Delta Modulation and Demodulation using Python

Objective:

- Understand and implement the process of Delta Modulation and Demodulation using Python.
- Visualize the encoded signal and reconstructed signal.
- Apply a Low-Pass Filter to improve the quality of the reconstructed signal.
- Compare the reconstructed signals with the original signal.

Introduction: A delta modulation (DM or Δ-modulation) is an analog-to-digital and digital-to-analog signal conversion technique used for transmission of voice information where quality is not of primary importance. DM is the simplest form of differential pulse-code modulation (DPCM) where the difference between successive samples are encoded into n-bit data streams. In delta modulation, the transmitted data are reduced to a 1-bit data stream. Its main features are:

- The analog signal *x(t)* is approximated with a series of segments.
- Each segment of the approximated signal is compared to the preceding bits and the successive bits are determined by this comparison.
- Only the change of information is sent, that is, only an increase or decrease of the signal amplitude from the previous sample is sent whereas a no-change condition causes the modulated signal to remain at the same 0 or 1 state of the previous sample.

To achieve high signal-to-noise ratio, delta modulation must use oversampling techniques, that is, the analog signal is sampled at a rate several times higher than the Nyquist rate.

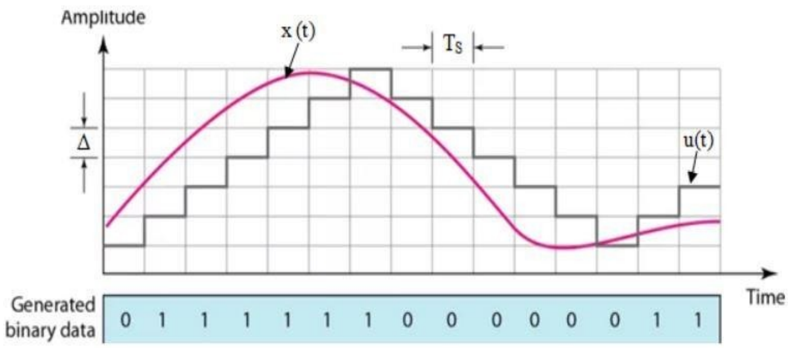


Figure 1: Delta modulation

x(t) represents the analog signal and $x_q(t)$ represents the staircase approximation. Following discrete relations explain the construcion of the staircase waveform which forms the basis of delta modulation.

$$e(nT_s) = x_q(nT_s) - x_q(nT_s - T_s)$$

$$e_q(nT_s) = \delta\ sgn[e_q(nT_s)]$$

$$x_q(nT_s) = x_q(nT_s - T_s) + e_q(nT_s)$$

Where T_s is the sampling instant, $e(nT_s)$ is the error signal and $e_q(nT_s)$ is the quantized version of error signal.

Block Diagram:

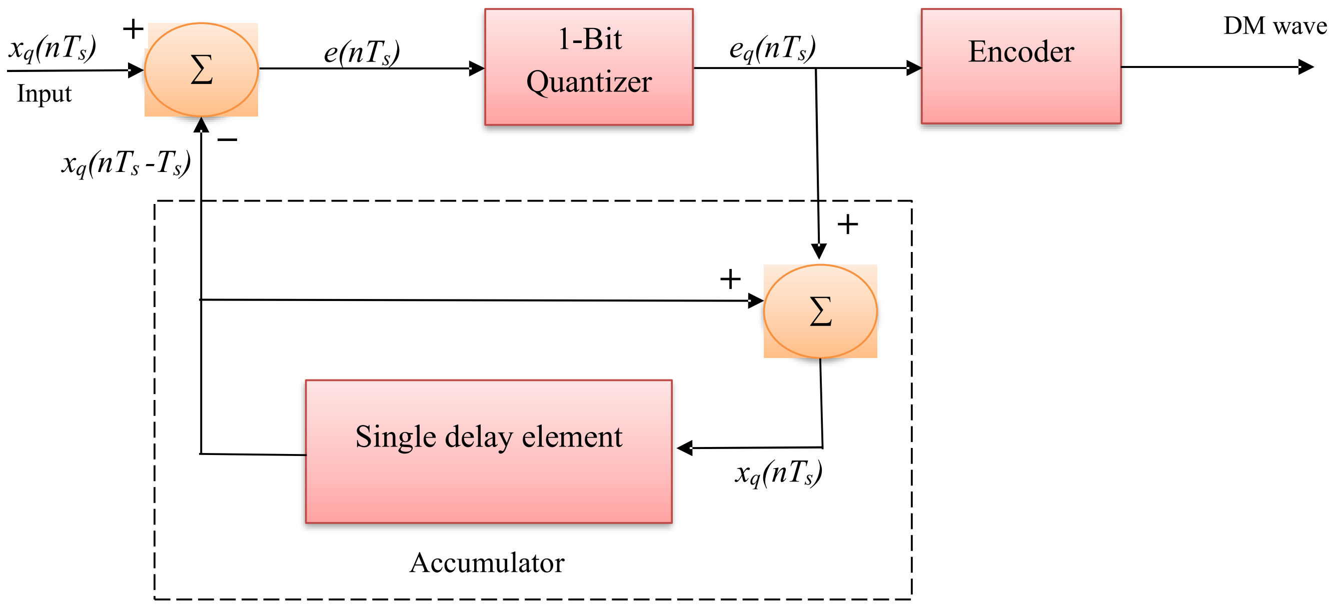


Figure 2: Block diagram of the DM transmitter

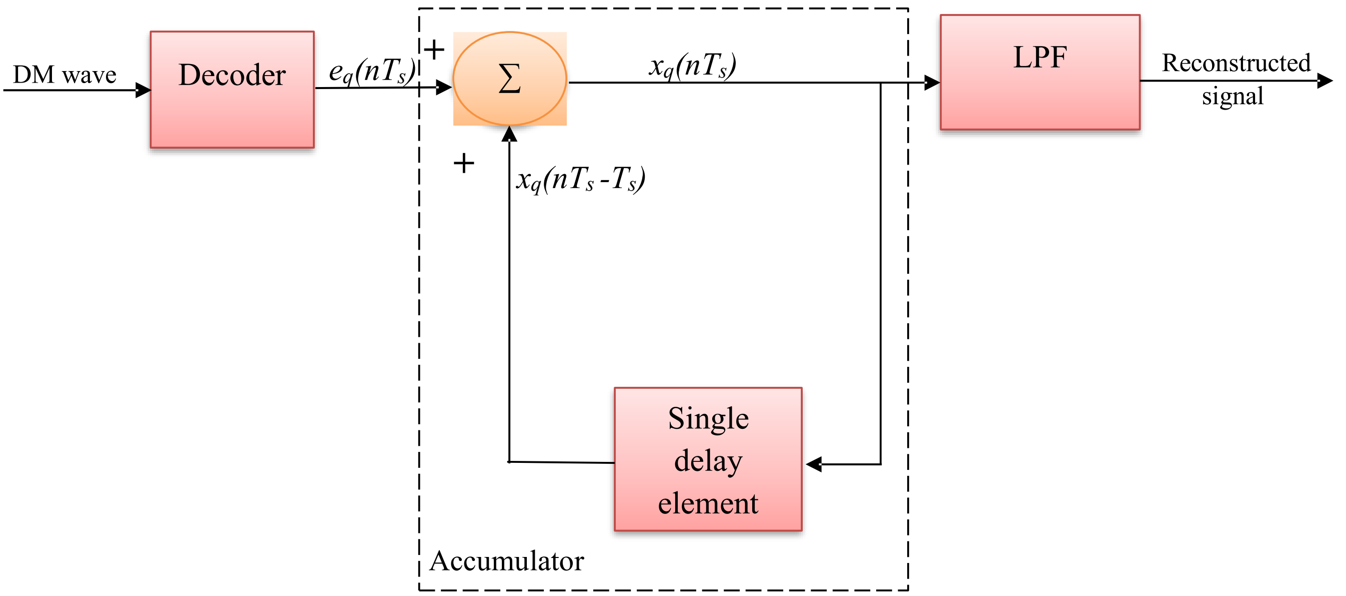


Figure 3: Block diagram of the DM receiver

Implementation Using Python

Steps

1. **Generate the Input Signal:**
   - A sine wave is used as the analog signal for demonstration.
2. **Delta Modulation:**
   - The difference between the current sample and the quantized signal is compared.
   - If the sample is greater, a `1` is appended to the binary sequence; otherwise, a `0`.
3. **Delta Demodulation:**
   - The binary sequence is interpreted to reconstruct the signal using the fixed step size.
   - A cumulative sum of the step size (positive or negative) generates the reconstructed signal.
4. **Apply Low-Pass Filter:**
   - A Butterworth filter is designed using `scipy` to smooth the reconstructed signal.
5. **Visualization:**
   - Plot the original signal, delta-modulated signal, reconstructed signal (before and after filtering), and overlay the original signal for comparison.

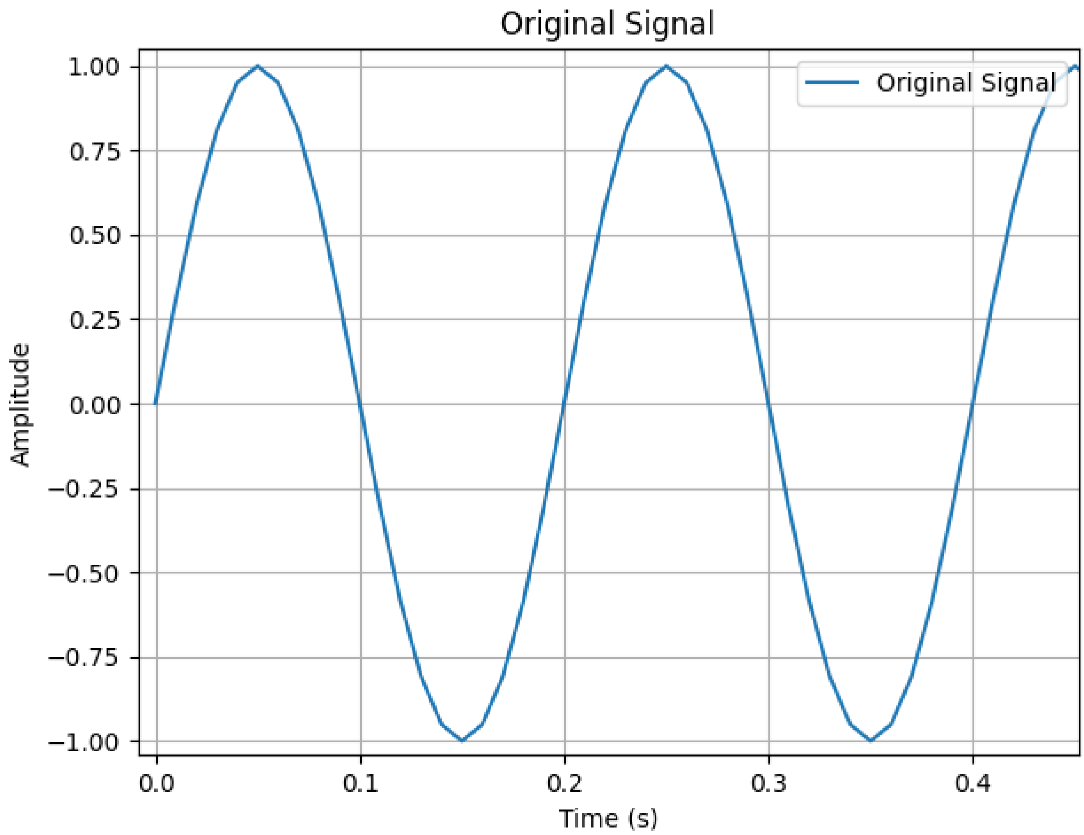
Python Code

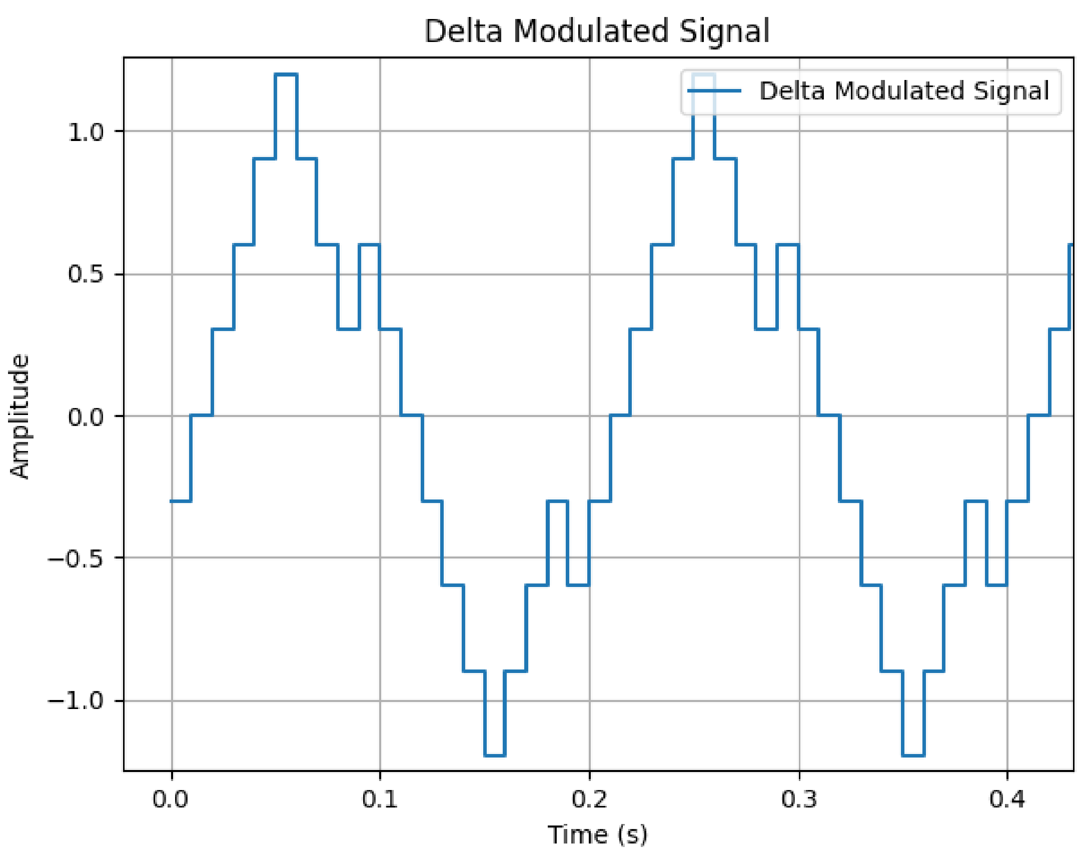
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import butter, filtfilt

# Parameters
sampling_rate = 100  # Sampling rate (Hz)
duration = 1          # Duration of the signal (seconds)
step_size = 0.3       # Step size for Delta Modulation

# Generate a sine wave signal
time = np.arange(0, duration, 1 / sampling_rate)
frequency = 5  # Frequency of the sine wave (Hz)
input_signal = np.sin(2 * np.pi * frequency * time)

# Delta Modulation
delta_modulated_signal = []
quantized_signal = [0]  # Initialize quantized signal with zero

```

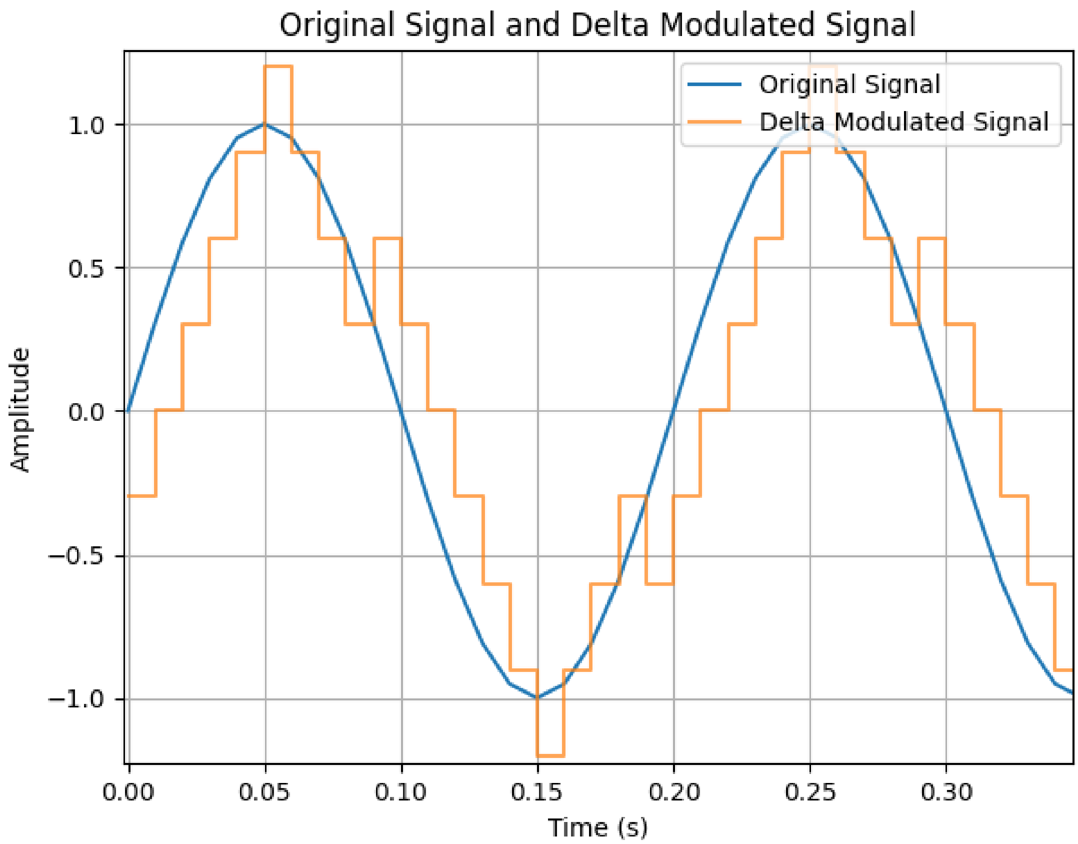
```python
for sample in input_signal:
    # Determine the step (1 or -1)
    if sample > quantized_signal[-1]:
        delta_modulated_signal.append(1)
        new_value = quantized_signal[-1] + step_size
    else:
        delta_modulated_signal.append(0)
        new_value = quantized_signal[-1] - step_size
    quantized_signal.append(new_value)

# Remove the initial zero from quantized_signal for plotting
quantized_signal = quantized_signal[1:]

# Delta Demodulation
reconstructed_signal = [0]  # Initialize the reconstructed
signal
for bit in delta_modulated_signal:
    if bit == 1:
        new_value = reconstructed_signal[-1] + step_size
    else:
        new_value = reconstructed_signal[-1] - step_size
    reconstructed_signal.append(new_value)

# Remove the initial zero from reconstructed_signal for
plotting
reconstructed_signal = reconstructed_signal[1:]

# Apply Low-Pass Filter to the Reconstructed Signal
def low_pass_filter(data, cutoff_freq, fs, order=4):
    nyquist = fs / 2.0
    normal_cutoff = cutoff_freq / nyquist
    b, a = butter(order, normal_cutoff, btype='low',
analog=False)
    filtered_signal = filtfilt(b, a, data)
    return filtered_signal

```

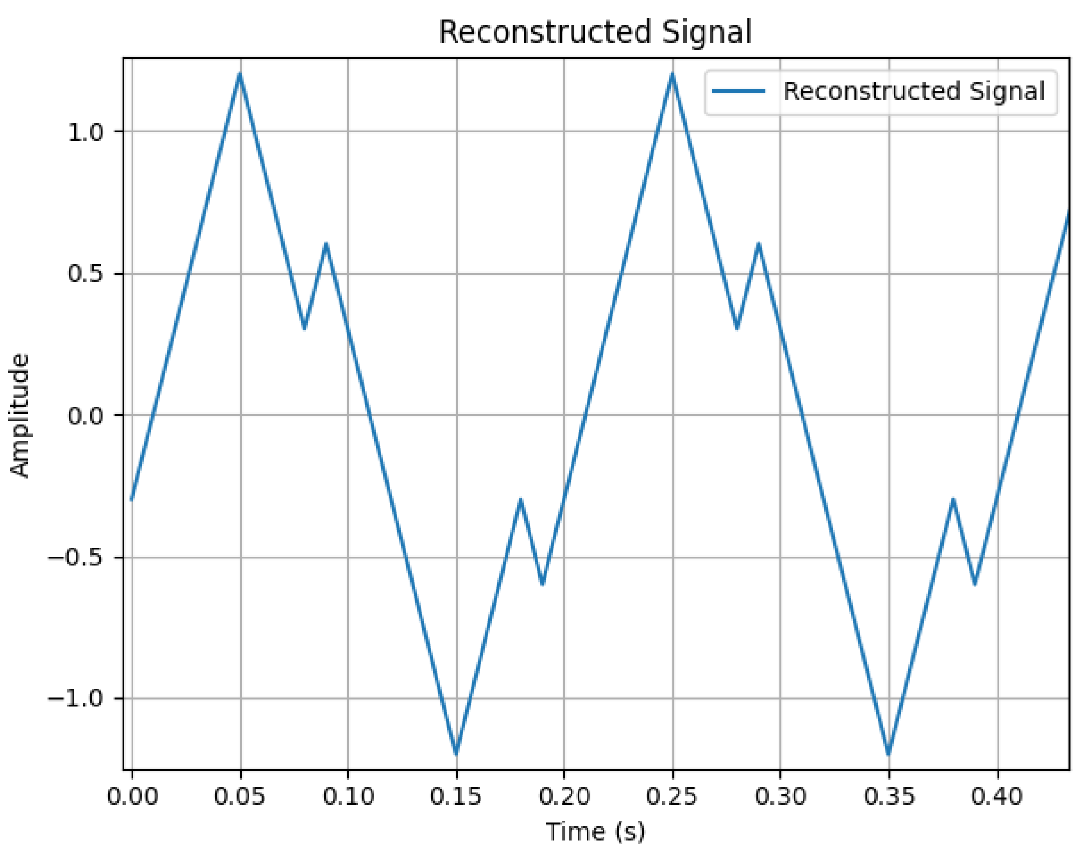
```
cutoff_frequency = 10  # Cutoff frequency for the LPF (Hz)
reconstructed_signal_lpf =
low_pass_filter(reconstructed_signal, cutoff_frequency,
sampling_rate)

# Plotting
fig, axs = plt.subplots(2, 3, figsize=(16, 10))

# Plot 1: Original Signal
axs[0, 0].plot(time, input_signal, label="Original Signal")
axs[0, 0].set_title("Original Signal")
axs[0, 0].set_xlabel("Time (s)")
axs[0, 0].set_ylabel("Amplitude")
axs[0, 0].grid()
axs[0, 0].legend(loc="upper right")

# Plot 2: Delta Modulated Signal
axs[0, 1].step(time, quantized_signal, label="Delta Modulated
Signal", where="post")
axs[0, 1].set_title("Delta Modulated Signal")
axs[0, 1].set_xlabel("Time (s)")
axs[0, 1].set_ylabel("Amplitude")
axs[0, 1].grid()
axs[0, 1].legend(loc="upper right")

# Plot 3: Original Signal on Top of Delta Modulated Signal
axs[0, 2].plot(time, input_signal, label="Original Signal")
axs[0, 2].step(time, quantized_signal, label="Delta Modulated
Signal", where="post", alpha=0.7)
axs[0, 2].set_title("Original Signal and Delta Modulated
Signal")
axs[0, 2].set_xlabel("Time (s)")
axs[0, 2].set_ylabel("Amplitude")
axs[0, 2].grid()
axs[0, 2].legend(loc="upper right")
```

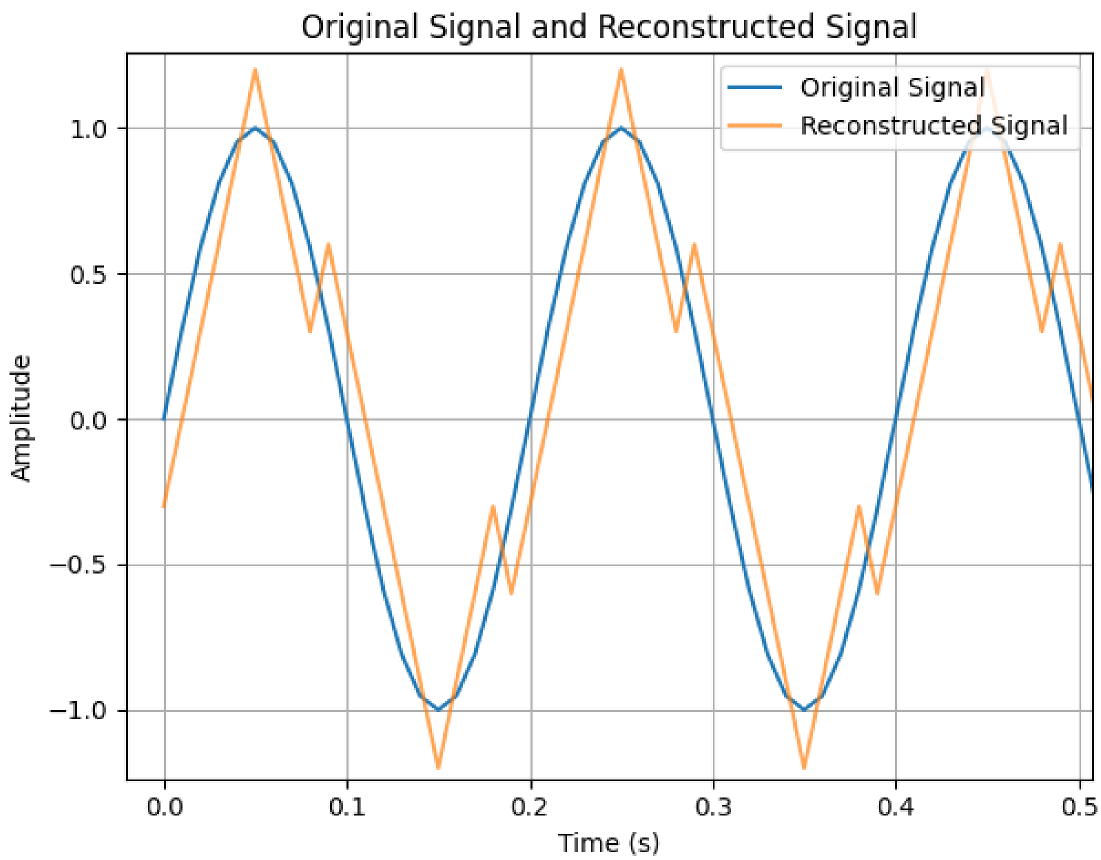
```python
# Plot 4: Reconstructed Signal
axs[1, 0].plot(time, reconstructed_signal, label="Reconstructed Signal")
axs[1, 0].set_title("Reconstructed Signal")
axs[1, 0].set_xlabel("Time (s)")
axs[1, 0].set_ylabel("Amplitude")
axs[1, 0].grid()
axs[1, 0].legend(loc="upper right")

# Plot 5: Reconstructed Signal on Top of Original Signal
axs[1, 1].plot(time, input_signal, label="Original Signal")
axs[1, 1].plot(time, reconstructed_signal, label="Reconstructed Signal", alpha=0.7)
axs[1, 1].set_title("Original Signal and Reconstructed Signal")
axs[1, 1].set_xlabel("Time (s)")
axs[1, 1].set_ylabel("Amplitude")
axs[1, 1].grid()
axs[1, 1].legend(loc="upper right")

# Plot 6: Reconstructed Signal after LPF on Top of Original Signal
axs[1, 2].plot(time, input_signal, label="Original Signal")
axs[1, 2].plot(time, reconstructed_signal_lpf, label="Output", alpha=0.7)
axs[1, 2].set_title("Original Signal and Reconstructed Signal (After LPF)")
axs[1, 2].set_xlabel("Time (s)")
axs[1, 2].set_ylabel("Amplitude")
axs[1, 2].grid()
axs[1, 2].legend(loc="upper right")

# Adjust layout
plt.tight_layout()
plt.show()
```

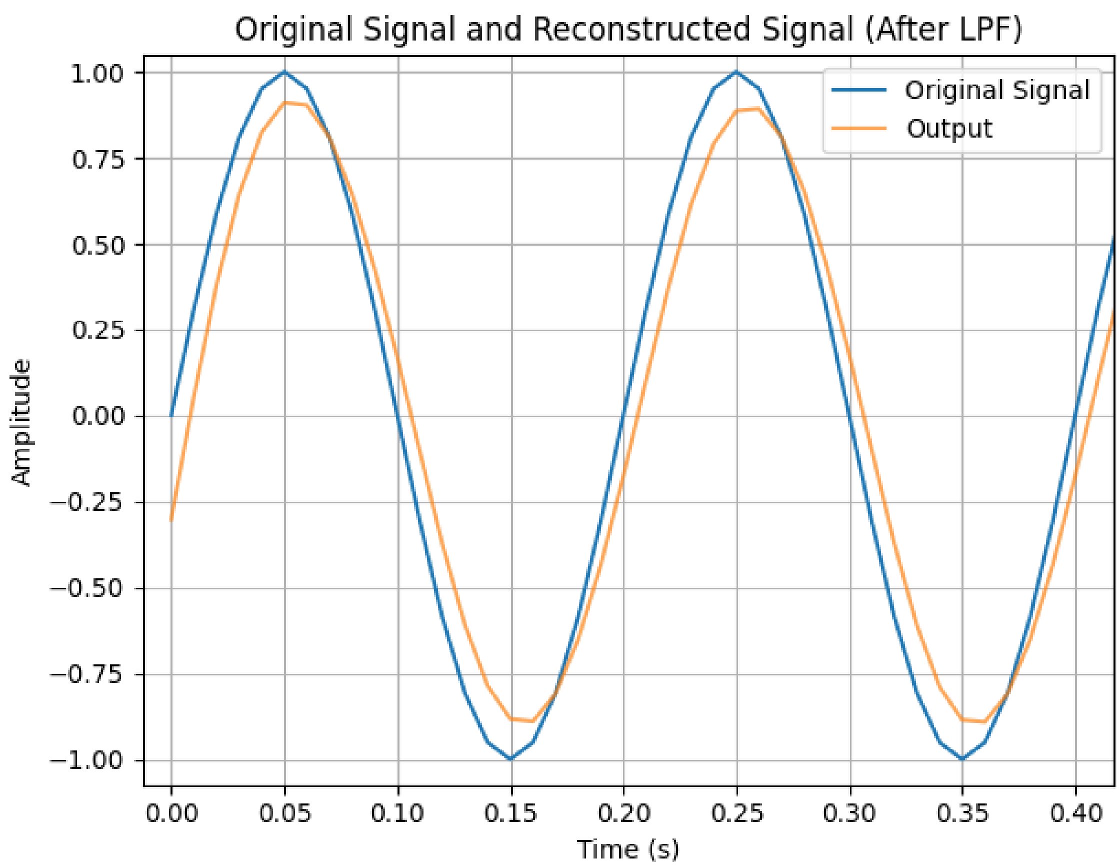

Original Signal
Original Signal
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Amplitude
0.0
0.1
0.2
0.3
0.4
Time (s)


Delta Modulated Signal
Delta Modulated Signal
1.0
0.5
0.0
−0.5
−1.0
Amplitude
0.0
0.1
0.2
0.3
0.4
Time (s)


Original Signal and Delta Modulated Signal
Original Signal
Delta Modulated Signal
Amplitude
Time (s)
1.0
0.5
0.0
−0.5
−1.0
0.00
0.05
0.10
0.15
0.20
0.25
0.30


Reconstructed Signal
Reconstructed Signal
Amplitude
Time (s)
1.0
0.5
0.0
−0.5
−1.0
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40


Original Signal and Reconstructed Signal
Original Signal
Reconstructed Signal
Amplitude
Time (s)
1.0
0.5
0.0
−0.5
−1.0
0.0
0.1
0.2
0.3
0.4
0.5


Original Signal and Reconstructed Signal (After LPF)
Original Signal
Output
Amplitude
Time (s)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40

Results

1. **Original Signal:** A sine wave with a frequency of 5 Hz is generated as the input.
2. **Delta Modulated Signal:** The staircase approximation of the input signal shows quantization.
3. **Reconstructed Signal:** The staircase output is the cumulative sum of step sizes from the binary sequence.
4. **Reconstructed Signal after LPF:** The Low-Pass Filter smooths the staircase into a continuous signal resembling the original.

Discussion

The implementation demonstrates the core principles of Delta Modulation and Demodulation.

- The **step size** plays a crucial role in determining the fidelity of the modulation and reconstruction process. Too large a step size increases granular noise, while a small step size may cause slope overload.
- The **Low-Pass Filter** effectively mitigates the staircase artifact, providing a closer match to the original signal.

Conclusion

Delta Modulation and Demodulation were successfully implemented and visualized in Python. By incorporating a Low-Pass Filter, the reconstructed signal closely approximates the original, highlighting the utility of post-processing in communication systems.