DSP lab 05

Name: Basil khowaja
id: bk08432

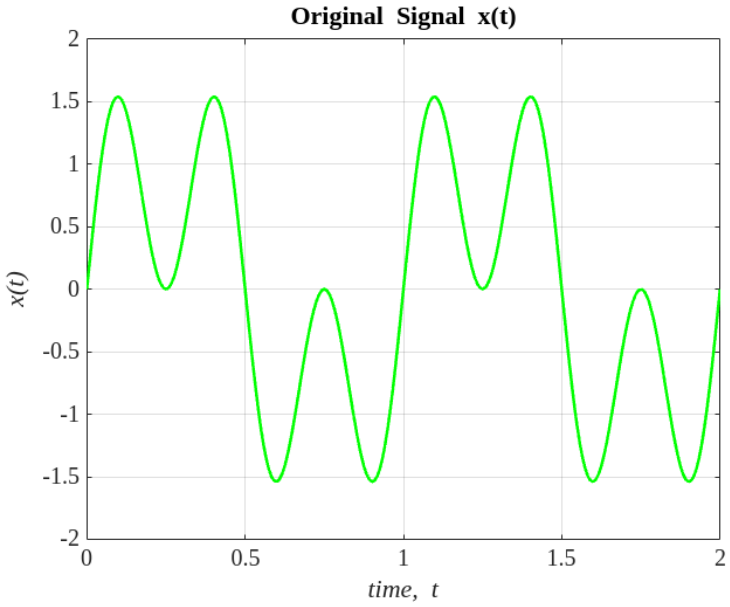
Activity 1)

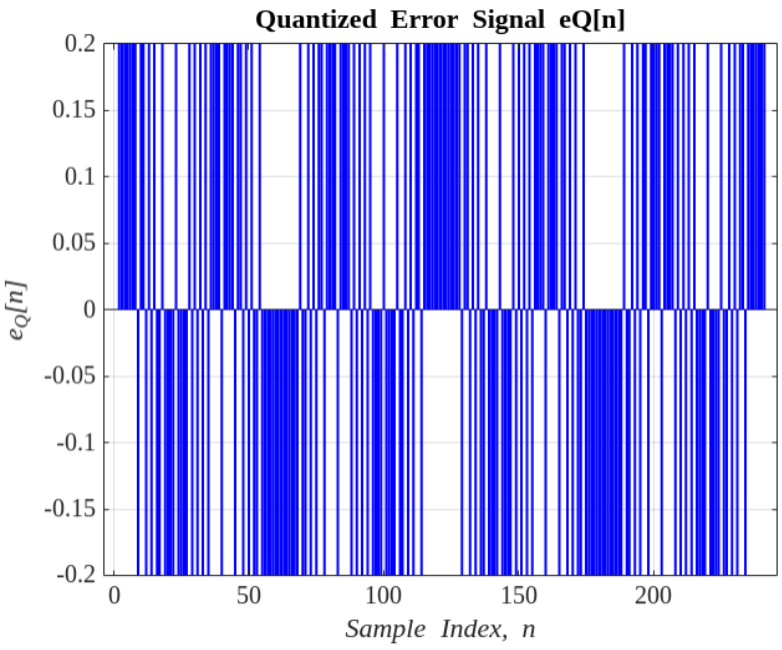
code:

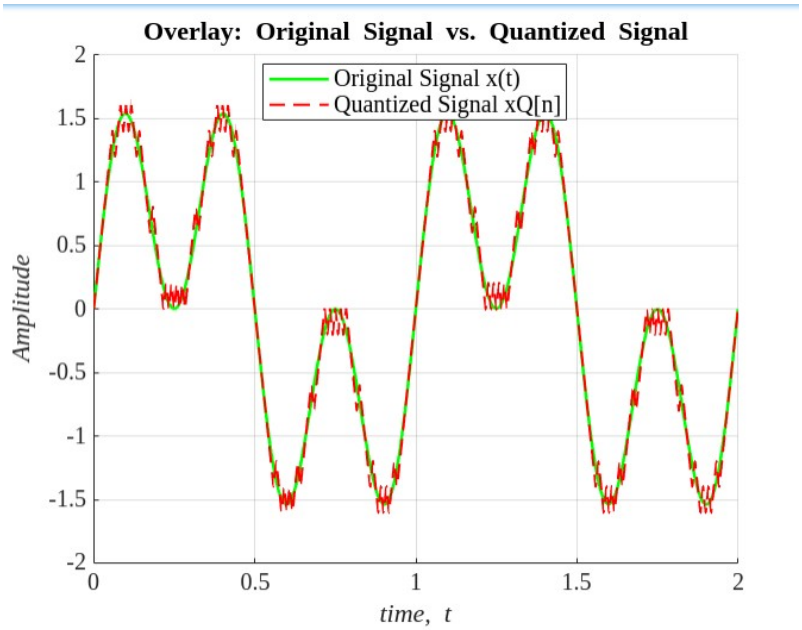
```
clc; clear; close all;
A1 = 1;
A3 = 1;
f1 = 1;
f3 = 3*f1;
K2 = 8;
K1 = 5;
fS = K1 * K2 * f3;


Tmax = 2;
Ts = 1/fS;
t = 0:Ts:Tmax;
n_samples = length(t);

x = A1 * cos(2 * pi * f1 * t - pi/2) + A3 * cos(2 * pi * f3 * t - pi/2);

xQ = zeros(1, n_samples);
eQ = zeros(1, n_samples);
Delta = 0.2;

for n = 2:n_samples
    e = x(n) - xQ(n-1);
    eQ(n) = Delta * sign(e);
    xQ(n) = xQ(n-1) + eQ(n);
end

figure;
plot(t, x, 'g', 'LineWidth', 1.5);
xlabel('\it{time, t}', 'FontSize', 14);
ylabel('\it{x(t)}', 'FontSize', 14);
title('\bf{Original Signal x(t)}', 'FontSize', 16);
grid on;
set(gca, 'FontSize', 12, 'FontName', 'Times New Roman');

figure;
stem(1:n_samples, eQ, 'b', 'Marker', 'none', 'LineWidth', 1.2);
xlabel('\it{Sample Index, n}', 'FontSize', 14);
ylabel('\it{e_Q[n]}', 'FontSize', 14);
title('\bf{Quantized Error Signal eQ[n]}', 'FontSize', 16);
grid on;
```

```
set(gca, 'FontSize', 12, 'FontName', 'Times New Roman');

figure;
hold on;
plot(t, x, 'g', 'LineWidth', 1.5);
plot(t, xQ, 'r--', 'LineWidth', 1.2);
hold off;
xlabel('\it{time, t}', 'FontSize', 14);
ylabel('\it{Amplitude}', 'FontSize', 14);
title('\bf{Overlay: Original Signal vs. Quantized Signal}', 'FontSize', 16);
legend({'Original Signal x(t)', 'Quantized Signal xQ[n]'}, 'FontSize', 12, 'Location', 'best');
grid on;
set(gca, 'FontSize', 12, 'FontName', 'Times New Roman');
```

Results:







Observation:

Initially, our task is to define the signal's parameters. Our base frequency f1 is 1 Hz and our amplitudes A1 and A3 are set to 1 as well. Three times f1 defines the second frequency f3, therefore adding a higher-frequency component to the signal. We also define K1 = 5 and K2 = 8, which assist to ascertain the sampling frequency fS. By selecting fS = K1 * K2 * f3, we guarantee a sufficiently high sampling rate to record all signal fluctuations free from aliasing. This stage guarantees that our input signal replics continuous signals found in the actual world, such sensor readings or sound waves.

We then create the time vector t spanning 0 to 2 seconds using a Ts = 1/fS step size. This defines the time instances at which we sample the waveform, hence producing a discrete-time representation of the signal. We next build the original signal x(t), a sum of two cosine waves with varying frequency. This signal stands for the ongoing input we wish to convert into digital form.

We so create two arrays: eQ[n] for the quantization error and xQ[n] for the quantized signal. The quantization step size Δ = 0.2 is another definition that controls the extent of change in the quantized value at every step. This stage helps one become ready for Sigma-Delta modulation, hence adopting a feedback system helps to lower long-term quantization mistakes.

We then start a cycle handling every signal sample. We compute the error e = x(n) - xQ(n-1) at every step to determine the real signal's variation from the previously quantified value. We next use quantization, varying the quantized result in discrete steps: eQ(n) = Δ * sign(e), therefore depending on whether the error is positive or negative we either increase or reduce the quantized value by Δ. At last, we update xQ[n] by xQ(n) = xQ(n-1) + eQ(n), so assuring that the quantized signal progressively follows the original signal and maintains track of accumulated errors.

We then graph the original signal x(t), therefore enabling our visualization of the continuous waveform prior to quantization. This stage offers a means of comparison for how effectively the quantization technique preserves the original signal's form.

We then stem plot the quantization error signal eQ[n]. Usually switching fast between positive and negative values, this graph displays how the mistake changes with time. Practically speaking, this

map indicates the degree of variation between the original and quantized signals at every phase. The fast switching suggests that the quantization noise is distributed over a large frequency spectrum, therefore lessening its effect on any one frequency component.
We last graph the overlap of the quantized and original signals. Red dashes depict the quantized signal; the original signal is presented in green. This graph lets us practically compare the two signals. The quantized signal, we find, closely resembles the original waveform but with minor step-like fluctuations. For high-fidelity digital conversion in applications such audio processing and telecommunications, Sigma-Delta modulation is therefore a valuable method since it essentially tracks the original signal while lowering the total quantization noise.

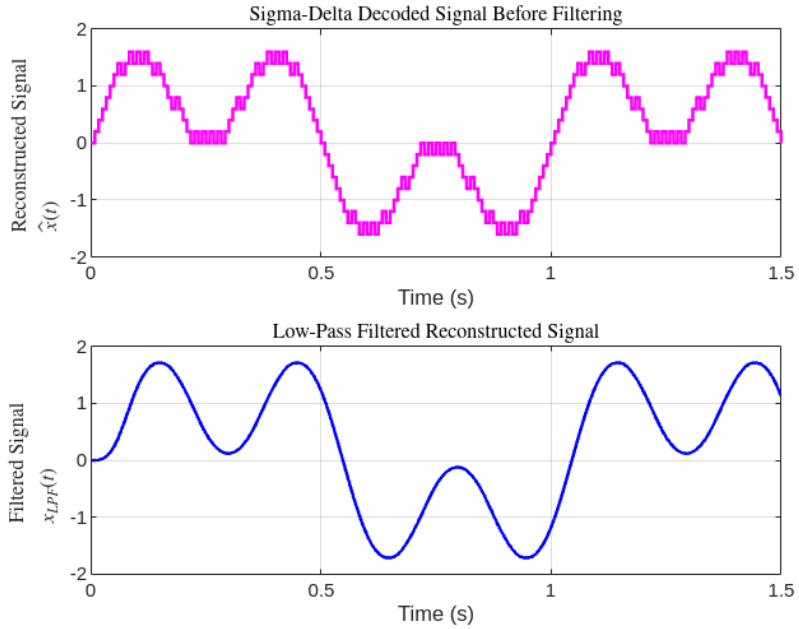
Activity 2)

code:

```
clc; clear; close all;

A1 = 1;
A3 = 1;
f1 = 1;
f3 = 3 * f1;
K2 = 8;
K1 = 5;
fS = K1 * K2 * f3;

Tmax = 1.5;
Ts = 1 / fS;
t = 0:Ts:Tmax;
n_samples = length(t);
x = A1 * cos(2 * pi * f1 * t - pi/2) + A3 * cos(2 * pi * f3 * t - pi/2);

xQ = zeros(1, n_samples);
eQ = zeros(1, n_samples);
Delta = 0.2;

for n = 2:n_samples
   e = x(n) - xQ(n-1);
   eQ(n) = Delta * sign(e);
   xQ(n) = xQ(n-1) + eQ(n);
end

x_hat = cumsum(eQ);
M = 5;
x_hat_padded = zeros(1, M * length(x_hat));
x_hat_padded(1:M:end) = x_hat;
t_padded = linspace(0, Tmax, length(x_hat_padded));

p = 6;
```

```
dt = Ts / M;
h = sinc([-p*M:p*M] * fS * dt / p) / p;

x_LPF = filter(h, 1, x_hat_padded);

figure;

subplot(2,1,1);
stairs(t, x_hat, 'm', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Reconstructed Signal $$\hat{x}(t)$$', 'Interpreter', 'latex');
title('Sigma-Delta Decoded Signal Before Filtering', 'Interpreter', 'latex');
grid on;
xlim([0 Tmax]); ylim([-2 2]);

subplot(2,1,2);
plot(t_padded, x_LPF, 'b', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Filtered Signal $$x_{LPF}(t)$$', 'Interpreter', 'latex');
title('Low-Pass Filtered Reconstructed Signal', 'Interpreter', 'latex');
grid on;
xlim([0 Tmax]); ylim([-2 2]);
```

Output:



observation:

The first subplot presents a classic stair-step-looking Sigma-Delta modulated signal. This happens because, employing a sort of differential quantization, Sigma-Delta modulation tracks changes (differences) in the input signal rather than directly encoding absolute amplitude values. The modulator changes the output in little discrete steps (Δ levels) by constantly comparing the input signal to an estimated (already quantized) copy.

The system basically stores the changes over time rather than exact values. As it tries to resemble the input, this produces a step-like waveform whereby the output rises or falls progressively. Sigma-Delta modulation prioritizes conserving low-frequency content (slow variations in amplitude) while introducing high-frequency noise (quantization noise) that can subsequently be eliminated. The strong sampling rate relative to the frequency components of the data results in the high-frequency aberrations seen in the stepwise pattern.

From a signal processing standpoint, instead of focusing quantization noise at lower frequencies as in traditional Pulse Code Modulation (PCM), this modulation technique distributes it over a wide frequency range. This method has the advantage in that most of the noise is directed toward higher frequencies, from which a basic low-pass filter can readily eliminate it. This is the reason the modulated signal seems jagged yet still has the general form of the original waveform.
Physically, the staircase pattern implies that the signal is fit for digital transmission, in which case binary sequences reflect step transitions. In digital systems, this is a benefit since it enables low-bit quantizer-based high-resolution encoding usually just 1-bit in use. Nevertheless, a low-pass filter is required since this type of encoding is not directly helpful for playback or analog processing without further reconstruction stages.

Applying a low-pass filter (LPF) to the modulated signal shows in the second subplot as This filtering stage serves primarily to eliminate the high-frequency quantization noise generated by the Sigma-Delta modulation technique while maintaining the real signal components. Successful reconstruction is shown by a smooth waveform that quite closely reflects the original input signal. From a frequency-domain standpoint, the modulated signal before filtering comprises both undesired high-frequency quantization noise and original low-frequency signal components. Acting as a brick-wall filter, the sinc-based low-pass filter removes frequencies beyond a given threshold therefore letting just the meaningful low-frequency content pass through. This filtering procedure turns the staircase-like waveform into a smooth sinusoidal waveform, therefore restoring the constant character of the original signal.

Mathematically, where the signal is smoothed by averaging neighboring samples depending on the impulse response of the filter, the filtering operation is identical to convolution in the time domain. Practically speaking, this results in the attenuation of the high-frequency noise components, therefore leaving just the fundamental frequency components of the original signal. The second plot clearly shows the improvement in signal quality since the reconstructed waveform there is free of the sharp transitions observed in the first image.

This filtering phase physically replics how digital-to- analog converters (DACs) rebuild audio signals for playback in electrical devices. A DAC uses a similar filtering technique to recover the original waveform before passing it to an amplifier or speaker when it gets a digital bitstream encoded using Sigma-Delta modulation. Accurate reconstruction of the signal following modulation emphasizes the benefit of Sigma-Delta methods in lowering hardware complexity without sacrificing great fidelity.

Activity 3)

code:

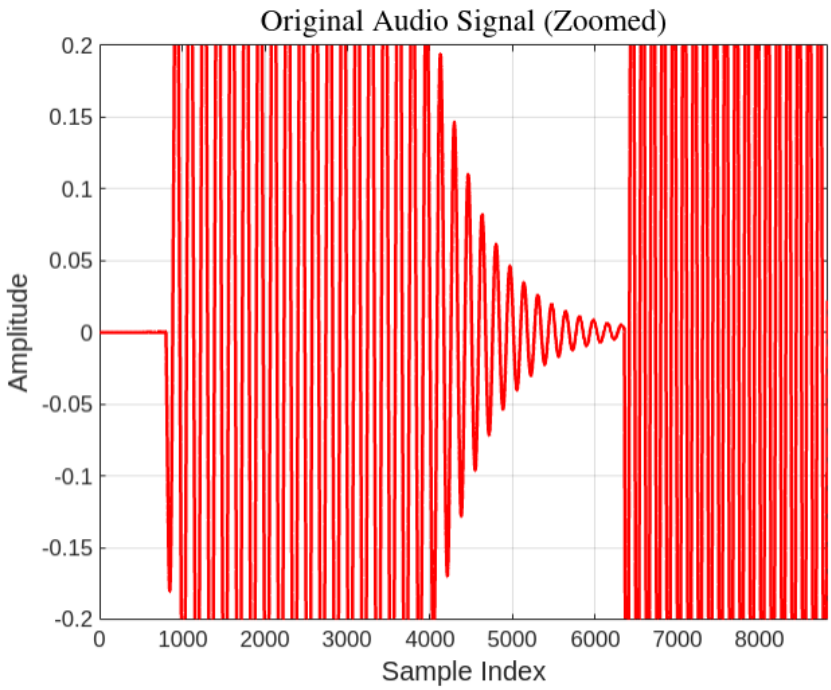
```
clc; clear; close all;

[audio_in, Fs] = audioread('lab5.mp3');
audio_in = mean(audio_in, 2);
audio_in = audio_in / max(abs(audio_in));
filter_order = 6;
[b, a] = butter(filter_order, 0.9, 'low');
audio_filtered = filtfilt(b, a, audio_in);

Fs_new = 4 * Fs;

t_original = (0:length(audio_filtered)-1) / Fs;
t_resampled = (0:length(audio_filtered)*Fs_new/Fs-1) / Fs_new;
audio_resampled = interp1(t_original, audio_filtered, t_resampled, 'spline');

delta = 0.05;
xQ = zeros(size(audio_resampled));
eQ = zeros(size(audio_resampled));

for n = 2:length(audio_resampled)
    e_n = audio_resampled(n) - xQ(n-1);
    eQ(n) = delta * sign(e_n);
    xQ(n) = xQ(n-1) + eQ(n);
end

x_hat = cumsum(eQ);

[b_lpf, a_lpf] = butter(6, 0.1, 'low');
x_LPF = filtfilt(b_lpf, a_lpf, x_hat);
x_LPF = x_LPF / max(abs(x_LPF));
t_new = (0:length(x_LPF)-1) / Fs_new;
t_original = (0:length(x_LPF)*Fs/Fs_new-1) / Fs;
x_reconstructed = interp1(t_new, x_LPF, t_original, 'spline');

[correlation, lags] = xcorr(audio_in, x_reconstructed);
[~, max_index] = max(correlation);
shift_value = lags(max_index);
```

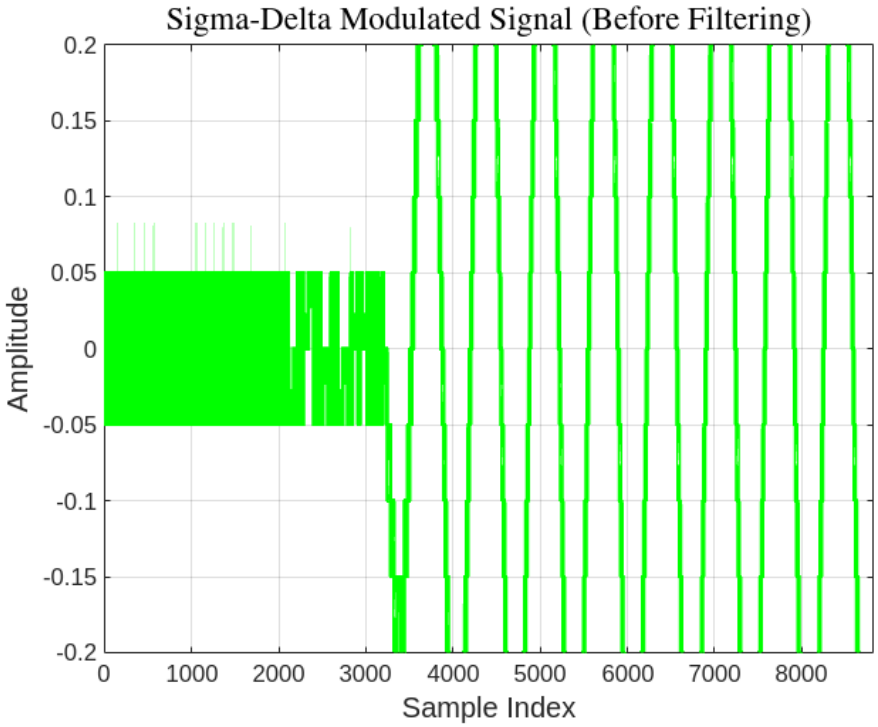
```
x_reconstructed = circshift(x_reconstructed, -shift_value);

disp('Playing Original Audio...');
sound(audio_in, Fs);
pause(length(audio_in)/Fs + 1);

disp('Playing Reconstructed Audio...');
sound(x_reconstructed, Fs);
pause(length(x_reconstructed)/Fs + 1);

original_size = whos('audio_in');
modulated_size = whos('x_hat');
reconstructed_size = whos('x_reconstructed');

fprintf('\nMemory Usage Report:\n');
fprintf('Original Audio: %.2f KB\n', original_size.bytes / 1024);
fprintf('After Modulation (Before Filtering): %.2f KB\n', modulated_size.bytes / 1024);
fprintf('After Final Reconstruction: %.2f KB\n', reconstructed_size.bytes / 1024);

samples_to_plot = round(0.2 * Fs);

figure;
plot(audio_in(1:samples_to_plot), 'r', 'LineWidth', 1.5);
title('Original Audio Signal (Zoomed)', 'Interpreter', 'latex', 'FontSize', 14);
xlabel('Sample Index', 'FontSize', 12);
ylabel('Amplitude', 'FontSize', 12);
xlim([0 samples_to_plot]);
ylim([-0.2 0.2]);
grid on;

figure;
plot(x_hat(1:samples_to_plot), 'g', 'LineWidth', 1.5);
title('Sigma-Delta Modulated Signal (Before Filtering)', 'Interpreter', 'latex', 'FontSize', 14);
xlabel('Sample Index', 'FontSize', 12);
ylabel('Amplitude', 'FontSize', 12);
xlim([0 samples_to_plot]);
ylim([-0.2 0.2]);
grid on;

figure;
plot(x_reconstructed(1:samples_to_plot), 'b', 'LineWidth', 1.5);
title('Reconstructed Audio Signal (After Filtering)', 'Interpreter', 'latex', 'FontSize', 14);
xlabel('Sample Index', 'FontSize', 12);
ylabel('Amplitude', 'FontSize', 12);
xlim([0 samples_to_plot]);
ylim([-0.2 0.2]);
grid on;

figure;
hold on;
```

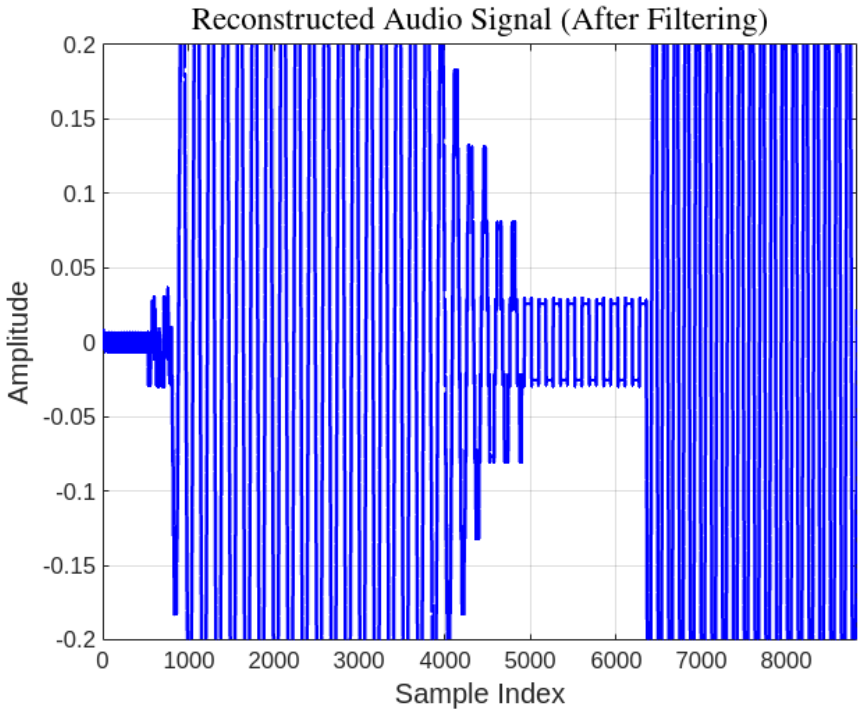
```
plot(audio_in(1:samples_to_plot), 'r', 'LineWidth', 1.5);
plot(x_reconstructed(1:samples_to_plot), 'b', 'LineWidth', 1.5);
hold off;
title('Overlay: Original vs Reconstructed (Zoomed)', 'Interpreter', 'latex', 'FontSize', 14);
xlabel('Sample Index', 'FontSize', 12);
ylabel('Amplitude', 'FontSize', 12);
legend({'Original Audio', 'Reconstructed Audio'}, 'FontSize', 12);
xlim([0 samples_to_plot]);
ylim([-0.2 0.2]);
grid on;
```

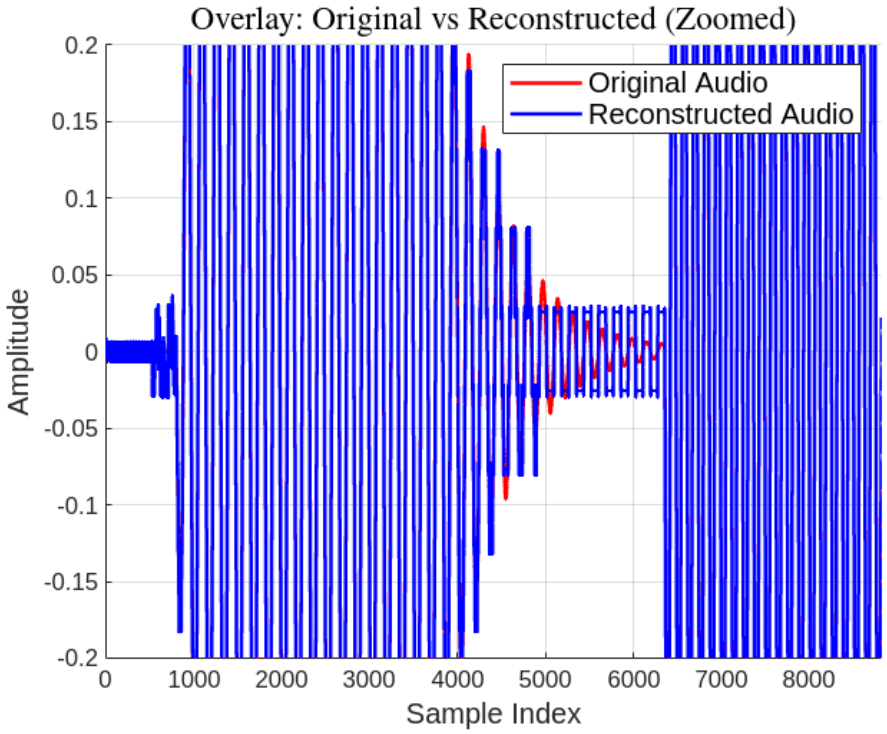
output:

```
Original Audio: 14473.19 KB
After Modulation (Before Filtering): 57892.75 KB
```




Sigma-Delta Modulated Signal (Before Filtering)
Amplitude
Sample Index
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
0
1000
2000
3000
4000
5000
6000
7000
8000


Reconstructed Audio Signal (After Filtering)
Amplitude
Sample Index
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
0
1000
2000
3000
4000
5000
6000
7000
8000



Observation:
One of the most obvious consequences of Sigma-Delta modulation is the appreciable data size growth. The original audio file runs about 14,493 KB initially, however after modulation the storage need leaps to 57,892 KB before filtering. Sigma-Delta modulation depends on oversampling to encode the signal using a high-frequency binary representation instead of simply storing exact amplitude values, so this significant rise results. Sigma-Delta modulation essentially codes changes over time, which needs a far greater sampling rate than encoding an audio source with a fixed amount of bits per sample (such as 16-bit PCM). This method has the benefit of distributing quantization noise over a large frequency range, which facilitates effective later stage filtering out. But this also produces a faster data flow, which increases file size prior to the ultimate reconstruction. excellent-end digital audio processing often shows this trade-off: bit-depth can be lowered using oversampling techniques yet excellent quality can be maintained.

Showing its changes in amplitude and frequency over time, the first graphic offers a zoom-in-view of the original audio signal. Representing the natural qualities of the original sound, the waveform seems smooth and continuous. The signal shows intervals of almost quiet, damping—gradual amplitude reduction—and areas of high-frequency oscillations. These characteristics, which define speech or music signals, show that the original audio had dynamic fluctuations in volume and pitch. Although the damping zone portrays a fading or resonating sound, physically the high-frequency oscillations match high-energy sound components such sharp transients or high-pitched tones. Quiet times point to pauses or low-intensity sound passages. In audio transmissions, where both high- and low-energy components add to total perception, this unpredictability is vital. The smooth character of the signal emphasizes the need of preserving precise amplitude resolution during the modulation process since sudden quantization can produce artifacts degrading sound quality.
The waveform takes on a rather distinct form following Sigma-Delta modulation of the initial input.

The modulated signal has step-like shifts with fast transitions between discrete levels rather than a smooth sinusoidal form. Sigma-Delta modulation stores the changes between consecutive samples rather than immediately amplitude values, so this behavior results. The modulator essentially tracks the variation between an expected version of the input signal and its actual form, then modulates its output in little discrete steps.

This graph shows a major finding: denser, more frequent step transitions in the modulated version translate from areas of high-frequency oscillations in the original signal. On the other hand, calmer parts of the signal feature substantially less transitions. This is so because Sigma-Delta modulation drives quantization noise into higher frequencies while preserving low-frequency content first priority. The high-density step changes match high-energy parts of the signal, when quick changes are required to follow the input waveform. The few transitions in the quiet areas point to low-energy components of the signal that call for little change.
This change in the frequency domain indicates that, using a low-pass filter, much of the quantization noise has been moved to higher frequencies, facilitating removal. One of the main benefits of Sigma-Delta modulation is this quality, sometimes referred to as noise shaping: it enables high-resolution audio representation even with a low-bit-depth quantizer. Still, at this point the signal still has high-frequency noise artifacts that distort the music, hence it is not yet fit for direct playing.

After a low-pass filter removes high-frequency noise brought by modulation, the third plot shows the audio signal. The reconstructed waveform is now far smoother and more precisely follows the original input than the step-like form of the modulated signal. This guarantees that Sigma-Delta modulation pushes noise into a frequency region where it may be readily eliminated while preserving the fundamental features of the audio stream.
By restoring the continuous character of the waveform, the filtering technique removes the sharp transitions shown in the modulated signal. This stage mathematically is equivalent to convolution with a low-pass filter, which reduces high-frequency components while preserving the original low-frequency information. Physically, this means that free from the quantization noise brought in during modulation, the rebuilt audio is now a faithful reproduction of the original sound.
This graph clearly shows that tiny discrepancies still exist in some parts, especially in locations where the original signal underwent fast fluctuations. During reconstruction, quantization effects and interpolation errors cause these little mistakes. Although they are usually undetectable in audio playback, they draw attention to the trade-offs in the modulation process—between bit-rate efficiency, signal resolution, and noise reduction.

To enable a direct comparison, the last plot overlays the rebuilt signal (blue) with the red original audio waveform. The near alignment of the two signals shows that Sigma-Delta modulation effectively preserves the fundamental information in the audio, therefore allowing accurate reconstruction. Small alterations, however, are seen in some areas, especially if the original signal displayed complicated fluctuations or fast amplitude changes.
Quantization noise, filtering approximations, and perhaps phase shifts brought by during reconstruction cause these little differences. Still, the general resemblance of the signals indicates how very successful the modulation and reconstruction techniques are in preserving aural fidelity. In practical applications, where digital-to- analog converters (DACs) have to guarantee that the rebuilt audio nearly resembles the original recording, this stage is absolutely important.
The outcomes of this work show the basic ideas behind effective data transmission, analog-to-digital conversion, and high-resolution digital audio processing. Because sigma-Delta modulation preserves signal integrity and lowers quantization errors, it is extensively applied in biomedical imaging, wireless communication systems, and audio signal processing.
Sigma-Delta DACs are found in professional audio equipment, music players, and cellphones in high-fidelity audio systems to guarantee accurate sound reproduction with lowest possible

distortion. Observed in this experiment, the increase in data size following modulation demonstrates how highly-quality audio conversion depends on oversampling to attain improved signal resolution.