DSP Lab 5 By Syed Asghar Abbas Zaidi

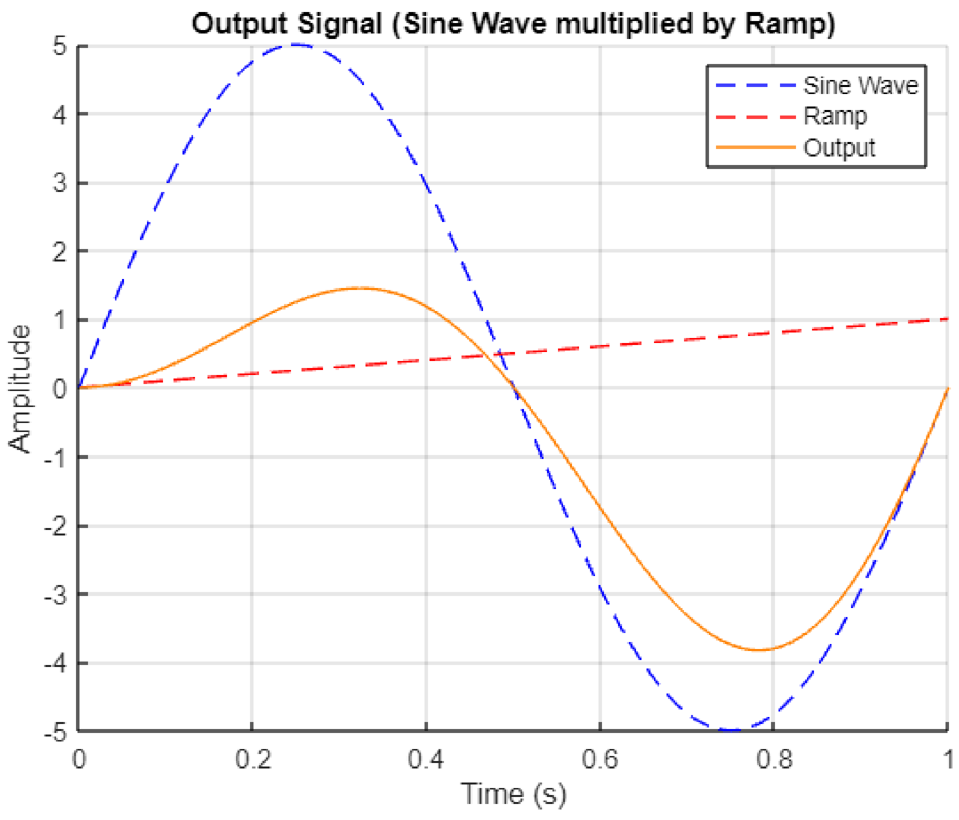
Task 1

a) Generate a sine wave signal having amplitude 5 with a frequency of 1 Hz and a duration of 1 second using MATLAB.

b) Multiply the sine wave signal by a ramp function to obtain a output like the figure below

In this task, we start by generating a sine wave signal with an amplitude of 5, a frequency of 1 Hz, and a duration of 1 second using MATLAB. The sine wave represents a periodic oscillation commonly found in various real-world phenomena. Next, we multiply this sine wave by a ramp function. The ramp function linearly increases from 0 to 1 over the same duration as the sine wave.

```
% Define parameters
amplitude = 5;      % Amplitude of the sine wave
frequency = 1;      % Frequency of the sine wave (in Hz)
DUR = 1;       % Duration of the sine wave (in seconds)
sampling_rate = 100; % Sampling rate (number of samples per second)

% Calculate time vector
t = linspace(0, DUR, DUR * sampling_rate);

hold on;
% Generate sine wave signal
sine_wave = amplitude * sin(2 * pi * frequency * t);
%(a)
plot(t, sine_wave , 'b', 'DisplayName', 'Sine Wave', 'LineStyle', '--');

% Generate ramp function
ramp = linspace(0, 1, length(t));
plot(t, ramp, 'r', 'DisplayName', 'Ramp', 'LineStyle', '--');

% Multiply sine wave signal by ramp function
output = sine_wave .* ramp;
%(b)
plot(t, output, 'Color', [1, 0.5, 0], 'DisplayName', 'Output');

% Plot the output
hold off;
xlabel('Time (s)');
ylabel('Amplitude');
title('Output Signal (Sine Wave multiplied by Ramp)');
grid on;

% Add legends
legend('show');
```



Sine Wave Generation: We define the characteristics of a sine wave using its amplitude, frequency, and duration. The `linspace` function constructs a time vector encompassing equally spaced samples across the specified time interval. The sine wave is then generated mathematically using the formula `amplitude * sin(2 * pi * frequency * t)`.

Ramp Function: A ramp function, created with `linspace`, increases linearly from 0 to 1 over the signal's duration. This ramp modulates the sine wave's amplitude, causing it to vary over time.

Task 1.2

c) Condition the amplitude of the resulting signal from -2 to 2.

We normalize the output signal to have values between -2 and 2. Normalization ensures that the quantization process operates within a specific range, making it easier to represent the signal digitally.

d) Quantize the signal using a 3-bit analog-to-digital converter (ADC) with a sample and hold technique. You can use round function to generate the quantized signal

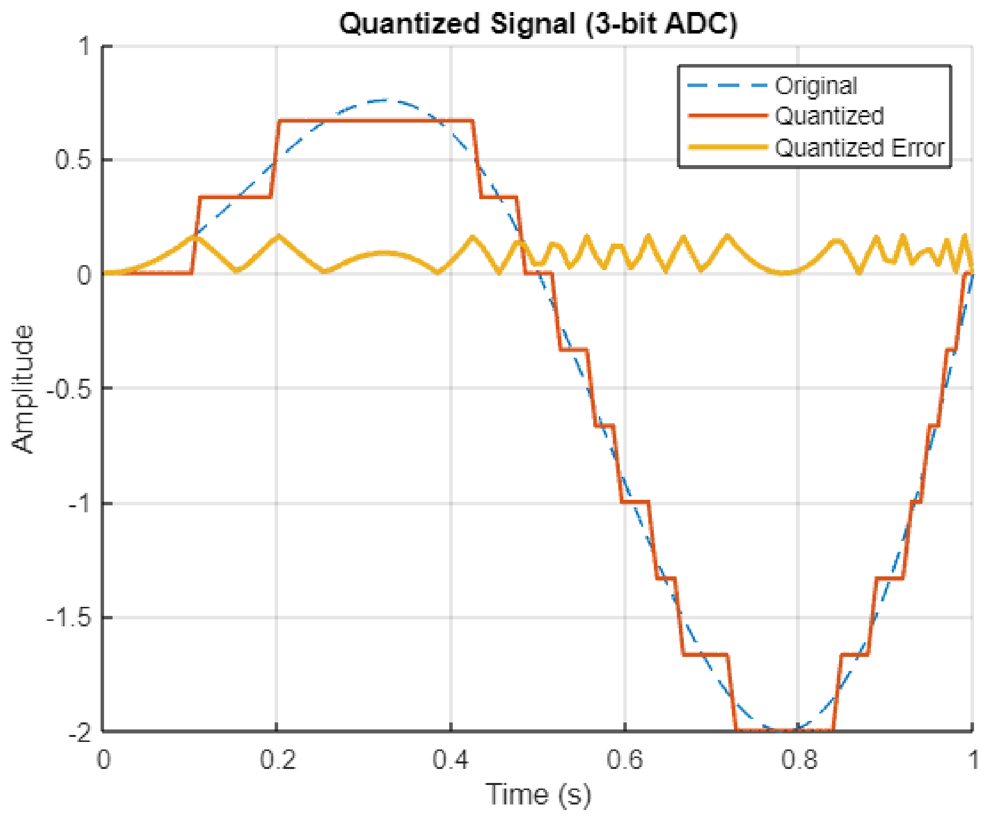
We define the number of bits used by the ADC (analog-to-digital converter). The number of quantization levels is calculated as 2 raised to the power of (number of bits - 1). The normalized signal (scaled to -1 to 1) undergoes quantization using the `round` function. This process maps the continuous signal values to discrete levels within the available quantization bins, effectively introducing an approximation

e) Find the Quantization error and plot the error for comparison

The absolute difference between the original and quantized signals represents the quantization error. This error quantifies the information loss incurred due to rounding the continuous signal values into discrete levels.

```
quantization_error = result - quantized_signal
```

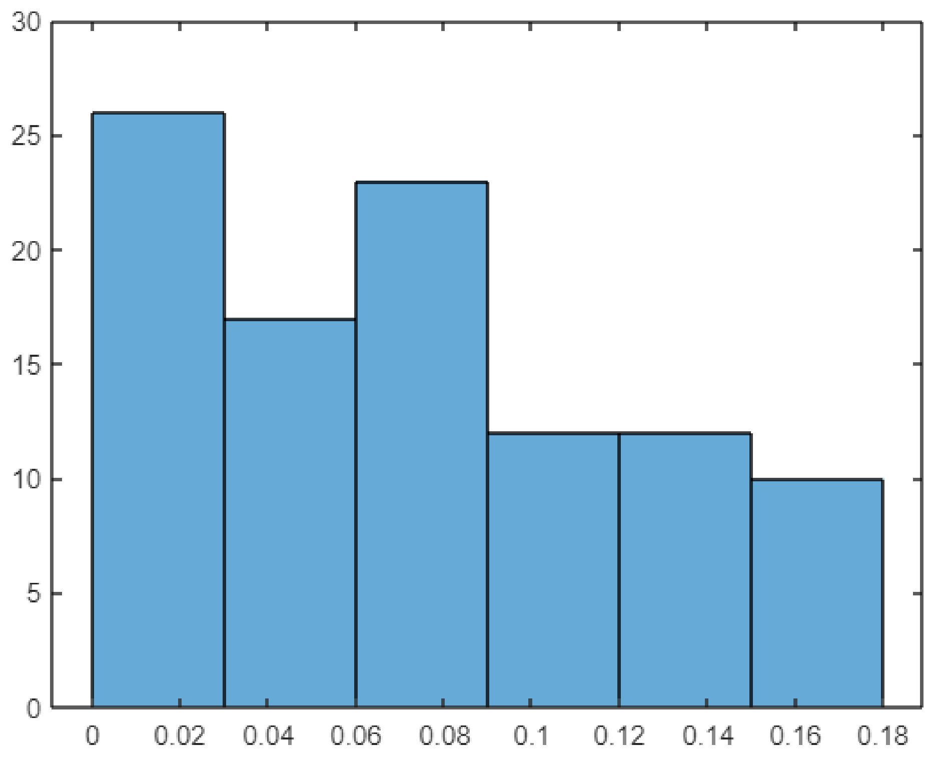
```
bit_depth = 3;      % Number of bits for ADC quantization

% Condition the amplitude from -2 to 2

Mini_1 = min(output);
Max_1 = max(output);
Range = abs(Max_1-Mini_1);

%c
output = output/max(abs(output));
output = output*2;


% Define parameters
num_levels = 2^(bit_depth-1); % Number of quantization levels

% Quantize the normalized signal using round function
normalized_signal = output;
%d
quantized_signal = round(normalized_signal * (num_levels - 1)) / (num_levels - 1);

% Calculate quantization error
% e
quantization_error = abs(output - quantized_signal);



% Plot the quantized signal
figure;
hold on;
plot(t, output, 'DisplayName', 'Original' , 'LineStyle', '--');
plot(t, quantized_signal, 'linewidth', 1.5, 'DisplayName', 'Quantized');
plot(t, quantization_error, 'LineWidth', 2 , 'DisplayName', 'Quantized Error');
hold off;
xlabel('Time (s)');
ylabel('Amplitude');
title('Quantized Signal (3-bit ADC)');
grid on;
legend('show');
```



f) Plot histogram for Quantization error

A histogram is a powerful tool for visualizing the distribution of the quantization error. By plotting the number of occurrences of each error value on the y-axis and the error value itself on the x-axis, we gain valuable insights into the behavior of the quantization process. By analyzing the histogram, we can assess the effectiveness of the chosen number of quantization bits. A wider spread of errors often indicates a need for more bits to achieve a desired level of precision.

```
%f
figure;
histogram(quantization_error);
```



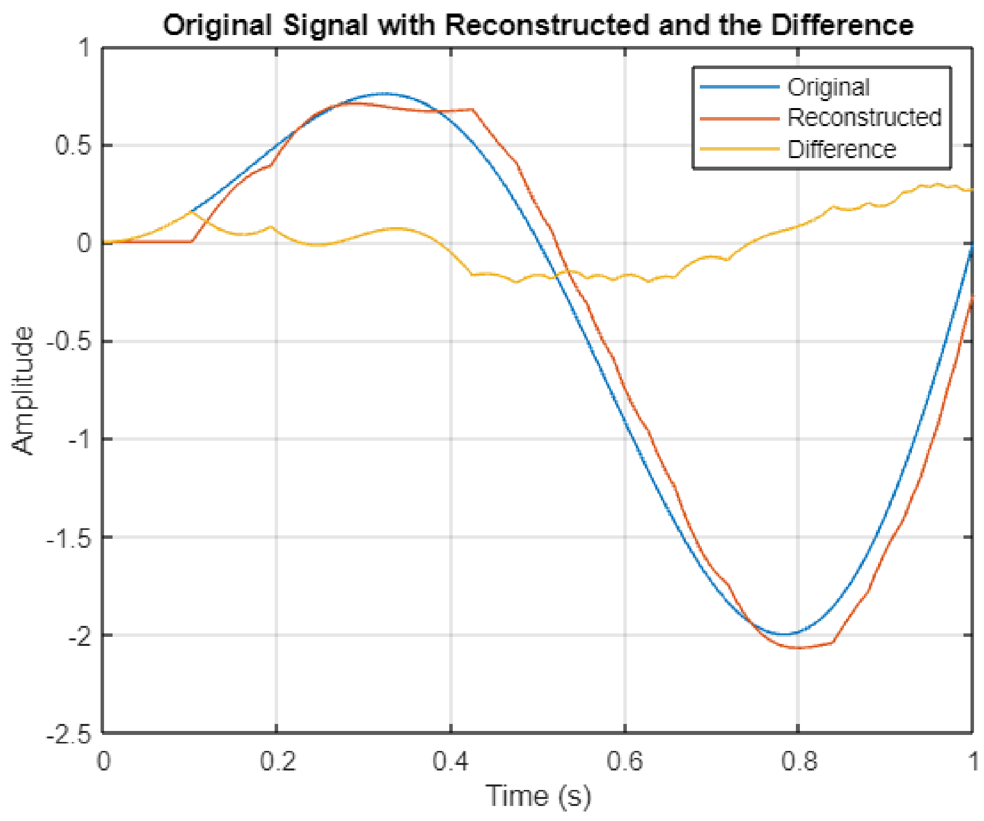
g) Reconstruct the original signal using an ideal low-pass filter (sinc filter) with a cutoff frequency of 5 Hz implemented on Quantized signal

An ideal low-pass filter, often implemented using the sinc function, is introduced. This filter allows frequencies below a defined cutoff frequency to pass through while attenuating higher frequencies. The filter is applied to the quantized signal using the `filter` function. By filtering out the high-frequency components introduced by quantization, the reconstructed signal aims to approximate the original sine wave more closely. We then plot the original, reconstructed, and difference signals to visually compare the effects of filtering and assess the reconstruction quality.

h) Plot the original signal, reconstructed signal, and the difference between these two signals on the same graph. (Reconstructed)

```
% Define cutoff frequency for the ideal low-pass filter
cutoff_freq = 5;

% Generate sinc function for the low-pass filter
sinc_func = sinc(2 * cutoff_freq * t);

% Normalize the sinc filter
LP = sinc_func / sum(sinc_func);

% Reconstruct the signal using the low-pass filter
reconstruction = filter(LP, 1, quantized_signal);
```

```
% Calculate the difference between the original and reconstructed signals
diff = output - reconstruction;

% Plot the original signal, reconstructed signal, and their difference
figure;
plot(t, output);
hold on;
plot(t, reconstruction);
plot(t, diff);
hold off;
legend('Original', 'Reconstructed', 'Difference');
xlabel('Time (s)');
ylabel('Amplitude');
title('Original Signal with Reconstructed and the Difference');
ylim([-2.5, 1]); % Limit y-axis for better visualization
grid on;
```



i)

Quantization effectively approximates a signal, resulting in a reconstructed waveform with minimal observable delay. However, the process of introducing a specific number of digital levels (2^num_of_bits) through quantization contrasts with the continuous nature of real-life signal values. As a consequence, encoding a signal using quantization can lead to some loss of information.

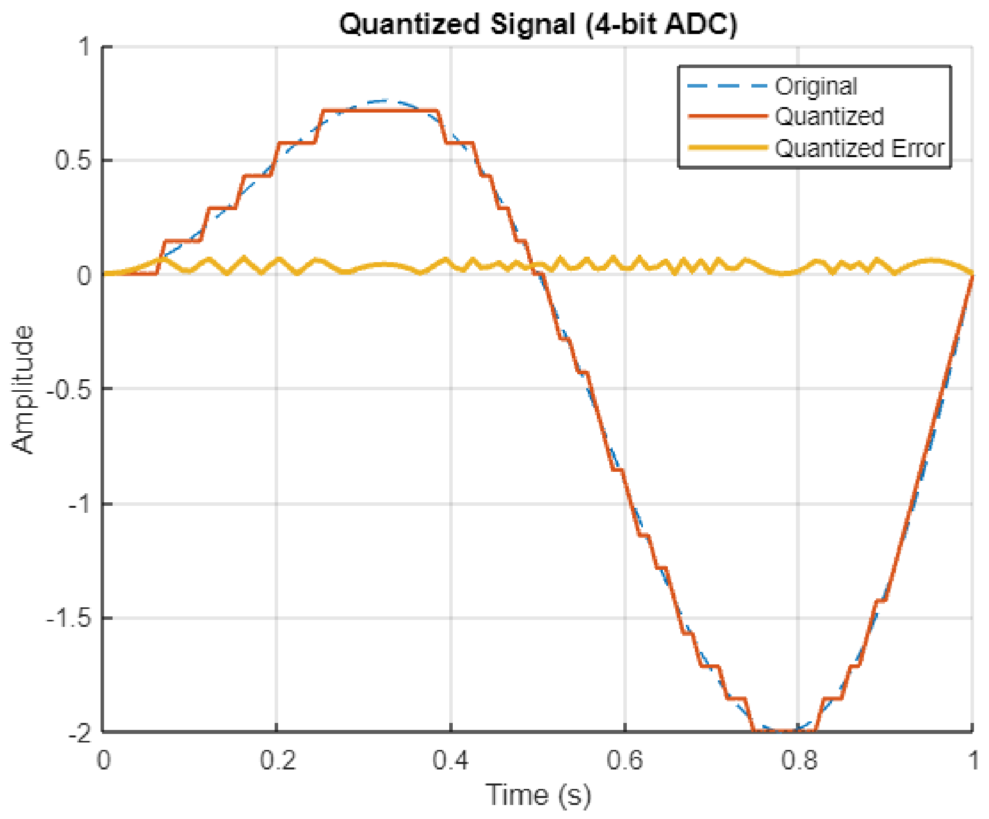
An ADC essentially discretizes the continuous signal amplitude by dividing the signal's range into a finite number of levels. The higher the number of bits used by the ADC, the more quantization levels are available. With more levels, the ADC can represent a wider range of signal values with greater precision. Consequently, the quantization error, which is the difference between the original signal value and its closest quantized level, is reduced. Conversely, using a lower number of bits translates to fewer quantization levels, resulting in a coarser representation of the original signal. This leads to a larger quantization error and a more significant loss of information.

Task 2

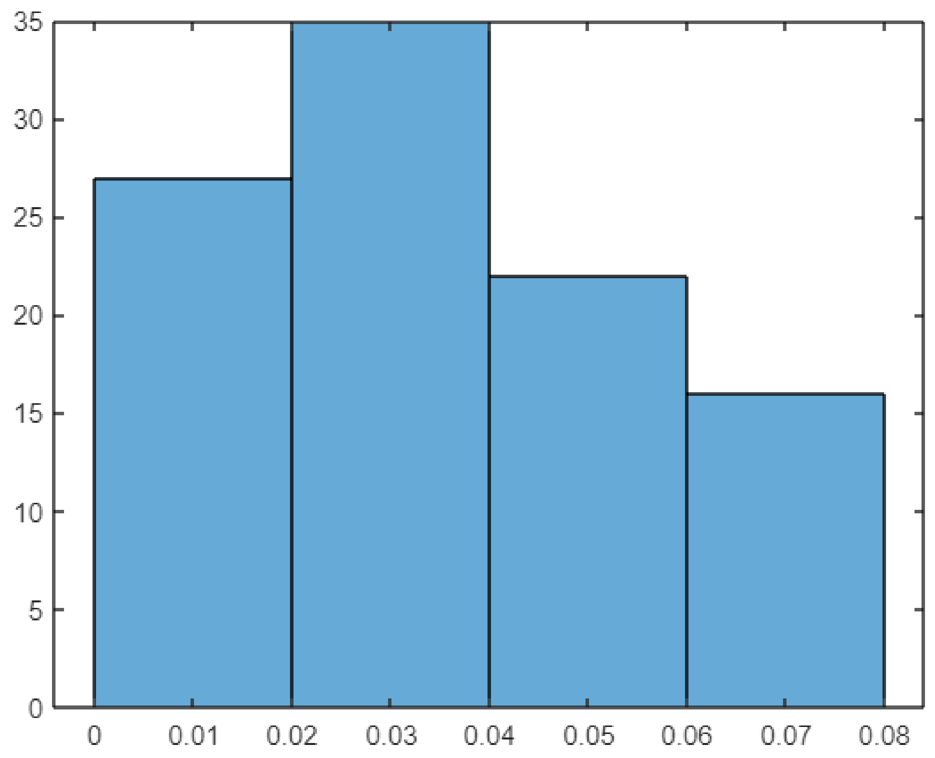
Implementing 4-bit ADC

```
 clear;
 % Define parameters
amplitude = 5;      % Amplitude of the sine wave
frequency = 1;      % Frequency of the sine wave (in Hz)
DUR = 1;       % Duration of the sine wave (in seconds)
sampling_rate = 100; % Sampling rate (number of samples per second)

% Calculate time vector
t = linspace(0, DUR, DUR * sampling_rate);
sine_wave = amplitude * sin(2 * pi * frequency * t);
ramp = linspace(0, 1, length(t));
output = sine_wave .* ramp;

% Add legends
legend('show');

bit_depth = 4;      % Number of bits for ADC quantization

% Condition the amplitude from -2 to 2

Mini_1 = min(output);
Max_1 = max(output);
Range = abs(Max_1-Mini_1);

%c
output = output/max(abs(output));
output = output*2;


% Define parameters
num_levels = 2^(bit_depth-1); % Number of quantization levels

% Quantize the normalized signal using round function
normalized_signal = output;
%d
quantized_signal = round(normalized_signal * (num_levels - 1)) / (num_levels - 1);
```

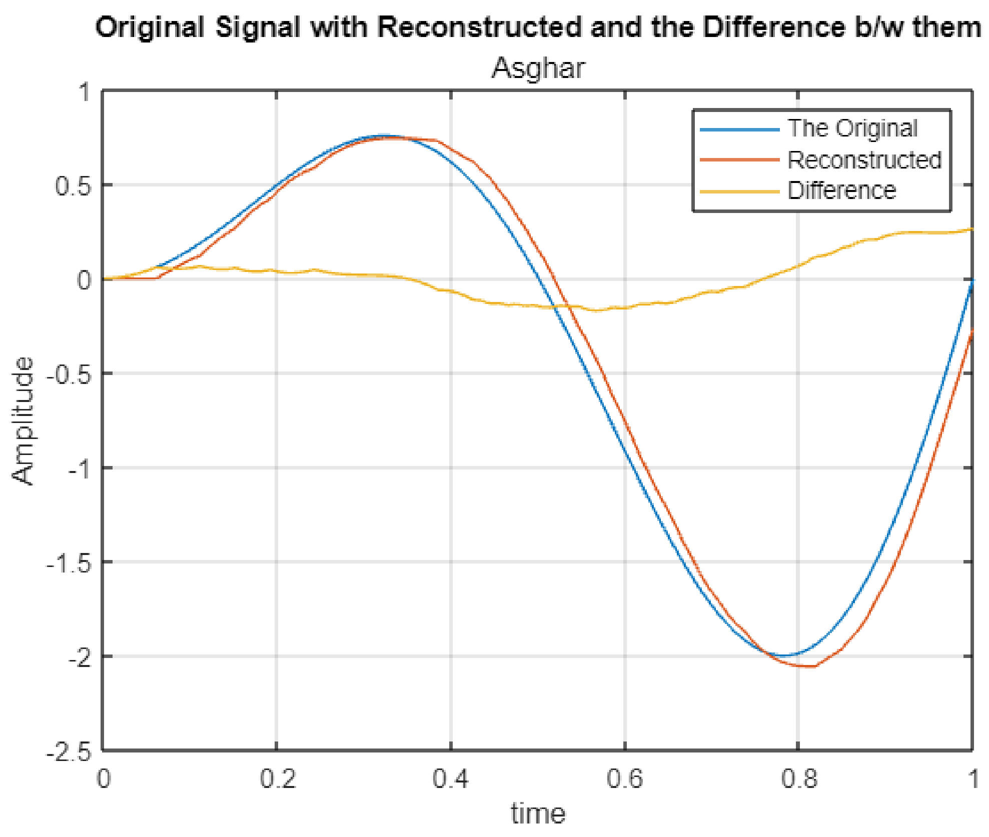
```
% Calculate quantization error
% e
quantization_error = abs(output - quantized_signal);



% Plot the quantized signal
figure;
hold on;
plot(t, output, 'DisplayName', 'Original' , 'LineStyle', '--');
plot(t, quantized_signal, 'linewidth', 1.5, 'DisplayName', 'Quantized');
plot(t, quantization_error, 'LineWidth', 2 , 'DisplayName', 'Quantized Error');
hold off;
xlabel('Time (s)');
ylabel('Amplitude');
title('Quantized Signal (4-bit ADC)');
grid on;
legend('show');
```



```

%f
figure;
histogram(quantization_error);
```

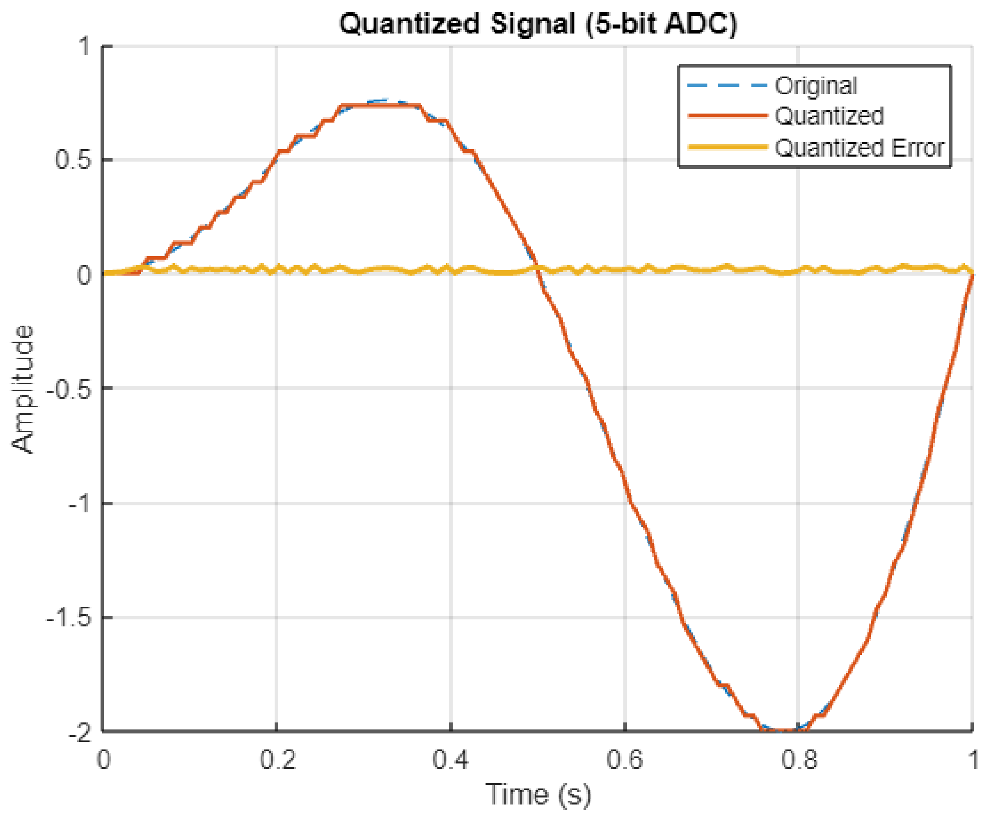


```
cuttoff_freq = 5;
 sinc_func = sinc(2*cuttoff_freq*t);
 LP = sinc_func/sum(sinc_func);
 reconstruction = filter(LP,1,quantized_signal);

 diff = output - reconstruction;
 figure;
 plot(t,output);
 hold on;
 plot(t,reconstruction);
 plot(t,diff);
 hold off;
 legend("The Original","Reconstructed","Difference");
 xlabel("time");ylabel("Amplitude");
 ylim([-2.5,1]);
 title("Original Signal with Reconstructed and the Difference b/w them", "Asghar")
 grid on;
```

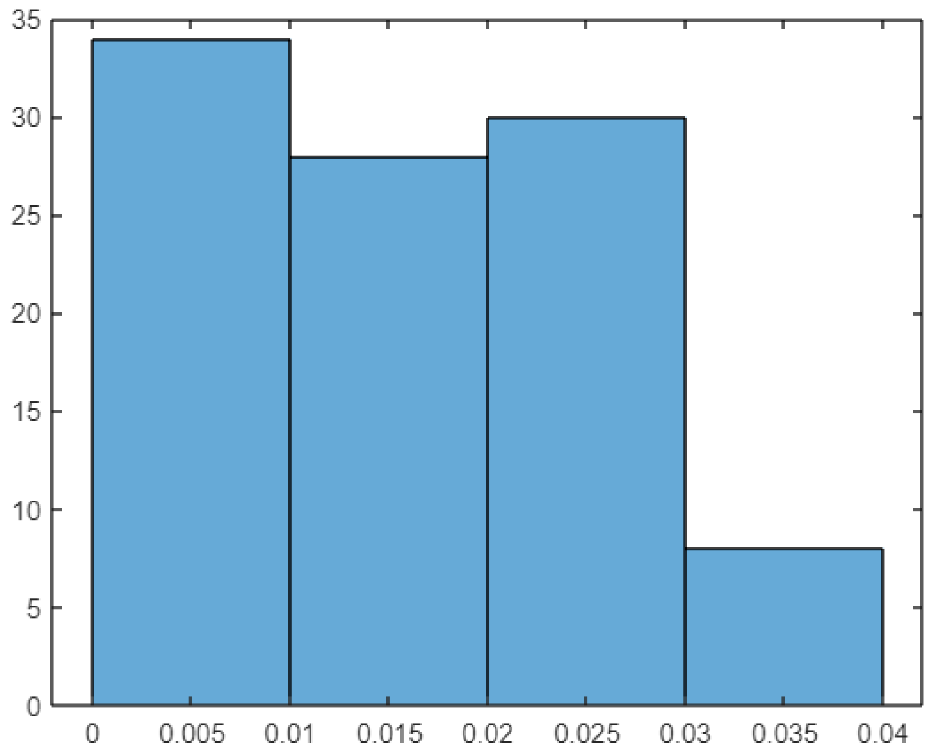


5-bit ADC

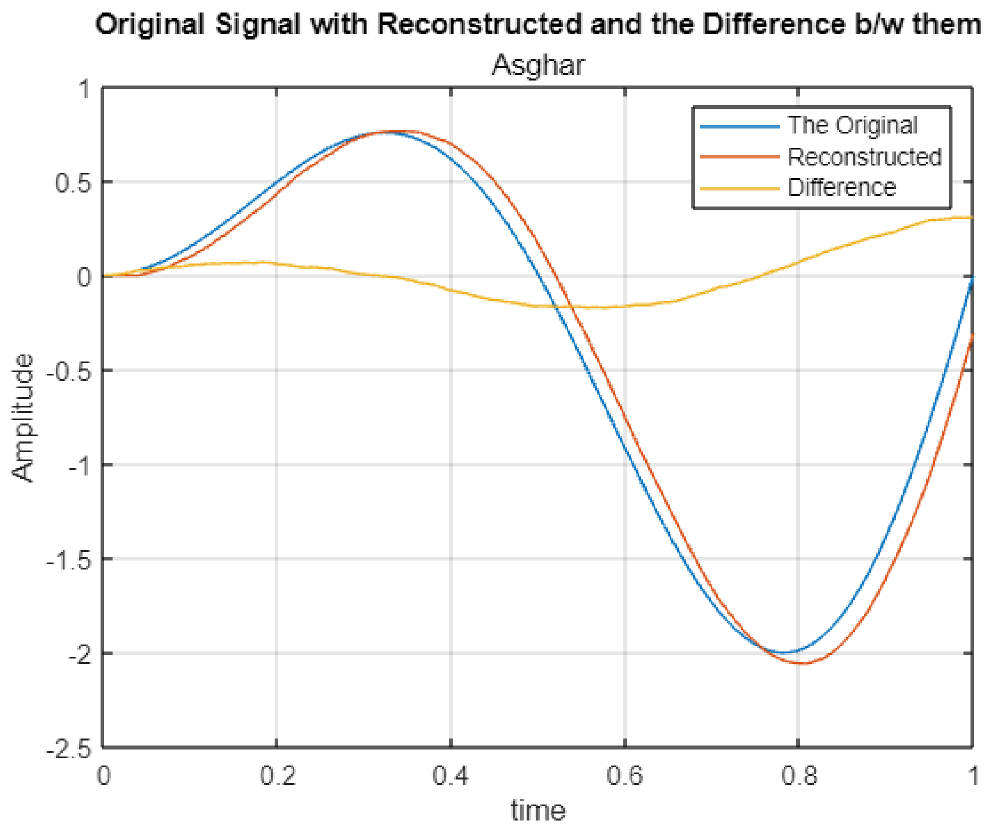
```
clear;
 % Define parameters
amplitude = 5;      % Amplitude of the sine wave
frequency = 1;      % Frequency of the sine wave (in Hz)
DUR = 1;       % Duration of the sine wave (in seconds)
sampling_rate = 100; % Sampling rate (number of samples per second)

% Calculate time vector
t = linspace(0, DUR, DUR * sampling_rate);
sine_wave = amplitude * sin(2 * pi * frequency * t);
ramp = linspace(0, 1, length(t));
output = sine_wave .* ramp;

% Add legends
legend('show');

bit_depth = 5;      % Number of bits for ADC quantization
Mini_1 = min(output);
Max_1 = max(output);
Range = abs(Max_1-Mini_1);

%c
output = output/max(abs(output));
output = output*2;


% Define parameters
num_levels = 2^(bit_depth-1); % Number of quantization levels

% Quantize the normalized signal using round function
normalized_signal = output;
%d
quantized_signal = round(normalized_signal * (num_levels - 1)) / (num_levels - 1);

% Calculate quantization error
% e
quantization_error = abs(output - quantized_signal);



% Plot the quantized signal
figure;
hold on;
plot(t, output, 'DisplayName', 'Original' , 'LineStyle', '--');
plot(t, quantized_signal, 'linewidth', 1.5, 'DisplayName', 'Quantized');
plot(t, quantization_error, 'LineWidth', 2 , 'DisplayName', 'Quantized Error');
hold off;
xlabel('Time (s)');
ylabel('Amplitude');
title('Quantized Signal (5-bit ADC)');
grid on;
legend('show');
```



```
%f
figure;
histogram(quantization_error);
```



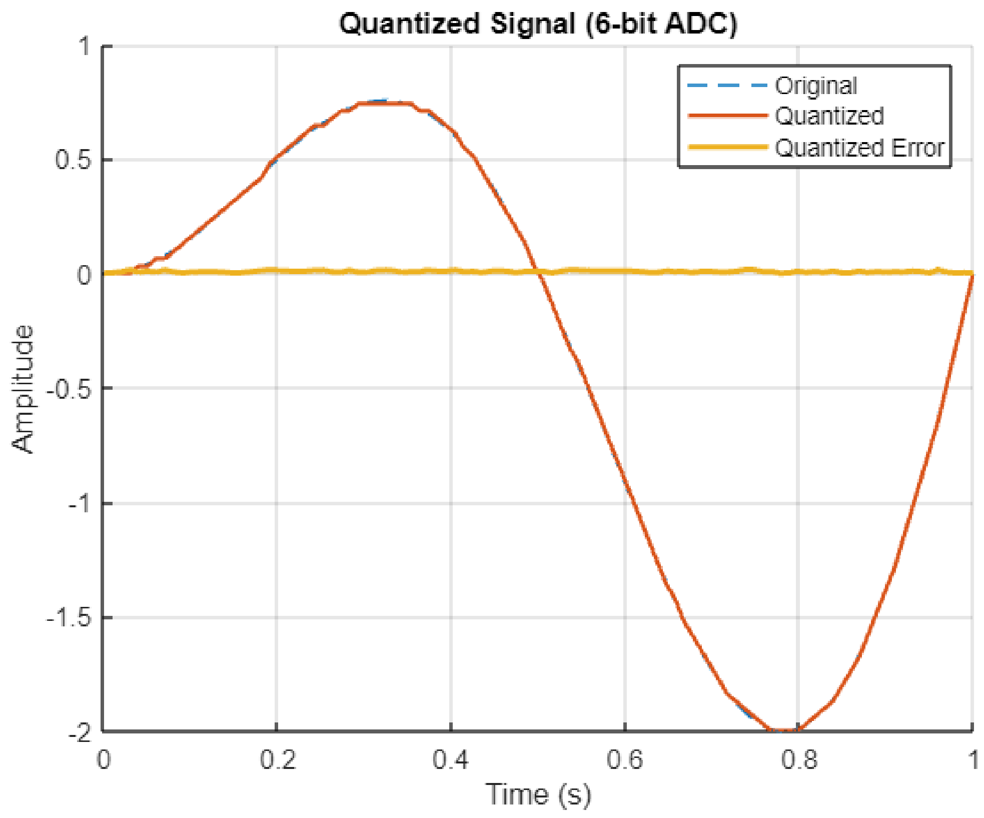
```
cuttoff_freq = 5;
 sinc_func = sinc(2*cuttoff_freq*t);
 LP = sinc_func/sum(sinc_func);
 reconstruction = filter(LP,1,quantized_signal);

 diff = output - reconstruction;
 figure;
 plot(t,output);
 hold on;
 plot(t,reconstruction);
 plot(t,diff);
 hold off;
 legend("The Original","Reconstructed","Difference");
 xlabel("time");ylabel("Amplitude");
 ylim([-2.5,1]);
 title("Original Signal with Reconstructed and the Difference b/w them", "Asghar")
 grid on;
```



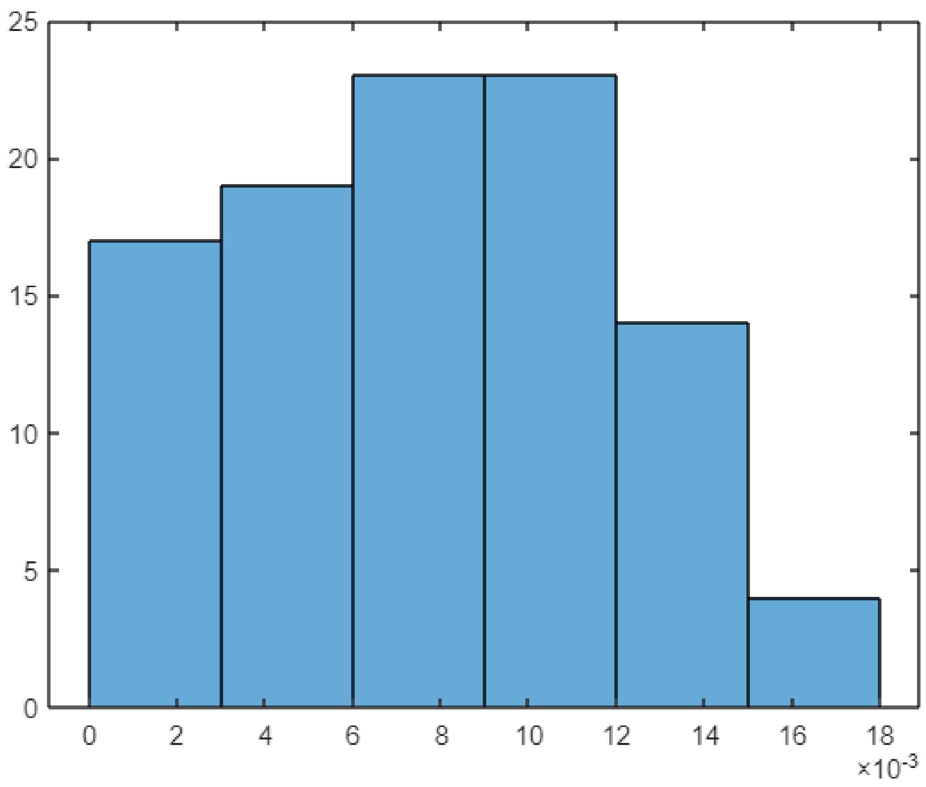
6-bit ADC

```
clear;
 % Define parameters
amplitude = 5;      % Amplitude of the sine wave
frequency = 1;      % Frequency of the sine wave (in Hz)
DUR = 1;       % Duration of the sine wave (in seconds)
sampling_rate = 100; % Sampling rate (number of samples per second)

% Calculate time vector
t = linspace(0, DUR, DUR * sampling_rate);
sine_wave = amplitude * sin(2 * pi * frequency * t);
ramp = linspace(0, 1, length(t));
output = sine_wave .* ramp;

% Add legends
legend('show');

bit_depth = 6;      % Number of bits for ADC quantization

% Condition the amplitude from -2 to 2

Mini_1 = min(output);
Max_1 = max(output);
Range = abs(Max_1-Mini_1);
```

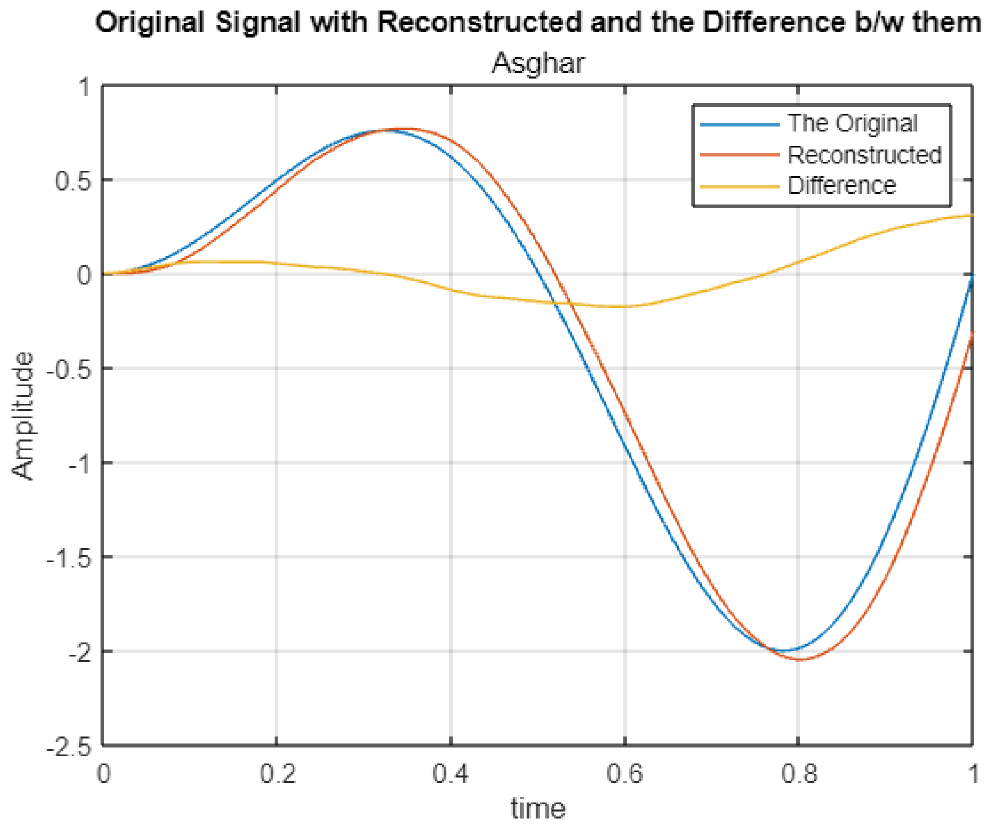
```
%c
output = output/max(abs(output));
output = output*2;


% Define parameters
num_levels = 2^(bit_depth-1); % Number of quantization levels

% Quantize the normalized signal using round function
normalized_signal = output;
%d
quantized_signal = round(normalized_signal * (num_levels - 1)) / (num_levels - 1);

% Calculate quantization error
% e
quantization_error = abs(output - quantized_signal);



% Plot the quantized signal
figure;
hold on;
plot(t, output, 'DisplayName', 'Original' , 'LineStyle', '--');
plot(t, quantized_signal, 'linewidth', 1.5, 'DisplayName', 'Quantized');
plot(t, quantization_error, 'LineWidth', 2 , 'DisplayName', 'Quantized Error');
hold off;
xlabel('Time (s)');
ylabel('Amplitude');
title('Quantized Signal (6-bit ADC)');
grid on;
legend('show');
```



```
%f
figure;
histogram(quantization_error);
```



```
cuttoff_freq = 5;
 sinc_func = sinc(2*cuttoff_freq*t);
 LP = sinc_func/sum(sinc_func);
 reconstruction = filter(LP,1,quantized_signal);

 diff = output - reconstruction;
 figure;
 plot(t,output);
 hold on;
 plot(t,reconstruction);
 plot(t,diff);
 hold off;
 legend("The Original","Reconstructed","Difference");
 xlabel("time");ylabel("Amplitude");
 ylim([-2.5,1]);
 title("Original Signal with Reconstructed and the Difference b/w them", "Asghar")
 grid on;
```



Short Explanation: Expanding the ADC bit depth enhances the quantization process, resulting in a closer approximation of the original waveform. This improved quantization enables a more faithful representation during reconstruction. With a higher bit count, the increased number of quantization levels allows for precise mapping of the original signal, ultimately minimizing information loss.

Longer Expalanation:

Bit Depth and Quantization

Bit depth refers to the number of bits used for each digital sample. It determines the resolution of the ADC, i.e., the number of possible digital values that can be used to represent an analog input. For instance, an ADC with a bit depth of 8 can represent 256 (2^8) different values.

Quantization is the process of mapping the infinite set of analog values to a finite set of digital values. This process inherently introduces some error, known as quantization error. The higher the bit depth, the lower the quantization error, resulting in a closer approximation of the original waveform.

Enhancing ADC Bit Depth

Expanding the ADC bit depth enhances the quantization process. With a higher bit count, the increased number of quantization levels allows for precise mapping of the original signal. This means that the digital representation of the signal will have a higher resolution, ultimately minimizing information loss.

Improved Signal Reconstruction

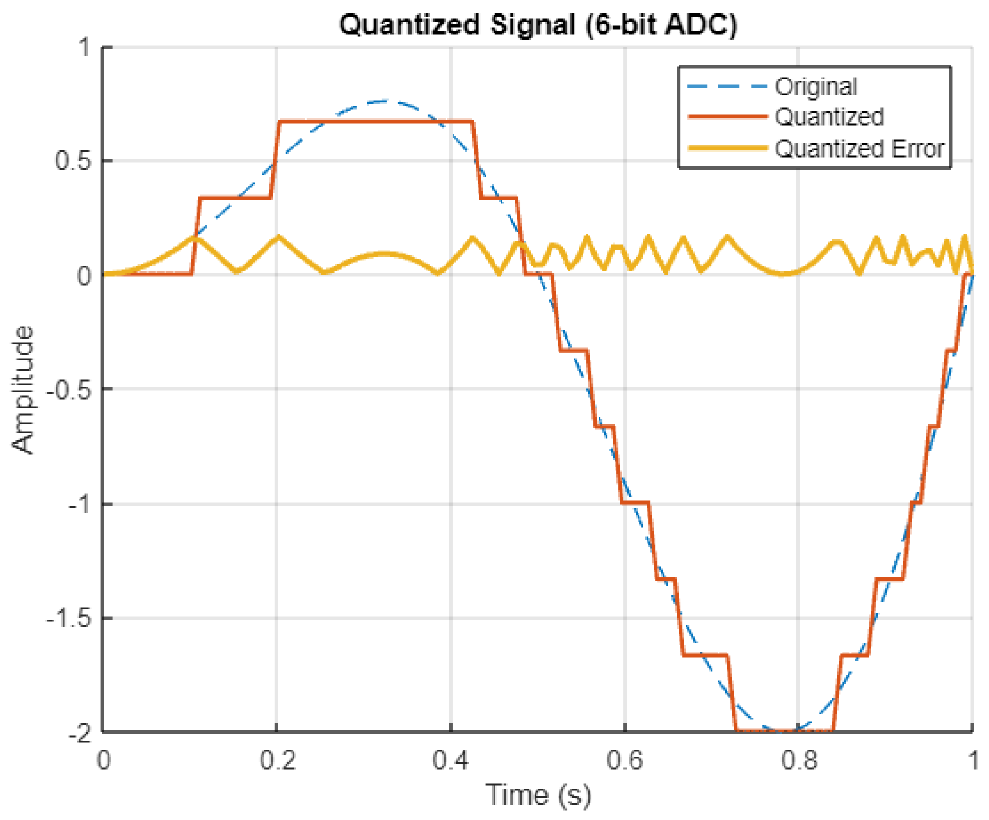
This improved quantization enables a more faithful representation during reconstruction. In digital systems, the digital signal must be converted back to an analog signal for it to be useful in real-world applications. This process is known as digital-to-analog conversion (DAC). The accuracy of the DAC process is directly related to the accuracy of the ADC process. Therefore, a more accurate ADC process (achieved through higher bit depth) results in a more accurate DAC process, leading to a more faithful representation of the original analog signal.

In conclusion, expanding the ADC bit depth is a critical aspect of improving the performance of digital systems. It enhances the quantization process, reduces information loss, and enables a more faithful representation of the original analog signal during reconstruction. This is particularly important in applications where high precision and accuracy are required.

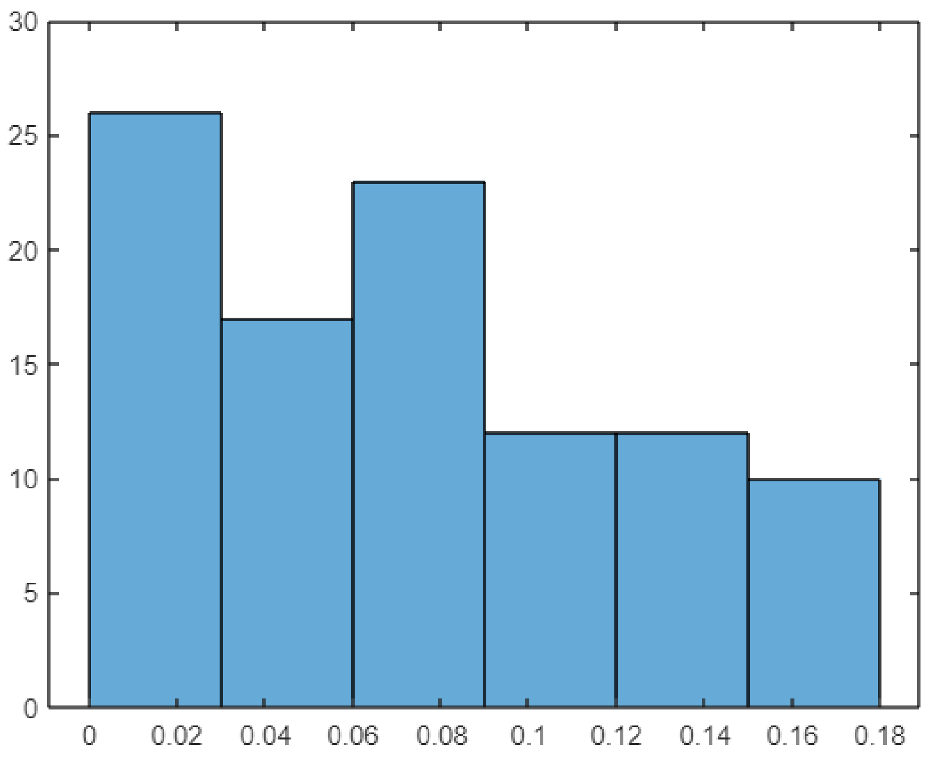
Task 3

```
clear;
 % Define parameters
amplitude = 5;      % Amplitude of the sine wave
frequency = 1;      % Frequency of the sine wave (in Hz)
DUR = 1;       % Duration of the sine wave (in seconds)
sampling_rate = 100; % Sampling rate (number of samples per second)

% Calculate time vector
t = linspace(0, DUR, DUR * sampling_rate);
sine_wave = amplitude * sin(2 * pi * frequency * t);
ramp = linspace(0, 1, length(t));
output = sine_wave .* ramp;

% Add legends
legend('show');

bit_depth = 3;      % Number of bits for ADC quantization

% Condition the amplitude from -2 to 2

Mini_1 = min(output);
Max_1 = max(output);
Range = abs(Max_1-Mini_1);

%c
output = output/max(abs(output));
output = output*2;


% Define parameters
num_levels = 2^(bit_depth-1); % Number of quantization levels

% Quantize the normalized signal using round function
normalized_signal = output;
```

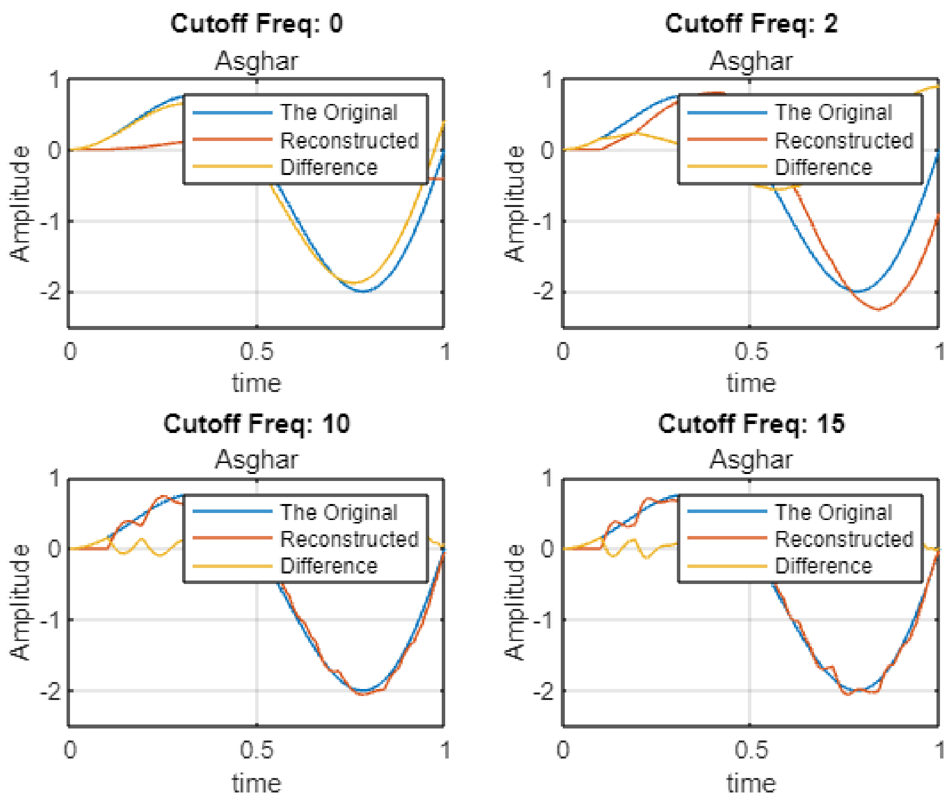
```
%d
quantized_signal = round(normalized_signal * (num_levels - 1)) / (num_levels - 1);

% Calculate quantization error
% e
quantization_error = abs(output - quantized_signal);



% Plot the quantized signal
figure;
hold on;
plot(t, output, 'DisplayName', 'Original' , 'LineStyle', '--');
plot(t, quantized_signal, 'linewidth', 1.5, 'DisplayName', 'Quantized');
plot(t, quantization_error, 'LineWidth', 2 , 'DisplayName', 'Quantized Error');
hold off;
xlabel('Time (s)');
ylabel('Amplitude');
title('Quantized Signal (6-bit ADC)');
grid on;
legend('show');
```



```
%f
figure;
histogram(quantization_error);
```



```
cutoff_freqs = [0, 2, 10, 15];
figure;

for i = 1:length(cutoff_freqs)
    CO_freq = cutoff_freqs(i);
    sinc_func = sinc(2*CO_freq*t);
    LP = sinc_func/sum(sinc_func);
    reconstruction = filter(LP,1,quantized_signal);

    diff = output - reconstruction;

    subplot(2,2,i);
    plot(t,output);
    hold on;
    plot(t,reconstruction);
    plot(t,diff);
    hold off;
    legend("The Original","Reconstructed","Difference");
    xlabel("time");ylabel("Amplitude");
    ylim([-2.5,1]);
    title(sprintf("Cutoff Freq: %d", CO_freq), "Asghar")
    grid on;
end
```



When adjusting the cutoff frequency of a perfect low-pass filter in signal processing, several important observations can be made regarding its effects on the reconstructed signal:

1. **Delay Reduction**: Increasing the cutoff frequency leads to a reduction in the delay of the reconstructed signal. This delay reduction indicates that the reconstructed signal aligns more closely in time with the original signal. As a result, any phase shifts introduced by the filtering process are minimized, leading to better alignment between the original and reconstructed signals.
2. **Phase Shift**: While increasing the cutoff frequency reduces the delay in the reconstructed signal, it may introduce a minor phase shift. This phase shift occurs due to the filtering process and can slightly alter the phase relationship between different frequency components of the signal. However, the phase shift is generally minimal in ideal low-pass filters.
3. **Decreased Precision**: As the cutoff frequency increases, the precision of the reconstructed signal may decrease. This decrease in precision is often associated with an increase in noise in the reconstructed signal. Higher cutoff frequencies allow more high-frequency components to pass through the filter, including noise present in the original signal or introduced during the quantization process.
4. **Increased Noise**: A higher cutoff frequency can result in more noise being present in the reconstructed signal. This noise may stem from quantization errors, sampling artifacts, or external interference. As a result, the reconstructed signal may more closely resemble the quantized signal, reducing the overall fidelity of the reconstruction.
5. **Optimal Reconstruction**: Despite the reduction in delay and potential phase shift, optimal signal reconstruction is often achieved at lower cutoff frequencies. Lower cutoff frequencies restrict the passage of high-frequency noise while still allowing essential components of the original signal to be reconstructed

accurately. By striking a balance between delay reduction and noise mitigation, lower cutoff frequencies generally yield reconstructed signals that closely resemble the original signal with minimal distortion.

In summary, manipulating the cutoff frequency of a perfect low-pass filter impacts the delay, phase shift, precision, and noise characteristics of the reconstructed signal. While increasing the cutoff frequency reduces delay and phase shift, it may also introduce noise and decrease precision, leading to suboptimal reconstruction. Optimal reconstruction is typically achieved at lower cutoff frequencies, where noise is minimized, and essential signal components are preserved with minimal distortion.

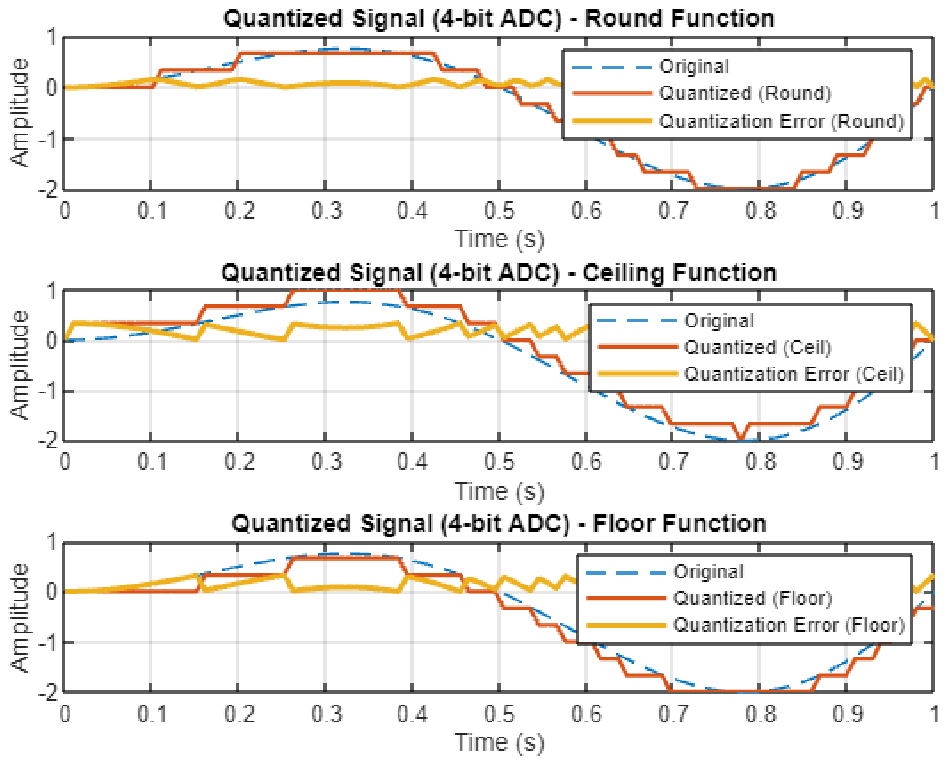
Task 4

3-bit ADC used and cutoff frequency = 5

```
clear;

% Define parameters
amplitude = 5;      % Amplitude of the sine wave
frequency = 1;      % Frequency of the sine wave (in Hz)
DUR = 1;       % Duration of the sine wave (in seconds)
sampling_rate = 100; % Sampling rate (number of samples per second)

% Calculate time vector
t = linspace(0, DUR, DUR * sampling_rate);
sine_wave = amplitude * sin(2 * pi * frequency * t);
ramp = linspace(0, 1, length(t));
output = sine_wave .* ramp;

% Add legends
legend('show');

bit_depth = 3;      % Number of bits for ADC quantization

% Condition the amplitude from -2 to 2

Mini_1 = min(output);
Max_1 = max(output);
Range = abs(Max_1-Mini_1);

% c
output = output/max(abs(output));
output = output*2;

% Define parameters
num_levels = 2^(bit_depth-1); % Number of quantization levels

% Quantize the normalized signal using round function
normalized_signal = output;
```

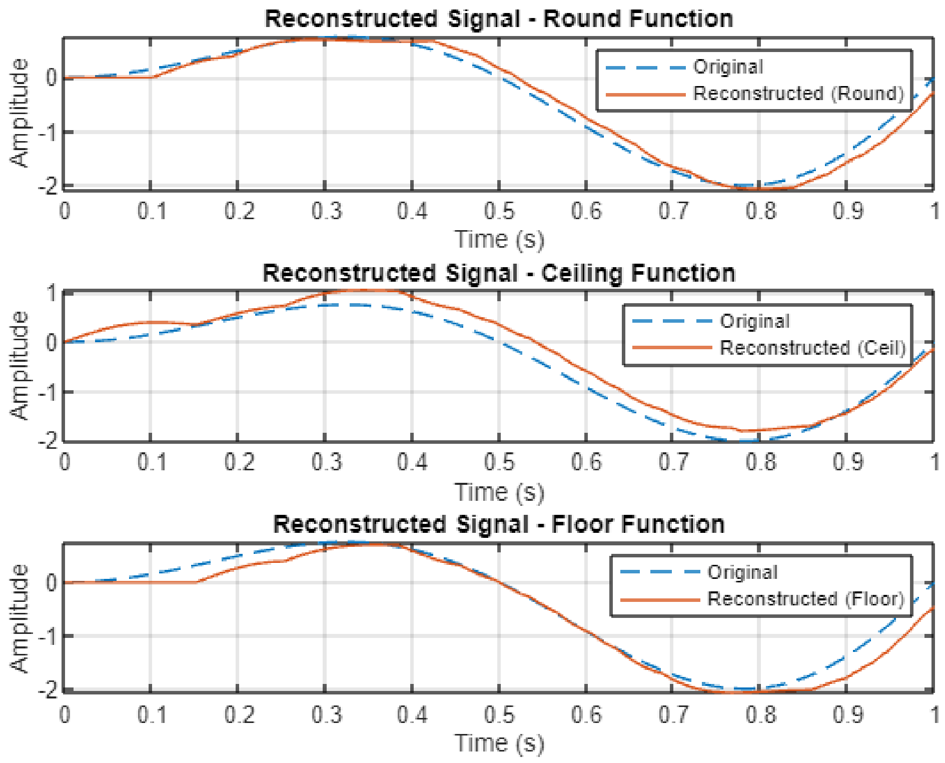
```
% d (round function)
quantized_signal_round = round(normalized_signal * (num_levels - 1)) / (num_levels
- 1);

% d (ceiling function)
quantized_signal_ceil = ceil(normalized_signal * (num_levels - 1)) / (num_levels -
1);

% d (floor function)
quantized_signal_floor = floor(normalized_signal * (num_levels - 1)) / (num_levels
- 1);

% Calculate quantization error
% e (round function)
quantization_error_round = abs(output - quantized_signal_round);

% e (ceiling function)
quantization_error_ceil = abs(output - quantized_signal_ceil);

% e (floor function)
quantization_error_floor = abs(output - quantized_signal_floor);

% Plot the quantized signals
figure;

% Plot for round function
subplot(3, 1, 1);
plot(t, output, 'DisplayName', 'Original' , 'LineStyle', '--');
hold on;
plot(t, quantized_signal_round, 'linewidth', 1.5, 'DisplayName', 'Quantized
(Round)');
plot(t, quantization_error_round, 'LineWidth', 2 , 'DisplayName', 'Quantization
Error (Round)');
hold off;
xlabel('Time (s)');
ylabel('Amplitude');
title('Quantized Signal (4-bit ADC) - Round Function');
grid on;
legend('show');

% Plot for ceiling function
subplot(3, 1, 2);
plot(t, output, 'DisplayName', 'Original' , 'LineStyle', '--');
hold on;
plot(t, quantized_signal_ceil, 'linewidth', 1.5, 'DisplayName', 'Quantized (Ceil)');
plot(t, quantization_error_ceil, 'LineWidth', 2 , 'DisplayName', 'Quantization
Error (Ceil)');
hold off;
xlabel('Time (s)');
ylabel('Amplitude');
```

```
title('Quantized Signal (4-bit ADC) - Ceiling Function');
grid on;
legend('show');

% Plot for floor function
subplot(3, 1, 3);
plot(t, output, 'DisplayName', 'Original' , 'LineStyle', '--');
hold on;
plot(t, quantized_signal_floor, 'linewidth', 1.5, 'DisplayName', 'Quantized
(Floor)');
plot(t, quantization_error_floor, 'LineWidth', 2 , 'DisplayName', 'Quantization
Error (Floor)');
hold off;
xlabel('Time (s)');
ylabel('Amplitude');
title('Quantized Signal (4-bit ADC) - Floor Function');
grid on;
legend('show');
```



```
% Reconstruction using Low Pass Filter

cuttoff_freq = 5;
sinc_func = sinc(2*cuttoff_freq*t);
LP = sinc_func/sum(sinc_func);

% Reconstruction using round function
```

```
reconstruction_round = filter(LP, 1, quantized_signal_round);

% Reconstruction using ceiling function
reconstruction_ceil = filter(LP, 1, quantized_signal_ceil);

% Reconstruction using floor function
reconstruction_floor = filter(LP, 1, quantized_signal_floor);

% Plotting the reconstructed signals
figure;

% Plot for round function
subplot(3, 1, 1);
plot(t, output, 'DisplayName', 'Original' , 'LineStyle', '--');
hold on;
plot(t, reconstruction_round, 'DisplayName', 'Reconstructed (Round)');
hold off;
xlabel('Time (s)');
ylabel('Amplitude');
title('Reconstructed Signal - Round Function');
grid on;
legend('show');

% Plot for ceiling function
subplot(3, 1, 2);
plot(t, output, 'DisplayName', 'Original' , 'LineStyle', '--');
hold on;
plot(t, reconstruction_ceil, 'DisplayName', 'Reconstructed (Ceil)');
hold off;
xlabel('Time (s)');
ylabel('Amplitude');
title('Reconstructed Signal - Ceiling Function');
grid on;
legend('show');

% Plot for floor function
subplot(3, 1, 3);
plot(t, output, 'DisplayName', 'Original' , 'LineStyle', '--');
hold on;
plot(t, reconstruction_floor, 'DisplayName', 'Reconstructed (Floor)');
hold off;
xlabel('Time (s)');
ylabel('Amplitude');
title('Reconstructed Signal - Floor Function');
grid on;
legend('show');
```



When quantizing a signal using different rounding functions such as "round", "floor", and "ceil", each function affects the quantization process and the resulting quantization error differently:

1. **Round Function**: The "round" function is a common choice for quantization as it rounds values to the nearest integer. This means that the quantization process takes into account both rounding up and rounding down, resulting in an approximation that tends to hover around zero. As a result, the mean quantization error is minimized, and the quantized signal **closely** resembles the original signal.
2. **Floor Function**: When using the "floor" function for quantization, values are rounded down to the nearest integer. This means that the quantization process shifts below the signal, leading to quantized values that are generally lower than the original signal. Consequently, the mean quantization error tends to be negative, as the quantized signal consistently **underestimates** the original signal.
3. **Ceil Function**: Conversely, the "ceil" function rounds values up to the nearest integer during quantization. This positions the quantization process above the signal, resulting in quantized values that are generally higher than the original signal. Consequently, the mean quantization error tends to be positive, as the quantized signal consistently **overestimates** the original signal.

Task 5

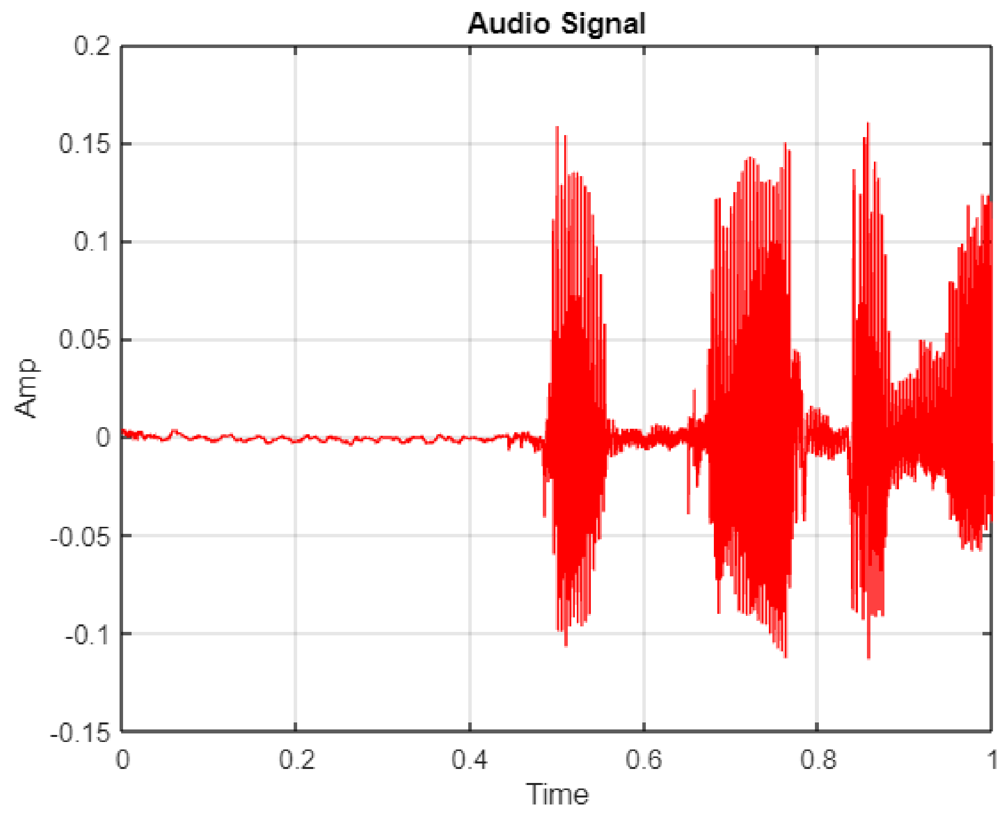
a)

```
% Define the file path of the audio file
audioFilePath = 'speech.wav';

% Read the audio file and extract the audio data (y) and sampling frequency (fs)
```

```
[y, fs] = audioread(audioFilePath);
```

b)

```
% Define the file path of the audio file
audioFilePath = 'speech.wav';

% Read the audio file and extract the audio data (y) and sampling frequency (fs)
[y, fs] = audioread(audioFilePath);

% Create a time vector representing the time in seconds for each sample in the
audio signal
time = (0:length(y)-1) / fs;

% Set the duration for plotting a segment of the audio (in seconds)
DUR = 1;

% Define the start index of the segment
SI = 1; %Start index

% Define the end index of the segment, ensuring it does not exceed the length of
the audio signal
EI = min(length(y), round(DUR * fs)); %end index

% Plot the segment of the audio signal from startIndex to endIndex against the
corresponding time values
figure;
plot(time(SI:EI), y(SI:EI),'LineWidth', 0.5, 'Color', 'r');
title('Audio Signal');
xlabel('Time');
ylabel('Amp');
grid on;
```



This MATLAB code segment begins by specifying the file path for an audio file named "speech.wav" and then reads the audio data and its sampling frequency using the `audioread` function. Subsequently, it creates a time vector to represent the time in seconds for each sample in the audio signal. A segment of the audio signal, determined by the specified duration (`DUR`), is then plotted against the corresponding time values, **allowing for visualization of the signal's amplitude variation over time**. The plot is customized with a red color line, labeled axes, a title, and grid lines for clarity. Overall, this code facilitates the visualization of a portion of the audio signal, aiding in the analysis of its temporal characteristics

c)

```
% Scale the audio signal y so that its maximum absolute value becomes 1
Scaled_Y = y / max(abs(y));

% Increase the amplitude of the normalized signal by a factor of 2
Y_Cond = Scaled_Y .* 2;

% Define the number of bits for quantization
NB = 2; %Num_Bits

% Calculate the number of quantization levels based on the number of bits
Q_Levs = 2^(NB);
```

In this MATLAB snippet, the audio signal 'y' is first scaled so that its maximum absolute value becomes 1, facilitating consistent signal processing. This normalized signal, stored in `Scaled_Y`, is then amplified by a factor of 2 to increase its amplitude, stored in `Y_Cond`. Subsequently, the number of bits for quantization (`NB`) is defined as 2, determining the precision of the quantization process. Based on this bit depth, the number of quantization levels (`Q_Levs`) is calculated as 2 raised to the power of the number of bits.

This preparatory step sets the stage for further signal processing, such as quantization, enabling accurate representation and manipulation of the audio signal within the defined bit resolution.

d)

This MATLAB code segment calculates the maximum and minimum values of the conditioned signal `Y_Cond`, computes the signal's range, normalizes the conditioned signal to the range [0, 1], quantizes the normalized signal to a specified number of quantization levels (`Q_Levs`), and adjusts the quantized signal to fit within the original range of the conditioned signal.

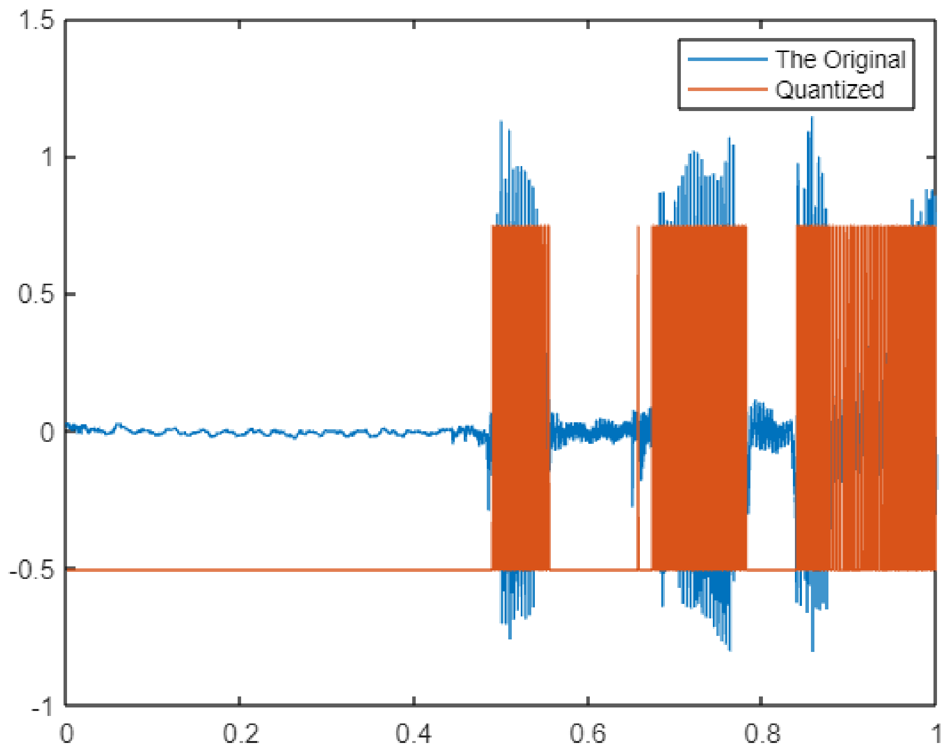
```
% Find the maximum and minimum values of the conditioned signal y_cond
Max_1 = max(Y_Cond);
Mini_1 = min(Y_Cond);

% Calculate the range of the conditioned signal
Range = abs(Max_1 - Mini_1);

% Normalize the conditioned signal to the range [0, 1]
Normalized_Y = (Y_Cond - Mini_1) / (Max_1 - Mini_1);

% Quantize the normalized signal to num_bits number of quantization levels
(q_levels)
Quantized_Y = round(Normalized_Y * (Q_Levs - 1)) * Range / (Q_Levs - 1) + Mini_1;
```

e)

This MATLAB code segment creates a new figure for plotting, overlays the conditioned signal `Y_Cond` and the quantized signal `Quantized_Y` against time, adds a legend to differentiate between the original and quantized signals.

```
% Create a new figure for plotting
figure;

% Plot the conditioned signal y_cond against time
plot(time(SI:EI), Y_Cond(SI:EI))

% Hold the current plot so that the next plot can be overlaid
hold on;

% Plot the quantized signal y_quantized against time
plot(time(SI:EI), Quantized_Y(SI:EI))

% Release the hold on the plot
```

```
hold off;

% Add a legend to the plot to differentiate between the original and quantized
signals
legend("The Original", "Quantized");
```



f)

This code segment creates an audioplayer object for both the conditioned signal `Y_Cond` and the quantized signal `Quantized_Y`. It then plays the conditioned audio signal, pauses execution for 7 seconds to allow listening, creates an audioplayer object for the quantized signal, and finally plays the quantized audio signal.

```
% Create an audioplayer object for the conditioned signal y_cond
OG_Audio = audioplayer(Y_Cond, fs);

% Play the conditioned audio signal
play(OG_Audio);

% Pause execution for 7 seconds to allow listening
pause(7);

% Create an audioplayer object for the quantized signal y_quantized
Q_Audio = audioplayer(Quantized_Y, fs);

% Play the quantized audio signal
play(Q_Audio);
```

g)

The code defines a cutoff frequency `CO_freq` and generates a sinc function `sinc_func` using MATLAB's `sinc` function.

```
% Define the cutoff frequency
CO_freq = 20;

% Create a sinc function with a width proportional to the cutoff frequency
% The sinc function is used to design low-pass filters in signal processing
sinc_func = sinc(2 * CO_freq * time);
```

h)

The code normalizes the sinc function `sinc_func` by dividing it by the sum of its values, ensuring that the resulting low-pass filter `LP` preserves the overall energy of the signal.

```
% Normalize the sinc function by dividing it by the sum of its values
% This ensures that the filter preserves the overall energy of the signal
LP = sinc_func / sum(sinc_func);
```

i)

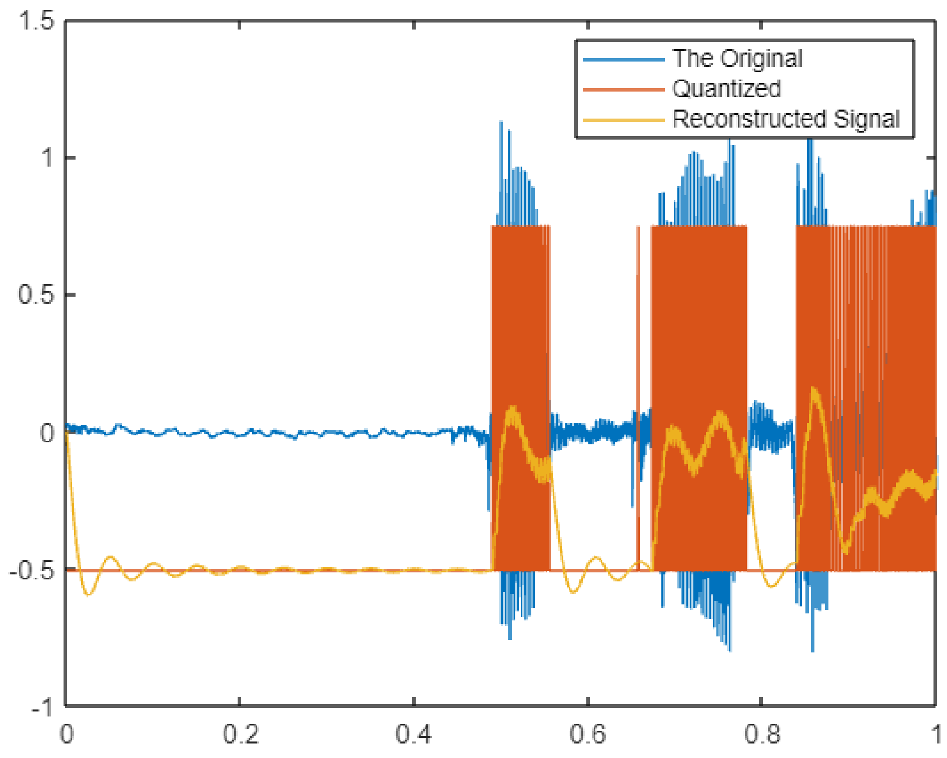
The code applies the low-pass filter `LP` to the quantized signal `Quantized_Y` using the `filter` function. The '1' argument specifies that the filter is a single-input single-output (SISO) filter. The resulting reconstructed signal is stored in the variable `RE`.

```
% Use the filter function to apply the low-pass filter (lpf) to the quantized
signal (y_quantized)
% The '1' argument indicates that the filter is a single-input single-output (SISO)
filter
% 'reconst' will contain the filtered signal
RE = filter(LP, 1, Quantized_Y); %Reconstruct
```

j) Plotting

```
% Create a new figure for plotting
figure;

% Plot the conditioned signal (y_cond) against time
plot(time(SI:EI), Y_Cond(SI:EI));

% Hold the current plot so that the next plots can be overlaid
hold on;
```

```
% Plot the quantized signal (y_quantized) against time
plot(time(SI:EI), Quantized_Y(SI:EI));

% Plot the reconstructed signal (reconst) against time
plot(time(SI:EI), RE(SI:EI));

% Release the hold on the plot
hold off;

% Add a legend to the plot to differentiate between the original, quantized, and
reconstructed signals
legend("The Original", "Quantized", "Reconstructed Signal");
```



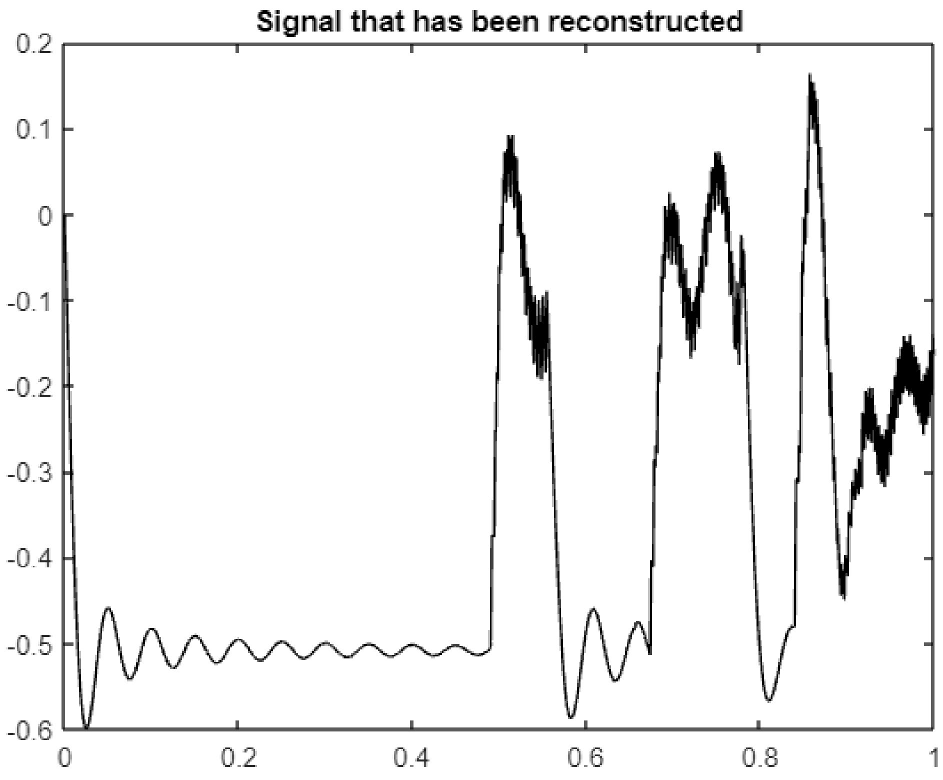
k)

playing the reconstructed audio

```
% Create an audioplayer object for the reconstructed signal (reconst)
reconstructed_audio = audioplayer(RE, fs);

% Play the reconstructed audio signal
play(reconstructed_audio);
```

l)

```
% Clear the current figure window
clf;

% Plot the reconstructed signal (reconst) against time
plot(time(SI:EI), RE(SI:EI), 'Color', 'k');

% Add a title to the plot
title("Signal that has been reconstructed");
```



m)

The signal was initially subjected to quantization, which essentially means it was simplified to only four levels, akin to the process carried out by a 2-bit analog-to-digital converter (ADC). This quantization inherently led to a discernible loss of information.

Following the quantization process, the signal then underwent filtration through a low-pass filter, where frequencies above a certain threshold (in this case, 20 Hz) were attenuated while lower frequencies were preserved.

Despite efforts to reconstruct the signal, it's important to note that the resulting waveform, while exhibiting a similar shape and delay as the original, displayed disparities in amplitudes. This discrepancy in amplitudes indicates that a considerable amount of information was lost during the quantization and subsequent filtration processes compared to the fidelity of the original signal.

3-bit ADC

```
% Define the number of bits for quantization
NB = 3;

% Calculate the number of quantization levels based on the number of bits
Q_Levs = 2^(NB);

% Find the maximum and minimum values of the conditioned signal y_cond
Max_1 = max(Y_Cond);
Mini_1 = min(Y_Cond);

% Calculate the range of the conditioned signal
Range = abs(Max_1 - Mini_1);

% Normalize the conditioned signal to the range [0, 1]
Normalized_Y = (Y_Cond - Mini_1) / (Max_1 - Mini_1);

% Quantize the normalized signal to num_bits number of quantization levels
(q_levels)
Quantized_Y = round(Normalized_Y * (Q_Levs - 1)) * Range / (Q_Levs - 1) + Mini_1;

% Define the cutoff frequency for the low-pass filter
CO_freq = 25;

% Create a sinc function with a width proportional to the cutoff frequency
sinc_func = sinc(2 * CO_freq * time(SI:EI));

% Normalize the sinc function by dividing it by the sum of its values
LP = sinc_func / sum(sinc_func);

% Use the filter function to apply the low-pass filter (lpf) to the quantized
signal (y_quantized)
% The '1' argument indicates that the filter is a single-input single-output (SISO)
filter
% 'reconst' will contain the filtered signal
RE = filter(LP, 1, Quantized_Y);

% Create an audioplayer object for the reconstructed signal (reconst)
reconstructed_audio = audioplayer(RE, fs);

% Play the reconstructed audio signal
play(reconstructed_audio);
```

4-bit ADC

```
% Define the number of bits for quantization
NB = 4;
```

```
% Calculate the number of quantization levels based on the number of bits
Q_Levs = 2^(NB);

% Find the maximum and minimum values of the conditioned signal y_cond
Max_1 = max(Y_Cond);
Mini_1 = min(Y_Cond);

% Calculate the range of the conditioned signal
Range = abs(Max_1 - Mini_1);

% Normalize the conditioned signal to the range [0, 1]
Normalized_Y = (Y_Cond - Mini_1) / (Max_1 - Mini_1);

% Quantize the normalized signal to num_bits number of quantization levels
(q_levels)
Quantized_Y = round(Normalized_Y * (Q_Levs - 1)) * Range / (Q_Levs - 1) + Mini_1;

% Define the cutoff frequency for the low-pass filter
CO_freq = 25;

% Create a sinc function with a width proportional to the cutoff frequency
sinc_func = sinc(2 * CO_freq * time(SI:EI));

% Normalize the sinc function by dividing it by the sum of its values
LP = sinc_func / sum(sinc_func);

% Use the filter function to apply the low-pass filter (lpf) to the quantized
signal (y_quantized)
% The '1' argument indicates that the filter is a single-input single-output (SISO)
filter
% 'reconst' will contain the filtered signal
RE = filter(LP, 1, Quantized_Y);

% Create an audioplayer object for the reconstructed signal (reconst)
reconstructed_audio = audioplayer(RE, fs);

% Play the reconstructed audio signal
play(reconstructed_audio);
```

When designing the filter with a cutoff frequency of 20, I observed minimal delay in the reconstructed signal compared to the original. However, the reconstructed signal exhibited diminished amplitudes, indicating a loss of information. Despite this loss, the audio of the reconstructed signal, while not as loud as the original, remained intelligible and effectively conveyed the intended message.

CONCLUSION:

Firstly, the audio signal underwent quantization, reducing it to only four levels, simulating the effect of a 2-bit analog-to-digital converter (ADC). This quantization led to a noticeable loss of information in the signal.

Subsequently, the quantized signal was subjected to a low-pass filter with a cutoff frequency of 20 Hz. This filtration process aimed to preserve lower frequencies while attenuating higher frequencies.

Despite efforts to reconstruct the signal, it was found that while the resulting waveform exhibited a similar shape and minimal delay compared to the original, there were disparities in amplitudes. This discrepancy suggested that a significant amount of information was lost during the quantization and filtration processes, impacting the fidelity of the reconstructed signal.

Further experimentation with different bit values—from 2 to 3 and 4—did not yield significant improvements in the reconstruction quality.

In conclusion, while the reconstruction process successfully retained the general structure and timing of the original signal, the loss of information resulted in diminished amplitudes and potential distortion. Nonetheless, by carefully selecting the quantization bit value and applying appropriate filtration techniques, it is possible to achieve a reconstructed signal that effectively conveys the intended message with minimal loss of fidelity.