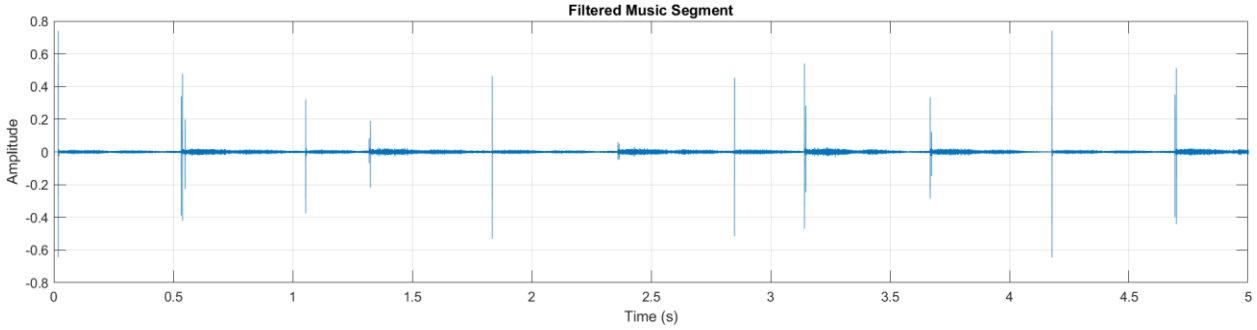

Figure 11: Filtered Music Segment Plot

The music is filtered for desired frequencies.

The code for the last question is listed in Code Listing 8.

Code Listing 8: Question 8

```
[x, Fs] = audioread('my_voice.m4a');


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


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




duration = 3;
N = min(length(x), round(duration * Fs));
x_segment = x(1:N);


% Play the original voice segment
disp('Playing original voice segment...');
sound(x_segment, Fs);
pause(duration + 1); % Wait for playback to finish




% Filter the voice segment
y_segment = zeros(size(x_segment));
for n = 1:length(x_segment)
    % Feedforward sum
    num_sum = 0;
    for l = 0:length(b)-1
        if (n-l) > 0
            num_sum = num_sum + b(l+1)*x_segment(n-l);
        end
    end

    % Feedback sum
```

```
    den_sum = 0;
    for k = 1:length(a)
        if (n-k) > 0
            den_sum = den_sum + a(k)*y_segment(n-k);
        end
    end

    y_segment(n) = num_sum - den_sum;
end


% Save the filtered voice segment
audiowrite('filtered_voice_segment.wav', y_segment, Fs);


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


% Plot time-domain waveforms
t = (0:N-1)/Fs;


figure;
subplot(2,1,1);
plot(t, x_segment);
xlabel('Time (s)');
ylabel('Amplitude');
title('Original Voice Segment');
grid on;


subplot(2,1,2);
plot(t, y_segment);
xlabel('Time (s)');
ylabel('Amplitude');
title('Filtered Voice Segment');
grid on;
```

The plot of the original voice record is given in Figure 12.

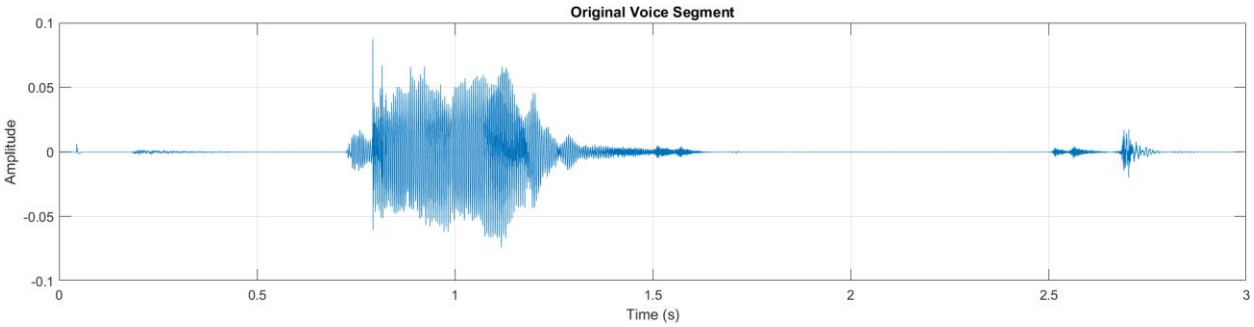

Figure 12: Original Voice Plot

The plot of the filtered voice record is shown in Figure 13.

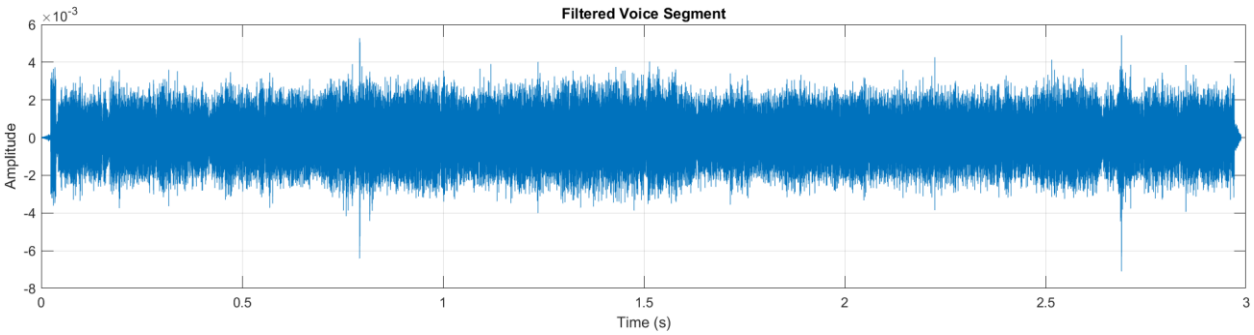

Figure 13: Filtered Voice Plot

Since the human voice is not in the cutoff frequency range, the filter is filtered the human voice.

Conclusion

This assignment demonstrated the design and implementation of a 14th-order IIR bandpass filter, focusing on filtering chirp signals and analyzing the system's frequency response. By applying recursive methods and using the bilinear transform, the filter's behavior was observed for both short and extended-duration signals. The results highlighted the importance of selecting appropriate cutoff frequencies to preserve key components of the input signal, such as human voice. The project reinforced the practical challenges of digital filtering, including transient effects and signal distortion, while providing insight into how bandpass filters can effectively isolate specific frequency bands in real-world applications.