Computational Assignment 1

Mahdi Tabatabaei 400101515
github repository

Digital Signal Processing

Dr. Shamsollahi

November 9, 2024

Digital Signal Processing

Computational Assignment 1

Mahdi Tabatabaei 400101515
github repository

Question 1

Consider the following difference equation of the system:

$$y[n] - 0.5y[n-1] + 0.25y[n-2] = x[n] + 2x[n-1] + x[n-3]$$

(a) Without using the Z-transform, determine and display the system's impulse response for $0 \leq n \leq 100$. You can use the recursive method for this task.

```
% Parameters
N = 100;                    % Number of points
y = zeros(1, N+1);          % Initialize output y[n] array
x = zeros(1, N+1);          % Initialize input x[n] array
x(1) = 1;                   % Impulse input (x[0] = 1)

% Recursive calculation based on difference equation
for n = 1:N
    if n == 1
        y(n) = x(n);
    elseif n == 2
        y(n) = x(n) + 2*x(n-1) + 0.5*y(n-1);
    elseif n == 3
        y(n) = x(n) + 2*x(n-1) + 0.5*y(n-1) - 0.25*y(n-2);
    else
        y(n) = x(n) + 2*x(n-1) + x(n-3) + 0.5*y(n-1) - 0.25*y(n-2);
    end
end

% Plotting the result
n = 0:N;
stem(n, y, 'filled');
xlabel('n', 'Interpreter', 'latex');
ylabel('y[n]', 'Interpreter', 'latex');
title('System Response y[n] for Given Difference Equation', '
Interpreter', 'latex');
grid on;

```

(b) First, using the Z-transform, find the impulse response of the system and explain the system's stability. (After manually obtaining the Z-transform, you can define the system using the tf command for the Z-transform and display its impulse response. To do this, make sure to thoroughly study the relevant material on this command.)

```
% Define the numerator and denominator coefficients for H(z)
numerator = [1 2 0 1];      % Coefficients of z^0, z^-1, z^-2, z^-3
denominator = [1 -0.5 0.25]; % Coefficients of z^0, z^-1, z^-2

```

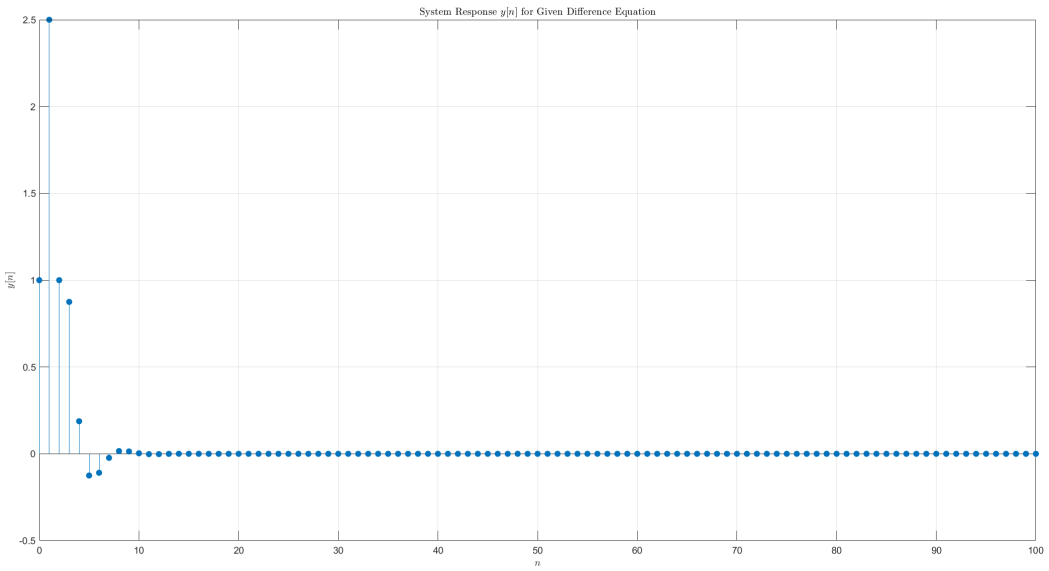

Figure 1: Impulse Response (Recursive Method)

```
% Define the transfer function using tf
Hz = tf(numerator, denominator, -1); % -1 indicates Z-transform

% Display the impulse response using impz
N = 100; % Length of impulse response
impulse_response = impz(numerator, denominator, N);

% Plot the impulse response
n = 0:N-1;
stem(n, impulse_response, 'filled');
xlabel('n', 'Interpreter', 'latex');
ylabel('h[n]', 'Interpreter', 'latex');
title('Impulse Response h[n] of the System', 'Interpreter', 'latex');
grid on;

% Find the poles of the transfer function
poles = roots(denominator);

% Display poles and assess stability
disp('Poles of the system:');
disp(poles);

if all(abs(poles) < 1)
    disp('The system is stable because all poles are inside the unit circle.');
else
    disp('The system is unstable because at least one pole is outside the unit circle.');
end
```

(c) Use the Z-transform to find the system's response to the following input.

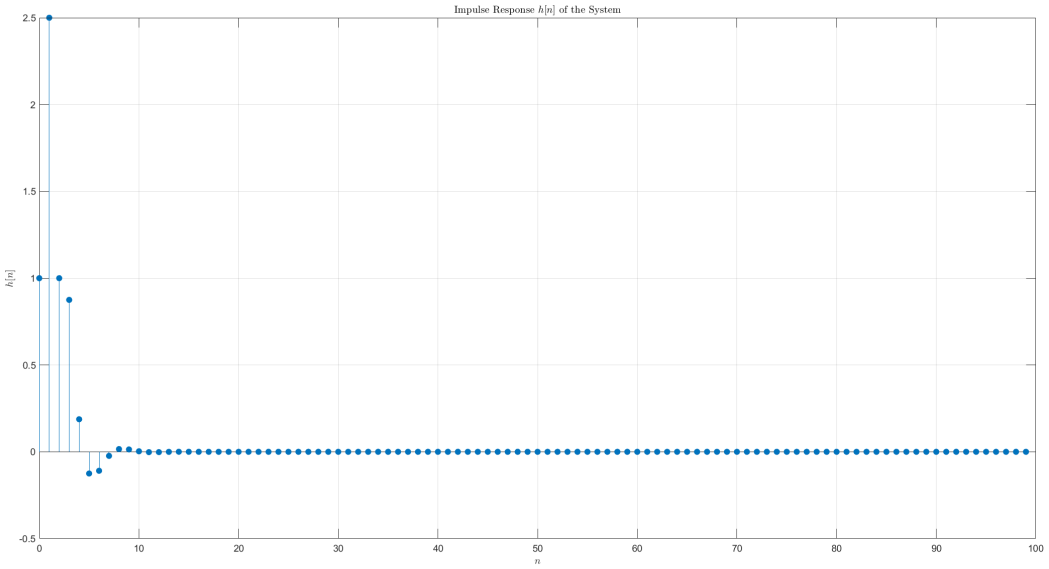

Figure 2: Impulse Response (Z Transform)

Poles of the system:

$$0.2500 + 0.4330i$$

$$0.2500 - 0.4330i$$

The system is stable because all poles are inside the unit circle and system is causal.

```
% Define the numerator and denominator coefficients for H(z)
numerator = [1 2 0 1];      % Coefficients of z^0, z^-1, z^-2, z^-3
denominator = [1 -0.5 0.25]; % Coefficients of z^0, z^-1, z^-2

% Define the transfer function using tf
Hz = tf(numerator, denominator, -1); % -1 indicates Z-transform

% Display the impulse response using impz
N = 100; % Length of impulse response
impulse_response = impz(numerator, denominator, N);

% Plot the impulse response
n = 0:N-1;
stem(n, impulse_response, 'filled');
xlabel('$n$', 'Interpreter', 'latex');
ylabel('$h[n]$', 'Interpreter', 'latex');
title('Impulse Response $h[n]$ of the System', 'Interpreter', 'latex');
grid on;

% Find the poles of the transfer function
poles = roots(denominator);

% Display poles and assess stability
disp('Poles of the system:');
disp(poles);

if all(abs(poles) < 1)
```

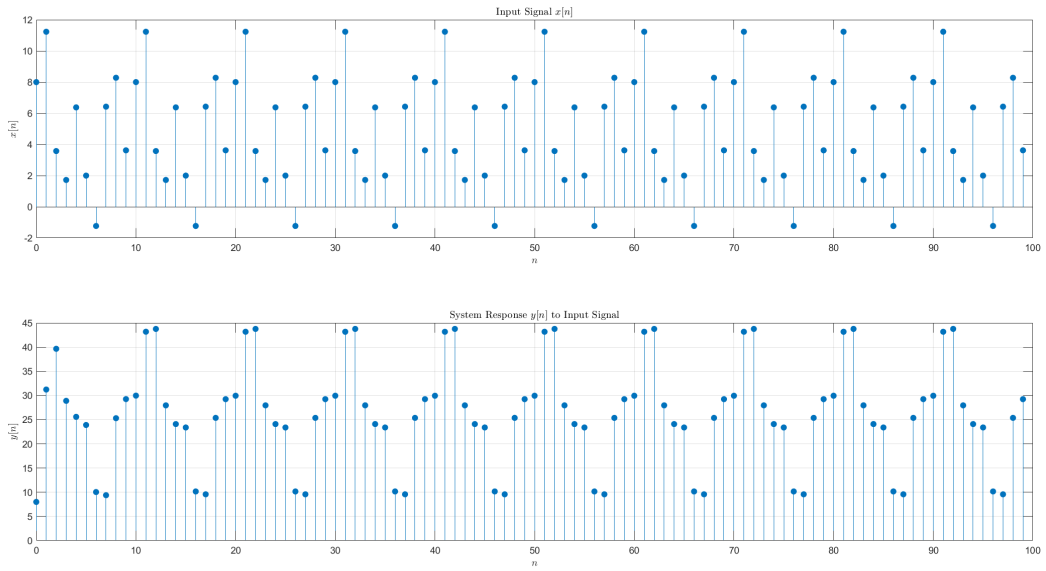

Figure 3: Response of Input

```
28          disp('The system is stable because all poles are inside the unit
    circle.');
29      else
30          disp('The system is unstable because at least one pole is outside
     the unit circle.');
31      end
32
```

(d) Plot the amplitude of frequency response of the system in the range $[-\pi, \pi]$ using the `freqz` command.

```
1   % Define the frequency range from -pi to pi
2   num_points = 512; % Number of points for better resolution
3   [H, w] = freqz(numerator, denominator, num_points, 'whole'); % 'whole
    ' covers -pi to pi
4
5   % Shift the frequency and response for plotting from -pi to pi
6   w = w - pi;
7   H = fftshift(H); % Shift zero frequency to the center
8
9   % Plot the magnitude of the frequency response
10  figure;
11  plot(w, abs(H), LineWidth=2);
12  xlabel('Frequency (radians/sample)');
13  ylabel('Magnitude');
14  title('Magnitude of Frequency Response |H(e^{j\omega})|');
15  grid on;
16
```

(e) Draw the frequency response of the system in the wrapped form (wrapped: $[-\pi, \pi]$) and in the unwrapped form (unwrapped: $[-\pi, \pi]$).

```
1   % Define the frequency range from -pi to pi
2   num_points = 512; % Number of points for better resolution
```

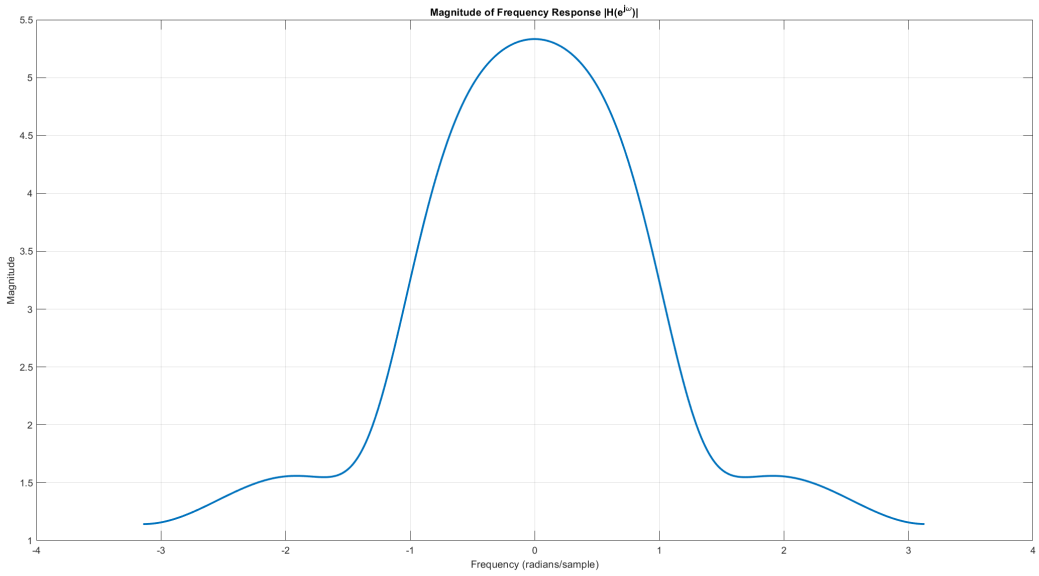

Figure 4: Amplitude of Frequency Response

```
    [H, w] = freqz(numerator, denominator, num_points, 'whole'); % 'whole
' covers -pi to pi

    % Shift the frequency and response for plotting from -pi to pi
    w = w - pi;
    H = fftshift(H); % Shift zero frequency to the center

    % Calculate the phase response (wrapped and unwrapped)
    phase_wrapped = angle(H);           % Wrapped phase
    phase_unwrapped = unwrap(angle(H)); % Unwrapped phase

    % Plot the wrapped phase response
    figure;
    subplot(2, 1, 1);
    plot(w, phase_wrapped, LineWidth = 2);
    xlabel('Frequency (radians/sample)');
    ylabel('Phase (radians)');
    title('Wrapped Phase Response');
    grid on;

    % Plot the unwrapped phase response
    subplot(2, 1, 2);
    plot(w, phase_unwrapped, LineWidth = 2);
    xlabel('Frequency (radians/sample)');
    ylabel('Phase (radians)');
    title('Unwrapped Phase Response');
    grid on;

```

(f) Calculate and plot the zeros and poles of the system using the `tf2zp` command and plot them using the `pzplot` command.

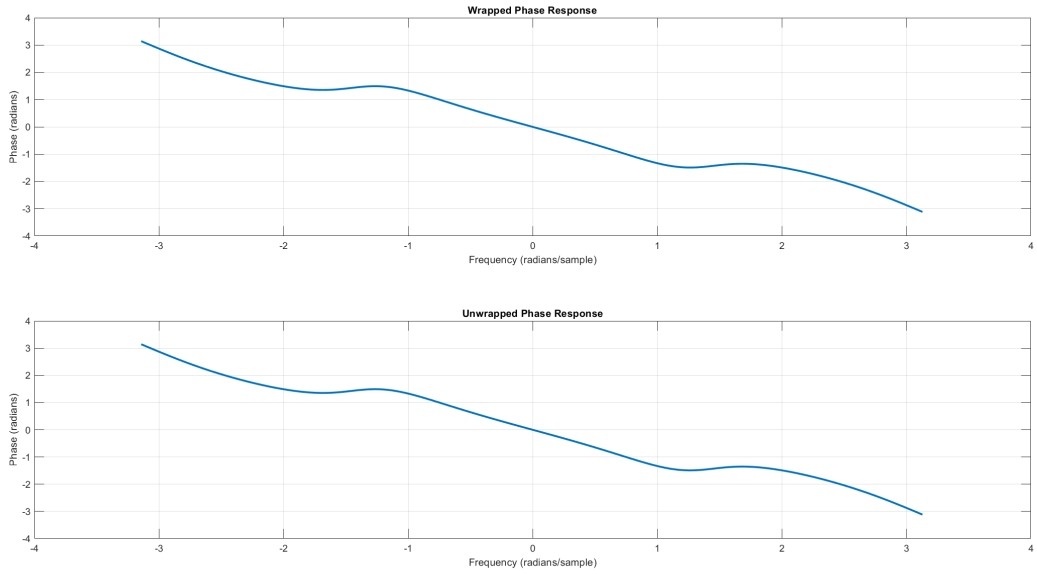

Figure 5: Phase of Frequency Response

Zeros of the system:

$$-2.2056 + 0.0000i$$

$$0.1028 + 0.6655i$$

$$0.1028 - 0.6655i$$

Poles of the system:

$$0.0000 + 0.0000i$$

$$0.2500 + 0.4330i$$

$$0.2500 - 0.4330i$$

Gain of the system:

$$1$$

```
% Compute the zeros, poles, and gain using tf2zp
[zeros, poles, gain] = tf2zpk(numerator, denominator);

% Display the zeros, poles, and gain for reference
disp('Zeros of the system:');
disp(zeros);
disp('Poles of the system:');
disp(poles);
disp('Gain of the system:');
disp(gain);

% Plot the zeros and poles using pzplot
figure;
pzplot(tf(numerator, denominator, -1)); % -1 indicates Z-domain
title('Pole-Zero Plot of the System');
grid on;

```

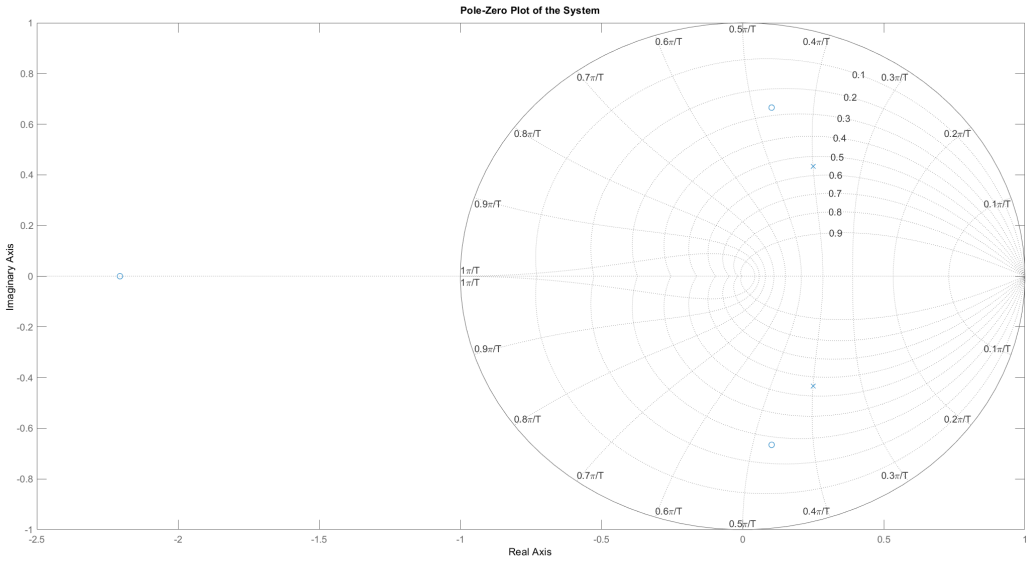

Figure 6: Poles and Zeros

Question 2

Consider the following system:

$$H(z) = \frac{1 - z^{-2}}{1 + 0.9z^{-1} + 0.6 \cdot z^{-2} + 0.05z^{-3}}$$

Using partial fraction decomposition, find the system's impulse response (using the `residue` or `residuez` command).

Residues:

$$1.5880 - 0.1409i$$

$$1.5880 + 0.1409i$$

$$-2.1760 + 0.0000i$$

Poles:

$$-0.4022 + 0.6011i$$

$$-0.4022 - 0.6011i$$

$$-0.0956 + 0.0000i$$

```
% Define the transfer function coefficients
numerator = [1 0 -1];        % Coefficients of 1 - z^(-2)
denominator = [1 0.9 0.6 0.05]; % Coefficients of 1 + 0.9z^(-1) + 0.6z^(-2) +
    0.05z^(-3)

% Use residuez to get the partial fraction expansion
[residues, poles, direct_term] = residuez(numerator, denominator);

% Display the results
disp('Residues:');
disp(residues);
```

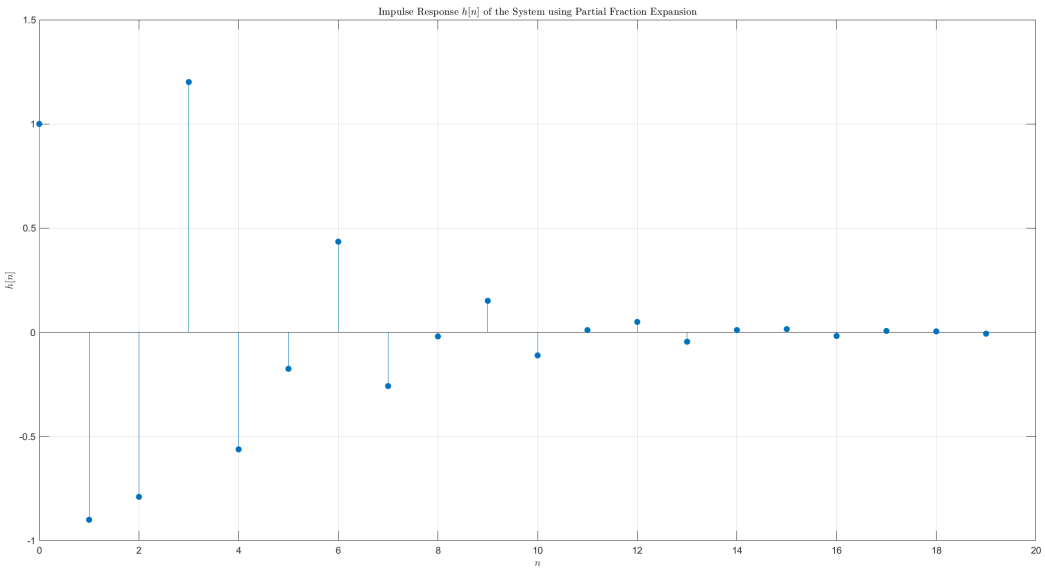

Figure 7: Impulse Response

```
disp('Poles:');
disp(poles);

% Generate impulse response from partial fractions if necessary
N = 20; % Define the length of the impulse response
impulse_response = impz(numerator, denominator, N);

% Plot the impulse response
n = 0:N-1;
stem(n, impulse_response, 'filled');
xlabel('n', 'Interpreter', 'latex');
ylabel('h[n]', 'Interpreter', 'latex');
title('Impulse Response h[n] of the System using Partial Fraction Expansion', '
    Interpreter', 'latex');
grid on;
```

Question 3

The following signal is given as a linear chirp:

$$x(t) = \cos(\pi\mu t^2 + 2\pi f_1 t + \phi)$$

(a) Find the instantaneous frequency of the signal. (The instantaneous frequency is equal to the derivative of the argument of the cosine function with respect to time.)

For the signal

$$x(t) = \cos(\pi\mu t^2 + 2\pi f_1 t + \phi)$$

the instantaneous frequency is obtained by taking the derivative of the argument of the cosine function with respect to time:

$$f(t) = \frac{1}{2\pi}\frac{d}{dt}\left(\pi\mu t^2 + 2\pi f_1 t + \phi\right)$$

By differentiating, we get:

$$f(t) = \frac{1}{2\pi}\left(2\pi\mu t + 2\pi f_1\right)$$

which simplifies to:

$$f(t) = \mu t + f_1$$

(b) Assume that $f_1 = 4$ kHz, $\mu = 600$ kHz/s, and ϕ is arbitrary. If the total duration of the signal is 50 milliseconds, determine the frequency sweep range of the signal within that time.

The instantaneous frequency $f(t)$ of a linear chirp signal is given by:

$$f(t) = f_1 + \mu t$$

1. **Starting Frequency:** At $t = 0$:

$$f(0) = f_1 = 4\,\text{kHz}$$

2. **Ending Frequency:** At $t = T = 0.05\,\text{s}$:

$$f(T) = f_1 + \mu T$$

Substituting the values:

$$f(T) = 4\,\text{kHz} + (600\,\text{kHz/s}) \cdot (0.05\,\text{s})$$

$$f(T) = 4\,\text{kHz} + 30\,\text{kHz} = 34\,\text{kHz}$$

Thus, the frequency range (sweep) of the signal over the interval $t = [0, 0.05]$ seconds is from **4 kHz to 34 kHz**.

(c) By sampling the signal with a sampling frequency of $f_s = 8$ kHz, plot the sampled signal.

```
% Parameters
f1 = 4e3;          % Initial frequency in Hz (4 kHz)
mu = 600e3;        % Chirp rate in Hz/s (600 kHz/s)
T = 0.05;          % Total time duration in seconds (50 ms)

% Parameters for the signal
phi = 0;           % Assume phase phi is 0 for simplicity
fs = 8e3;          % Sampling frequency in Hz (8 kHz)
t_sampled = 0:1/fs:T; % Sampled time vector

% Sampled signal x(t) using sampled time vector
x_sampled = cos(pi * mu * t_sampled.^2 + 2 * pi * f1 * t_sampled +
phi);

% Plot continuous chirp signal
figure;
```

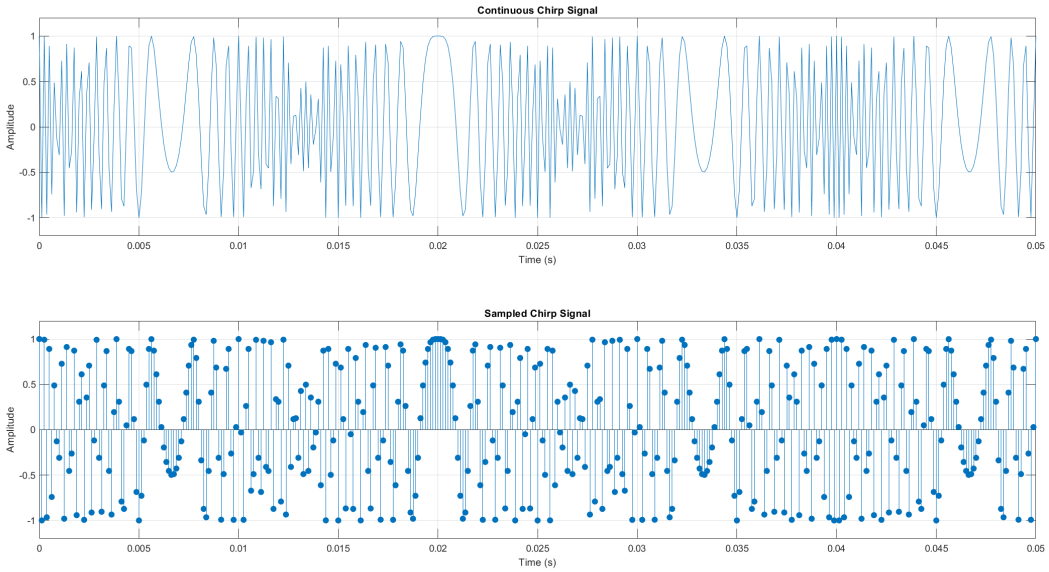

Figure 8: Discrete Signal of Chirp

```
16
17      subplot(2, 1, 1);
18      plot(t_sampled, x_sampled); % Continuous chirp
19      ylim([-1.2 1.2])
20      xlabel('Time (s)');
21      ylabel('Amplitude');
22      title('Continuous Chirp Signal');
23      grid on;
24
25      % Plot sampled chirp signal
26      subplot(2, 1, 2);
27      stem(t_sampled, x_sampled, 'filled'); % Sampled chirp
28      ylim([-1.2 1.2])
29      xlabel('Time (s)');
30      ylabel('Amplitude');
31      title('Sampled Chirp Signal');
32      grid on;
33
```

(d) Considering the changes in frequency over time, explain why in certain intervals of time the signal's frequency momentarily appears to be zero. Use the plots to find these intervals.

> In a linear chirp signal, the frequency changes over time. When the instantaneous frequency $f(t) = f_1 + \mu t$ gets close to zero, the signal temporarily stops oscillating and looks flat in the time-domain plot. This happens at points where the oscillations slow down or disappear, such as around 7 ms, 20 ms, and 33 ms in Figure 8. These moments show when the frequency briefly reaches zero before rising again.

(e) Is there any relations between aliasing and the calculations of part d?

Since we use a sampling frequency greater than twice the maximum signal frequency (to avoid aliasing), we know that, at a fixed sampling frequency, the likelihood of aliasing increases as the signal frequency gets higher.

Question 4

(a) The signals $x_1[n] = \sin^2\left(\frac{\pi}{10}n\right)$ and $x_2[n] = \sin\left(\frac{\pi}{10}n\right)$ are given. Plot and calculate the Fourier transform of $x_1[n]$ and $x_2[n]$ over the interval $[-\pi, \pi]$. Compare and analyze the Fourier transforms of these signals with respect to their frequency characteristics (use `fftshift` and `fft` for calculating the Fourier transform and plotting them).

```
% Define the range of n
n = -50:50; % You can adjust the range as needed

% Define the signals x1[n] and x2[n]
x1 = (sin(pi * n / 10) .^ 2) ./ ((pi * n / 10) .^ 2);
x2 = sin(pi * n / 10) ./ (pi * n / 10);

% Handle n = 0 separately to avoid division by zero
x1(n == 0) = 1; % Using limit as n -> 0 for x1[n]
x2(n == 0) = 1; % Using limit as n -> 0 for x2[n]

% Calculate Fourier Transforms using FFT
N = 1024; % Number of points for FFT
X1_fft = fftshift(fft(x1, N));
X2_fft = fftshift(fft(x2, N));

% Frequency range in [-pi, pi]
omega = linspace(-pi, pi, N);

% Plot the signals x1[n] and x2[n]
figure;
subplot(2, 1, 1);
stem(n, x1, 'filled');
xlabel('n');
ylabel('x_1[n]');
title('Signal x_1[n]');

subplot(2, 1, 2);
plot(omega, abs(X1_fft));
xlabel('\omega');
ylabel('|X_1(e^{j\omega})|');
title('Magnitude of Fourier Transform of x_1[n]');

% Plot the magnitude of Fourier Transform of x1[n] and x2[n]
figure;

subplot(2, 1, 1);
stem(n, x2, 'filled');
xlabel('n');
ylabel('x_2[n]');
title('Signal x_2[n]');

subplot(2, 1, 2);
plot(omega, abs(X2_fft), LineWidth=1.5);
xlabel('\omega');
ylabel('|X_2(e^{j\omega})|');
```

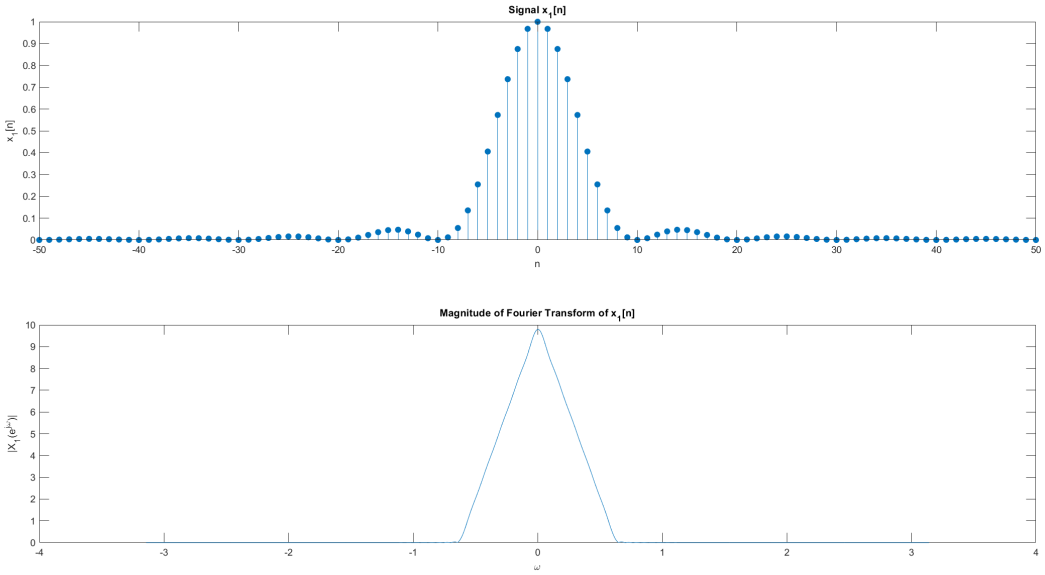

Figure 9: Signal $x_1[n]$

```
47      title('Magnitude of Fourier Transform of x_2[n]');
48
```

- **Narrower Spectrum for $x_1[n]$:** The Fourier Transform of $x_1[n]$ shows a concentrated peak around zero frequency, suggesting a low-pass characteristic with fewer high-frequency components.
- **Wider Spectrum for $x_2[n]$:** The Fourier Transform of $x_2[n]$ has a broader peak and more frequency components, indicating that it passes a wider range of frequencies.

This difference is consistent with the nature of the signals: $x_1[n]$ (sinc-squared function) has more concentrated energy in the lower frequencies, while $x_2[n]$ (sinc function) has a more spread-out frequency content.

(b) Generate the following signals based on $x_2[n]$, calculate and plot their Fourier transforms over the interval $[-\pi, \pi]$. Plot these transforms and compare them with part (a).

$$y_1[n] = x_2[2n]$$

$$y_2[n] = \begin{cases} x_2\left[\frac{n}{2}\right] & \text{if } n \text{ is even} \\ 0 & \text{if } n \text{ is odd} \end{cases}$$

$$y_3[n] = x_2[n] \times \sin(2\pi \times 0.3 \times n)$$

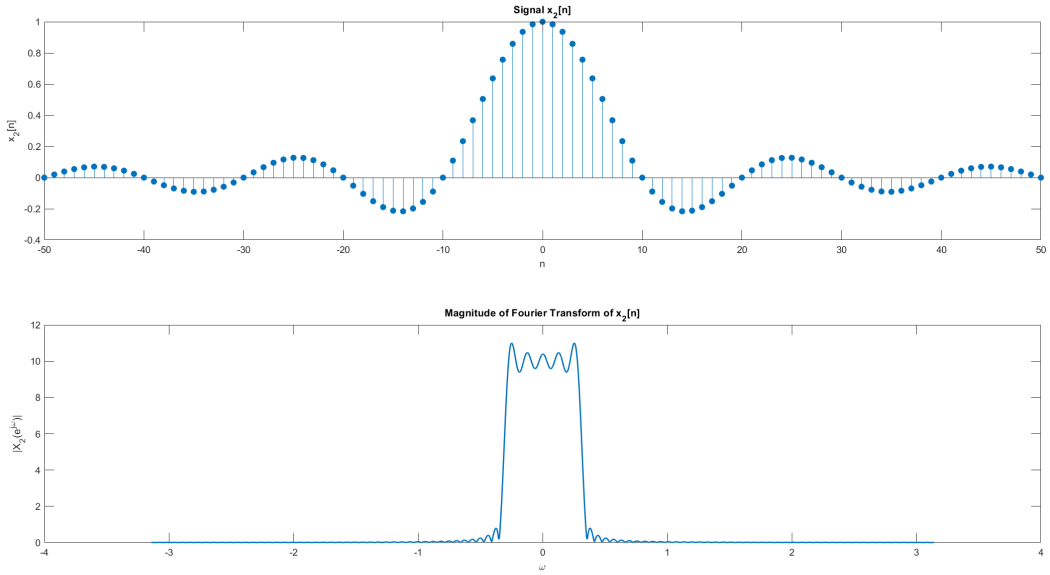

Figure 10: Signal $x_2[n]$

For this part, we expect the following Fourier transforms for $y_1[n]$, $y_2[n]$, and $y_3[n]$:
1. **Fourier Transform of $y_1[n]$:**

$$y_1[n] = x_2[2n]$$
$$Y_1(e^{j\omega}) = \frac{1}{2}\sum_{k=0}^{1} X_2\left(e^{j\frac{\omega}{2}+j\frac{2\pi k}{2}}\right)$$

This expression shows that $Y_1(e^{j\omega})$ is obtained by summing two shifted versions of $X_2(e^{j\omega})$, spaced π apart.
2. **Fourier Transform of $y_2[n]$:**

$$y_2[n] = x_2\left[\frac{n}{2}\right] \text{ for even } n, \text{ and 0 for odd } n$$
$$Y_2(e^{j\omega}) = X_2(e^{j2\omega})$$

This scaling of ω by 2 compresses the frequency spectrum of X_2 into Y_2, effectively reducing the range of ω.
3. **Fourier Transform of $y_3[n]$:**

$$y_3[n] = x_2[n] \cdot \sin(2\pi \times 0.3 \cdot n)$$
$$Y_3(e^{j\omega}) = \frac{1}{2j}\left(X_2\left(e^{j(\omega+2\pi\times 0.3)}\right) - X_2\left(e^{j(\omega-2\pi\times 0.3)}\right)\right)$$

This expression shows that $Y_3(e^{j\omega})$ is the difference of two shifted versions of $X_2(e^{j\omega})$, representing the effect of modulation by $\sin(2\pi \times 0.3 \cdot n)$.

```
    x2(n == 0) = 1; % Handle n = 0 separately to avoid division by zero

    % Define y1[n] = x2[2n]
```

```
% To calculate y1 correctly, we need to redefine the range for n to account for the downsampling
n_y1 = -25:25; % Adjusted range for y1, since we are sampling every 2nd n in x2
y1 = sin(pi * (2 * n_y1) / 10) ./ (pi * (2 * n_y1) / 10);
y1(n_y1 == 0) = 1; % Handle n_y1 = 0 separately to avoid division by zero

% Define y2[n] based on even/odd conditions
y2 = zeros(size(n));
for i = 1:length(n)
    if mod(n(i), 2) == 0
        y2(i) = x2(n(i) / 2 + 51); % Index adjustment for zero-centered n
    end
end

% Define y3[n] = x2[n] * sin(2 * 0.3 * n)
y3 = x2 .* sin(2 * pi * 0.3 * n);

% Plot the signals x2[n], y1[n], y2[n], and y3[n]
figure;
subplot(4, 1, 1);
stem(n, x2, 'filled');
xlabel('n');
ylabel('x_2[n]');
title('Signal x_2[n]');

subplot(4, 1, 2);
stem(n_y1, y1, 'filled');
xlabel('n');
ylabel('y_1[n]');
title('Signal y_1[n] = x_2[2n]');

subplot(4, 1, 3);
stem(n, y2, 'filled');
xlabel('n');
ylabel('y_2[n]');
title('Signal y_2[n]');

subplot(4, 1, 4);
stem(n, y3, 'filled');
xlabel('n');
ylabel('y_3[n]');
title('Signal y_3[n] = x_2[n] * sin(2\pi \cdot 0.3 \cdot n)');

% Fourier Transform calculations
N = 1024; % Number of points for FFT
omega = linspace(-pi, pi, N);

% Fourier Transforms of the signals
X2_fft = fftshift(fft(x2, N));
Y1_fft = fftshift(fft(y1, N));
Y2_fft = fftshift(fft(y2, N));
Y3_fft = fftshift(fft(y3, N));

% Plot Fourier Transforms
figure;
```

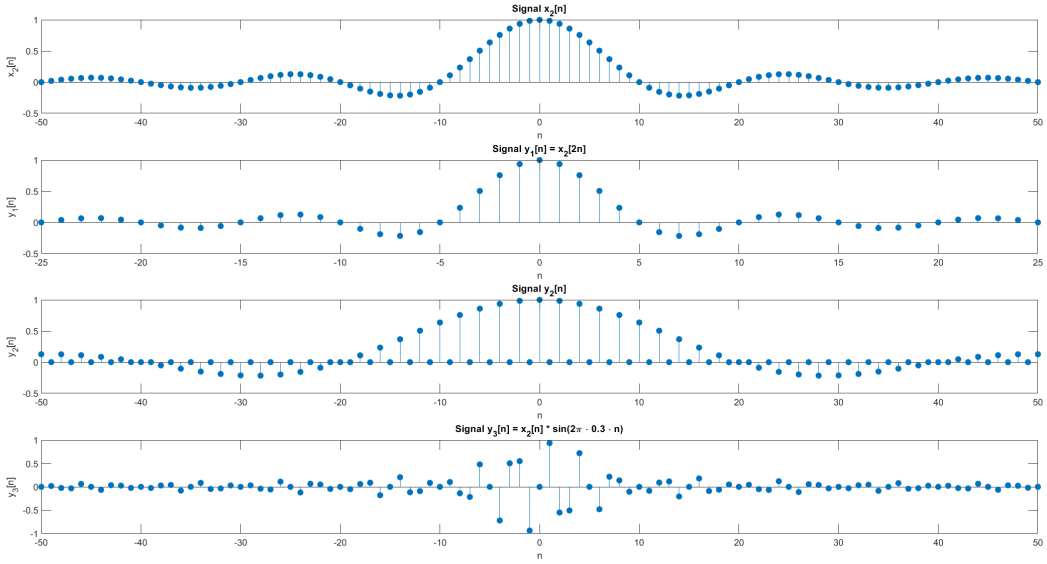

Figure 11: Time Domain

```
58    subplot(4, 1, 1);
59    plot(omega, abs(X2_fft), LineWidth=1.5);
60    xlabel('\omega');
61    ylabel('|X_2(e^{j\omega})|');
62    title('Magnitude of Fourier Transform of x_2[n]');
63
64    subplot(4, 1, 2);
65    plot(omega, abs(Y1_fft), LineWidth=1.5);
66    xlabel('\omega');
67    ylabel('|Y_1(e^{j\omega})|');
68    title('Magnitude of Fourier Transform of y_1[n]');
69
70    subplot(4, 1, 3);
71    plot(omega, abs(Y2_fft), LineWidth=1.5);
72    xlabel('\omega');
73    ylabel('|Y_2(e^{j\omega})|');
74    title('Magnitude of Fourier Transform of y_2[n]');
75
76    subplot(4, 1, 4);
77    plot(omega, abs(Y3_fft), LineWidth=1.5);
78    xlabel('\omega');
79    ylabel('|Y_3(e^{j\omega})|');
80    title('Magnitude of Fourier Transform of y_3[n]');
81
```

Question 5

You are not allowed to use built-in MATLAB functions to solve this question.

(a) Write a function named `sinc_interpolation` that receives a sampled signal and its sampling period, as well as the desired time period of the output signal. It outputs the interpolated signal. Use the sinc function for time-domain interpolation. For this purpose, use a finite length of the sinc function as an approximation to the original sampled signal.

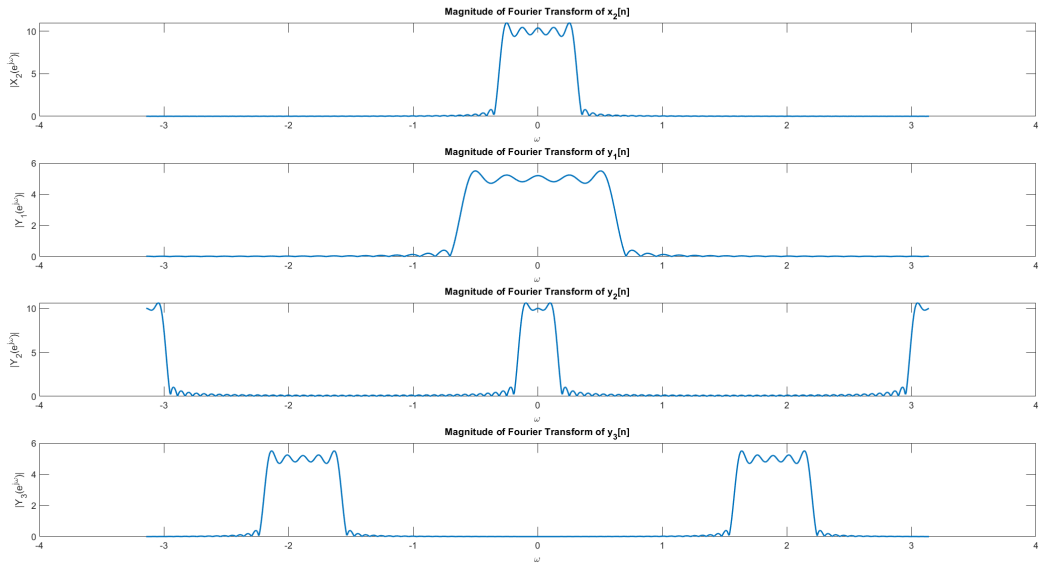

Figure 12: Frequency Domain

```
function y_interp = sinc_interpolation(x_samples, t_samples, t_interp, Ts)
    % Initialize the interpolated signal vector
    y_interp = zeros(size(t_interp));

    % Perform sinc interpolation for each point in t_interp
    for k = 1:length(t_interp)
        % Calculate the sinc kernel for each sample point
        sinc_kernel = sinc((t_interp(k) - t_samples) / Ts);

        % Calculate the interpolated value at time t_interp(k)
        y_interp(k) = sum(x_samples .* sinc_kernel);
    end
end

```

(b) Apply your function to interpolate the following signal:

```
% Define parameters
Fs = 5000;              % Sampling frequency (5 kHz)
Ts = 1 / Fs;            % Sampling period (0.0002 seconds)

% Define continuous and sampled time vectors
t_continuous = 0:1e-5:0.01;  % Continuous time vector with very fine resolution
t_samples = 0:Ts:0.01;       % Sampled time vector based on Fs

% Generate the continuous signal
x_continuous = sin(1000 * pi * t_continuous) + sin(2000 * pi * t_continuous);

% Generate the sampled signal
x_samples = sin(1000 * pi * t_samples) + sin(2000 * pi * t_samples);

```

```
% Set the interpolation time vector to match the continuous time vector
t_interp = t_continuous;

% Perform sinc interpolation using the provided sinc_interpolation function
x_interp1 = sinc_interpolation(x_samples, t_samples, t_interp, Ts);
limited_x_interp1 = limited_sinc_interpolation(x_samples, t_samples, t_interp, Ts, 9);

% Plotting
figure;

% Plot original continuous signal
subplot(4, 1, 1);
plot(t_continuous, x_continuous, 'LineWidth', 1.5);
xlabel('Time (s)', 'Interpreter', 'latex');
ylabel('Amplitude', 'Interpreter', 'latex');
title('Original Continuous Signal $x(t)$', 'Interpreter', 'latex');
grid on;

% Plot sampled signal
subplot(4, 1, 2);
stem(t_samples, x_samples, 'filled');
xlabel('Time (s)', 'Interpreter', 'latex');
ylabel('Amplitude', 'Interpreter', 'latex');
title('Sampled Points $x[n]$', 'Interpreter', 'latex');
grid on;

% Plot interpolated signal
subplot(4, 1, 3);
plot(t_interp, x_interp1, 'LineWidth', 1.2);
xlabel('Time (s)', 'Interpreter', 'latex');
ylabel('Amplitude', 'Interpreter', 'latex');
title('Interpolated Signal using Sinc Interpolation $\hat{x}(t)$', 'Interpreter', 'latex');
grid on;

% Plot interpolated signal
subplot(4, 1, 4);
plot(t_interp, limited_x_interp1, 'LineWidth', 1.2);
xlabel('Time (s)', 'Interpreter', 'latex');
ylabel('Amplitude', 'Interpreter', 'latex');
title('Interpolated Signal using Limited Sinc Interpolation $\hat{x}(t)$', 'Interpreter', 'latex');
grid on;

```

(c) Write another function named `limited_sinc_interpolation` similar to the function from part 1, but with a limited length of the sinc function. The sinc length should be smaller than in part 1 (only the 9 first lobes of the sinc function is required).

```
function x_interp = limited_sinc_interpolation(x_samples, t_samples, t_interp, Ts, L)
    % Initialize the interpolated signal vector
    x_interp = zeros(size(t_interp));

    % Perform limited sinc interpolation for each point in t_interp
```

```
    for i = 1:length(t_interp)
        % Find the nearest sample index in t_samples to the current t_interp(i)
        [~, nearest_idx] = min(abs(t_samples - t_interp(i)));

        % Define the range of indices for the limited sinc kernel
        start_idx = max(nearest_idx - L, 1);
        end_idx = min(nearest_idx + L, length(t_samples));

        % Calculate the sinc kernel for the limited range
        sinc_kernel = sinc((t_interp(i) - t_samples(start_idx:end_idx)) / Ts);

        % Calculate the interpolated value at time t_interp(i) using limited kernel
        x_interp(i) = sum(x_samples(start_idx:end_idx) .* sinc_kernel);
    end
end

```

(d) Use this function instead of `sinc_interpolation` in part 2 and plot and compare the results.

```
% Define parameters
Fs = 5000;              % Sampling frequency (5 kHz)
Ts = 1 / Fs;            % Sampling period (0.0002 seconds)

% Define continuous and sampled time vectors
t_continuous = 0:1e-5:0.01;  % Continuous time vector with very fine resolution
t_samples = 0:Ts:0.01;       % Sampled time vector based on Fs

% Generate the continuous signal
x_continuous = sin(1000 * pi * t_continuous) + sin(2000 * pi * t_continuous);

% Generate the sampled signal
x_samples = sin(1000 * pi * t_samples) + sin(2000 * pi * t_samples);

% Set the interpolation time vector to match the continuous time vector
t_interp = t_continuous;

% Perform sinc interpolation using the provided sinc_interpolation function
x_interp1 = sinc_interpolation(x_samples, t_samples, t_interp, Ts);
limited_x_interp1 = limited_sinc_interpolation(x_samples, t_samples, t_interp, Ts, 9);

% Plotting
figure;

% Plot original continuous signal
subplot(4, 1, 1);
plot(t_continuous, x_continuous, 'LineWidth', 1.5);
xlabel('Time (s)', 'Interpreter', 'latex');
ylabel('Amplitude', 'Interpreter', 'latex');
```

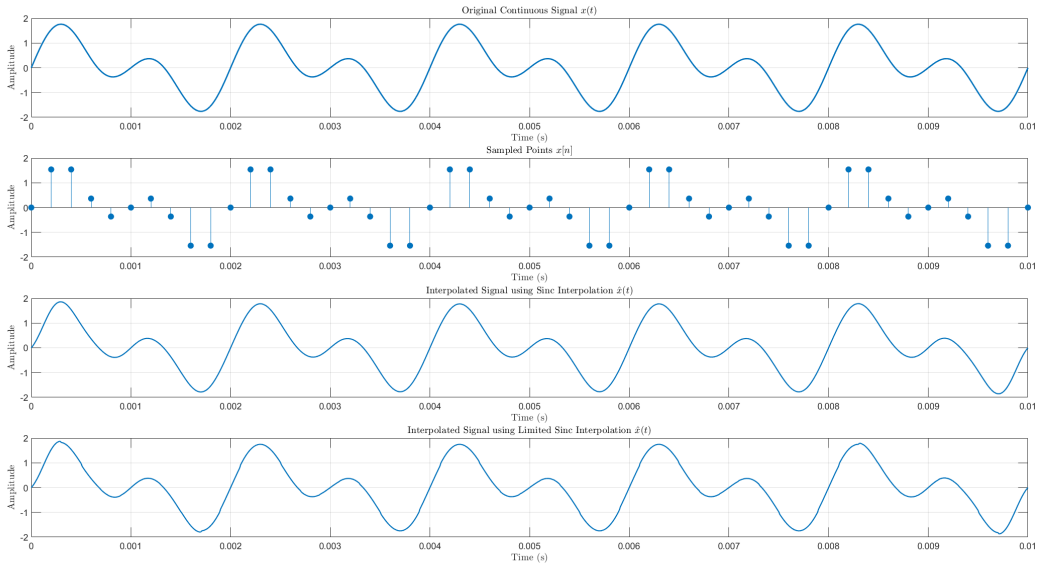

Figure 13: signal $x(t)$, $x[n]$, *interploated*, and *limited*$_i$*nterpolated*

```
30      title('Original Continuous Signal $x(t)$', 'Interpreter', 'latex');
31      grid on;
32
33      % Plot sampled signal
34      subplot(4, 1, 2);
35      stem(t_samples, x_samples, 'filled');
36      xlabel('Time (s)', 'Interpreter', 'latex');
37      ylabel('Amplitude', 'Interpreter', 'latex');
38      title('Sampled Points $x[n]$', 'Interpreter', 'latex');
39      grid on;
40
41      % Plot interpolated signal
42      subplot(4, 1, 3);
43      plot(t_interp, x_interp1, 'LineWidth', 1.2);
44      xlabel('Time (s)', 'Interpreter', 'latex');
45      ylabel('Amplitude', 'Interpreter', 'latex');
46      title('Interpolated Signal using Sinc Interpolation $\hat{x}(t)$', 'Interpreter
        ', 'latex');
47      grid on;
48
49      % Plot interpolated signal
50      subplot(4, 1, 4);
51      plot(t_interp, limited_x_interp1, 'LineWidth', 1.2);
52      xlabel('Time (s)', 'Interpreter', 'latex');
53      ylabel('Amplitude', 'Interpreter', 'latex');
54      title('Interpolated Signal using Limited Sinc Interpolation $\hat{x}(t)$', '
        Interpreter', 'latex');
55      grid on;
56
```

We can conclude that if we interpolate using fewer sinc functions, the signal remains well-reconstructed, and the omitted portion of the signal is minimal.

(e) change the sampling frequency of input and output and plot both signals using both the functions and compare the results. how well does the interpolation work in each case?

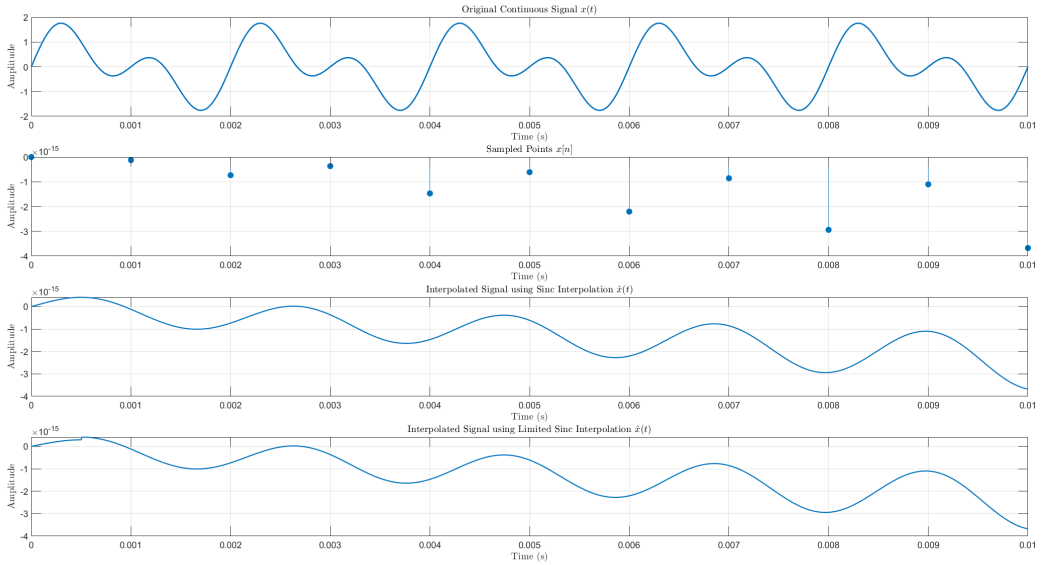

Figure 14: $F_S = 1000$

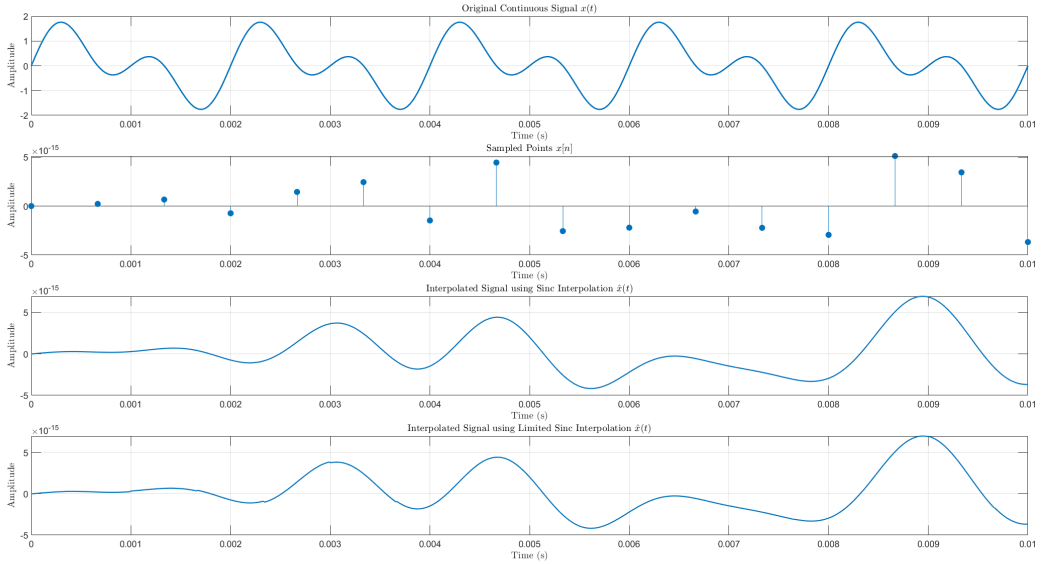

Figure 15: $F_S = 1500$

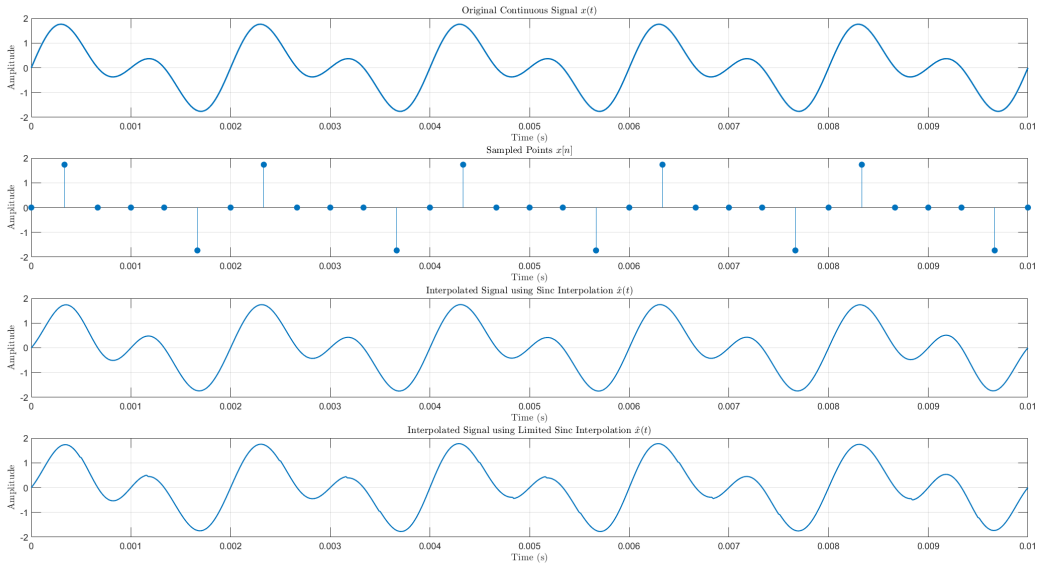

Figure 16: $F_S = 3000$

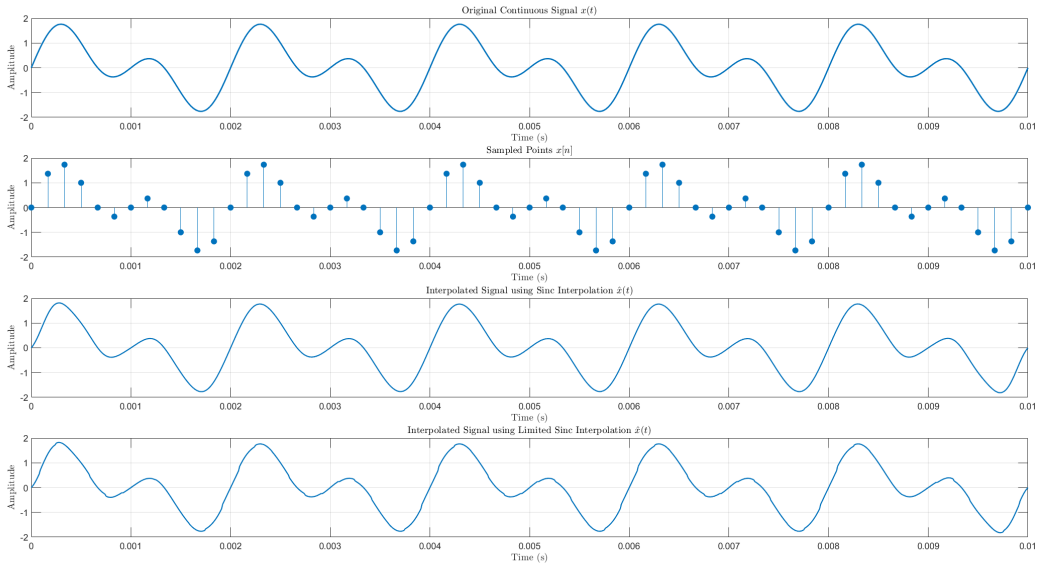

Figure 17: $F_S = 6000$

We can see that Figures 14 and 15 are not well reconstructed due to the aliasing effect, but the other two are well reconstructed.