Experiment No. : 02

Experiment Name: Implementation of Pulse Code Modulation and Delta Modulation.

Objectives: The objective of this lab is to understand and implement Pulse Code Modulation (PCM) for converting an analogue signal into a digital signal, and to analyse the effects of sampling, quantization, and encoding on the signal quality and to understand the principles of Delta Modulation (DM) and its application in digital communication systems. The experiment aims to implement DM, analyse its performance in encoding and decoding signals, and evaluate the effects of quantization noise and signal-to-noise ratio (SNR).

Theory:

Delta Modulation (DM) is a method of analogue-to-digital signal conversion that uses the difference between successive samples of the signal to encode information. It is a simplified form of Differential Pulse Code Modulation (DPCM) and is used for voice transmission and other applications with limited bandwidth.

Delta Modulation Process:

Sampling: The analogue signal is sampled at regular intervals.
Quantization: The difference between the current and previous samples is quantized into a binary value.
Encoding: The quantized difference is encoded as a bitstream.
Integration: The received bitstream is integrated to reconstruct the signal.

Pulse Code Modulation (PCM) is a method used to digitally represent analogue signals. It involves three key steps:

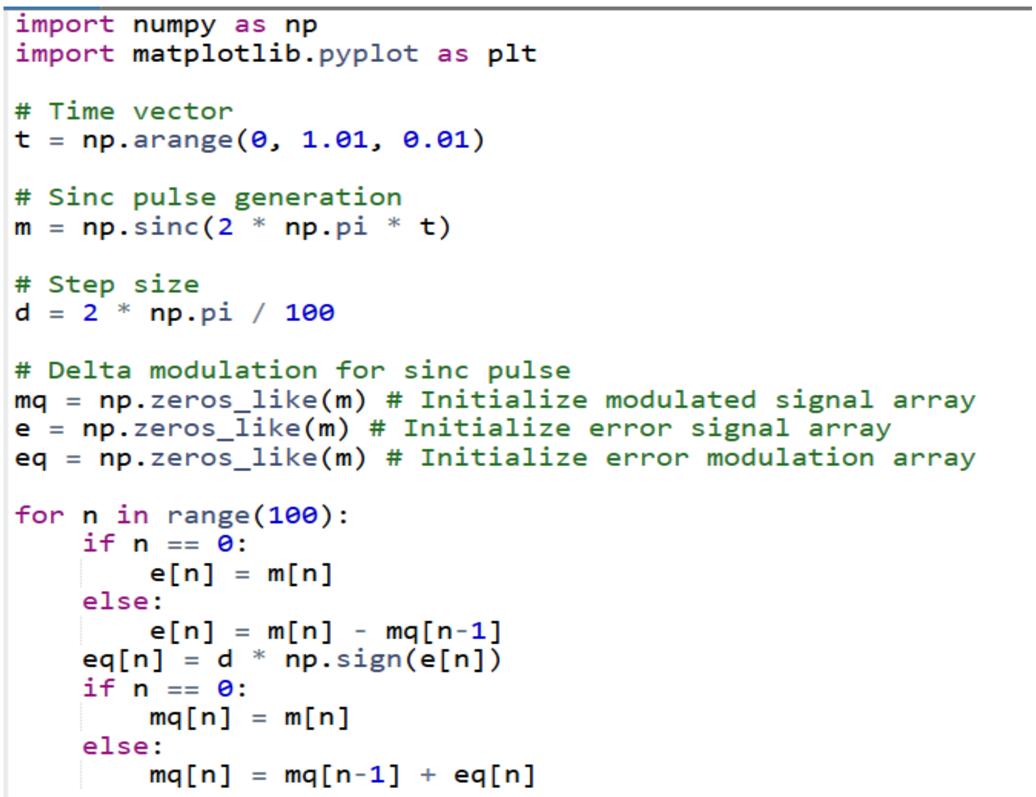
Sampling: The analogue signal is sampled at regular intervals. According to the Nyquist-Shannon sampling theorem, the sampling rate must be at least twice the highest frequency present in the signal to avoid aliasing.

Quantization: Each sampled value is approximated to the nearest value within a finite set of discrete levels. This process introduces quantization error.

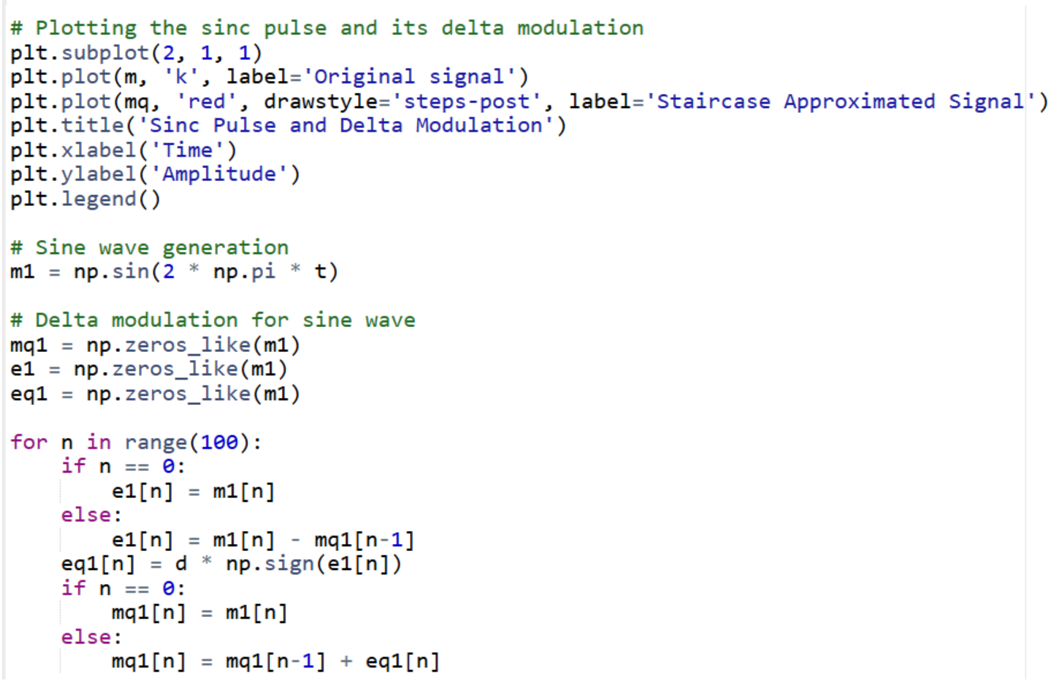
Encoding: The quantized values are converted into a binary format for digital representation. This binary data can be transmitted or stored efficiently.

Implemention Step:

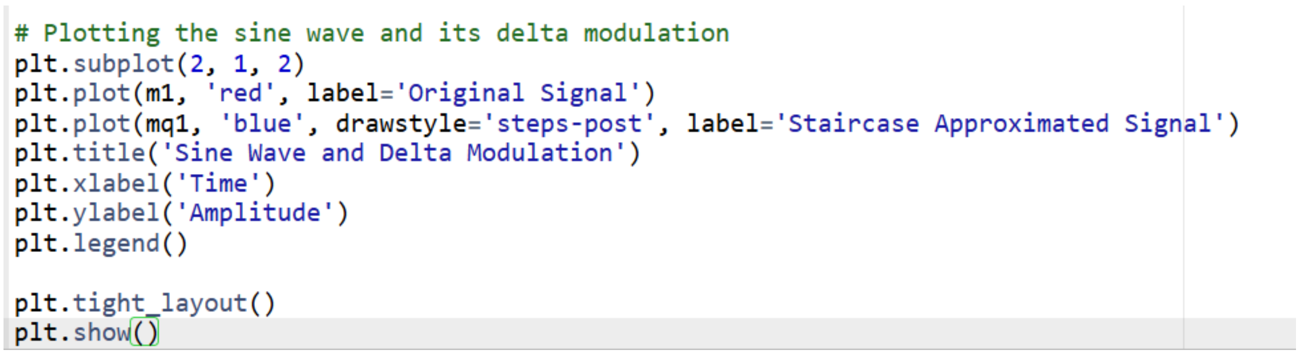
Delta Modulation:



```python
import numpy as np
import matplotlib.pyplot as plt

# Time vector
t = np.arange(0, 1.01, 0.01)

# Sinc pulse generation
m = np.sinc(2 * np.pi * t)

# Step size
d = 2 * np.pi / 100

# Delta modulation for sinc pulse
mq = np.zeros_like(m) # Initialize modulated signal array
e = np.zeros_like(m) # Initialize error signal array
eq = np.zeros_like(m) # Initialize error modulation array

for n in range(100):
    if n == 0:
        e[n] = m[n]
    else:
        e[n] = m[n] - mq[n-1]
    eq[n] = d * np.sign(e[n])
    if n == 0:
        mq[n] = m[n]
    else:
        mq[n] = mq[n-1] + eq[n]
```



```python
# Plotting the sinc pulse and its delta modulation
plt.subplot(2, 1, 1)
plt.plot(m, 'k', label='Original signal')
plt.plot(mq, 'red', drawstyle='steps-post', label='Staircase Approximated Signal')
plt.title('Sinc Pulse and Delta Modulation')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.legend()

# Sine wave generation
m1 = np.sin(2 * np.pi * t)

# Delta modulation for sine wave
mq1 = np.zeros_like(m1)
e1 = np.zeros_like(m1)
eq1 = np.zeros_like(m1)

for n in range(100):
    if n == 0:
        e1[n] = m1[n]
    else:
        e1[n] = m1[n] - mq1[n-1]
    eq1[n] = d * np.sign(e1[n])
    if n == 0:
        mq1[n] = m1[n]
    else:
        mq1[n] = mq1[n-1] + eq1[n]
```

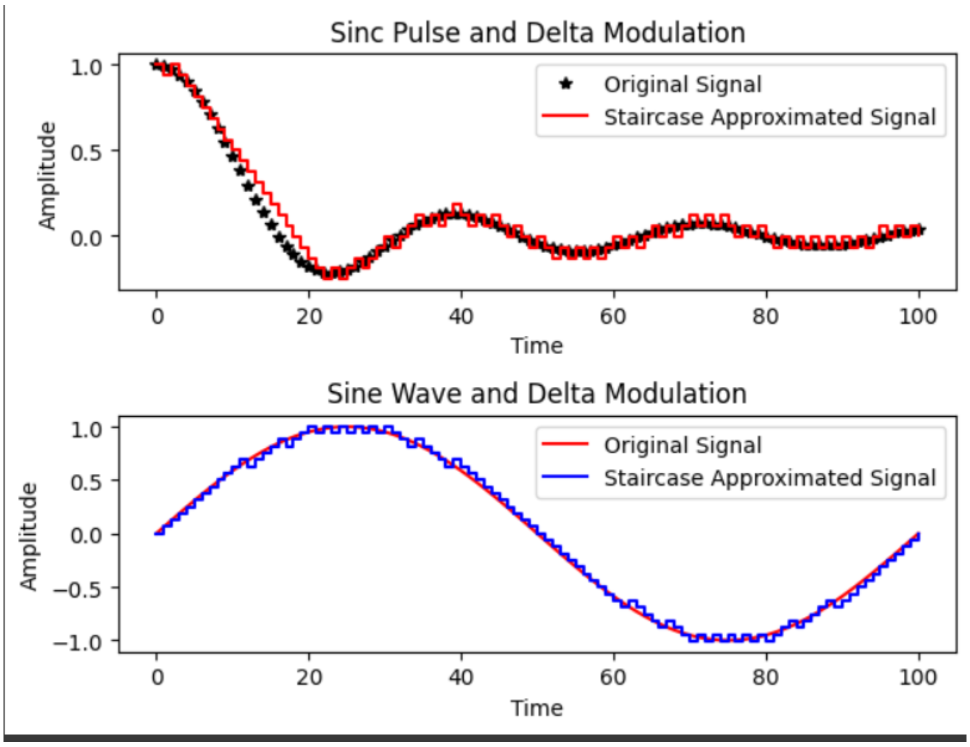


```
# Plotting the sine wave and its delta modulation
plt.subplot(2, 1, 2)
plt.plot(m1, 'red', label='Original Signal')
plt.plot(mq1, 'blue', drawstyle='steps-post', label='Staircase Approximated Signal')
plt.title('Sine Wave and Delta Modulation')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.legend()

plt.tight_layout()
plt.show()
```

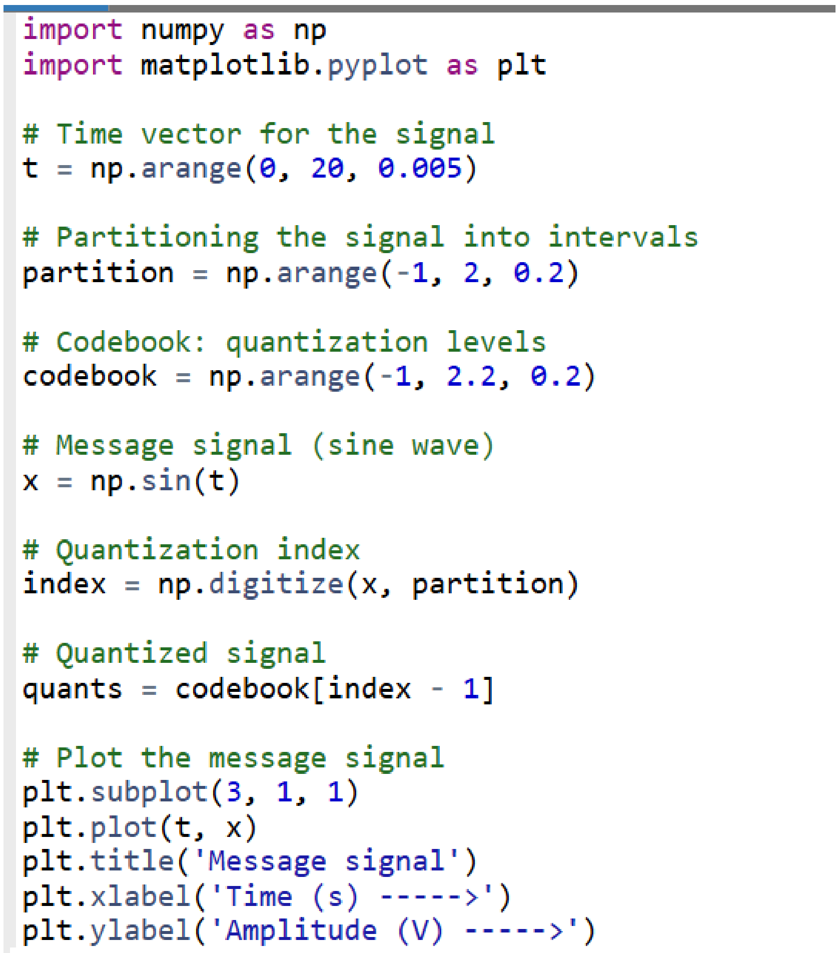
Explanation:

1. **Sine Pulse Generation (m):** A sine pulse is generated *using np.sinc(2 * np.pi * t)*, which is a scaled sine function.
2. **Delta Modulation for Sine Pulse (mq):** A loop runs to calculate the error (e) between the current signal and the modulated signal (mq[n-1]), then applies delta modulation by updating mq.
3. **Sine Wave Generation (m1):** A sine wave is generated using np.sin(2 * np.pi * t).
4. **Delta Modulation for Sine Wave (mq1):** Similar to the sinc pulse modulation, the sine wave is modulated using the delta modulation algorithm.
5. **Plotting:** Two subplots are created: one for the sine pulse and its delta modulation, and one for the sine wave and its delta modulation.

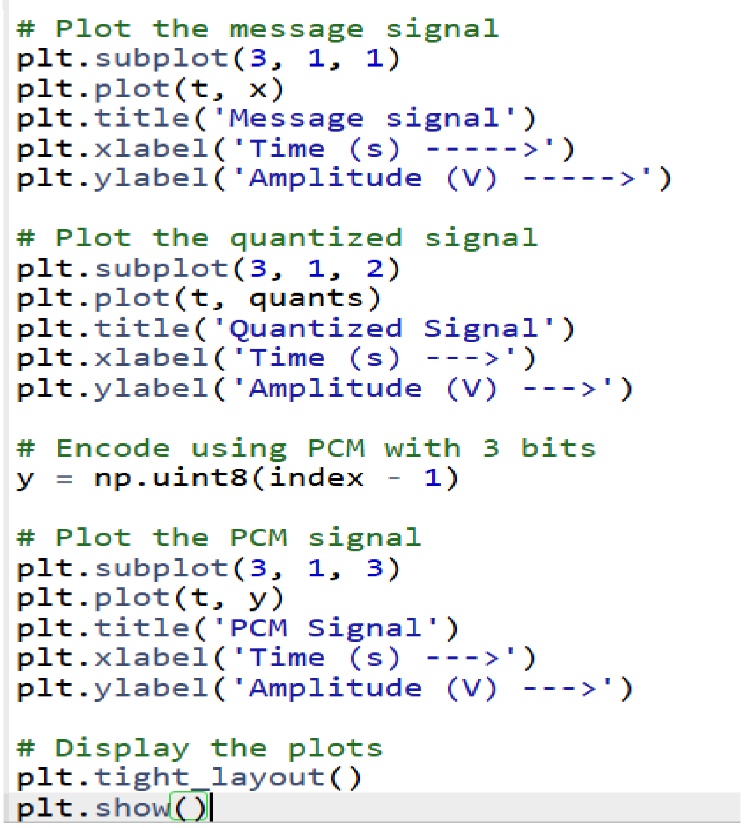
Output:



Pulse Code Modulation(PCM):



```python
import numpy as np
import matplotlib.pyplot as plt

# Time vector for the signal
t = np.arange(0, 20, 0.005)

# Partitioning the signal into intervals
partition = np.arange(-1, 2, 0.2)

# Codebook: quantization levels
codebook = np.arange(-1, 2.2, 0.2)

# Message signal (sine wave)
x = np.sin(t)

# Quantization index
index = np.digitize(x, partition)

# Quantized signal
quants = codebook[index - 1]

# Plot the message signal
plt.subplot(3, 1, 1)
plt.plot(t, x)
plt.title('Message signal')
plt.xlabel('Time (s) ----->')
plt.ylabel('Amplitude (V) ----->')
```



```python
# Plot the message signal
plt.subplot(3, 1, 1)
plt.plot(t, x)
plt.title('Message signal')
plt.xlabel('Time (s) ----->')
plt.ylabel('Amplitude (V) ----->')

# Plot the quantized signal
plt.subplot(3, 1, 2)
plt.plot(t, quants)
plt.title('Quantized Signal')
plt.xlabel('Time (s) --->')
plt.ylabel('Amplitude (V) --->')

# Encode using PCM with 3 bits
y = np.uint8(index - 1)

# Plot the PCM signal
plt.subplot(3, 1, 3)
plt.plot(t, y)
plt.title('PCM Signal')
plt.xlabel('Time (s) --->')
plt.ylabel('Amplitude (V) --->')

# Display the plots
plt.tight_layout()
plt.show()
```

Explanation :

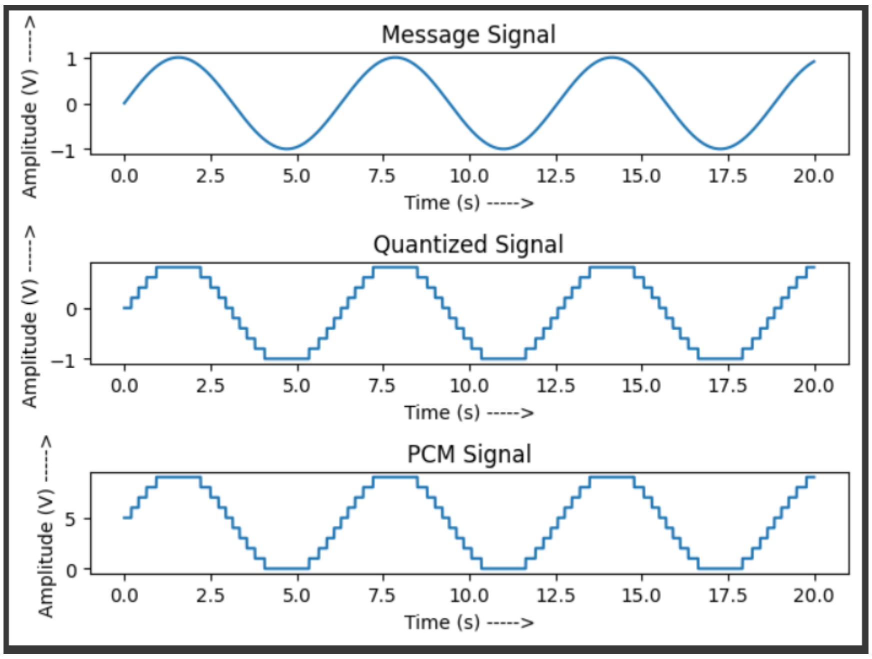
1. **Time Vector (t)**: The t array is created with values starting from 0 to 20, with a step of 0.005. This is the time axis for plotting.
2. **Partitioning the Signal**: The partition array defines the intervals that the signal values will be divided into for quantization. Here, the partition levels range from -1 to 2, with a step of 0.2.
3. **Codebook**: The codebook contains the quantization levels, i.e., the discrete values the continuous signal x will be mapped to after quantization. This also ranges from -1 to 2.2 with a step of 0.2.
4. **Message Signal (x)**: A sine wave (np.sin(t)) is used as the message signal. This is the signal that will undergo quantization.
5. **Quantization (index)**: np.digitize(x, partition) is used to find which partition each sample of the message signal x belongs to. This returns the index of the partition where each sample is located.
6. **Quantized Signal (quants)**: The quants array stores the quantized signal, which is created by mapping the index values to the codebook.
7. **Pulse Code Modulation (PCM)**: The PCM signal is created by converting the quantization indices into integers using np.uint8(index - 1). The resulting y array represents the digitalized form of the quantized signal.
8. **Plotting**:

   Message Signal: Plots the original continuous signal (x).

   Quantized Signal: Plots the quantized version of the signal (quants).

   PCM Signal: Plots the final PCM signal (y), which is the digitally encoded version of the quantized signal.
9. **plt.tight_layout()**: This ensures the subplots don't overlap, making the visual display of the graphs neat.

Output:



Discussion:

Quantization Noise: The step size (δ) plays a critical role in Delta Modulation. A smaller step size reduces quantization noise but may increase the risk of slope overload distortion. Conversely, a larger step size reduces slope overload but increases granular noise.

Slope Overload Distortion: Observed when the input signal changes rapidly, causing the modulator to fail in tracking the signal accurately. This is more prominent for high-frequency signals or signals with rapid amplitude changes.

Signal-to-Noise Ratio (SNR): The performance of Delta Modulation can be quantified by the SNR. Adjusting the step size can optimize the SNR for different signal characteristics.

Practical Applications: Delta Modulation is used in voice communication systems, such as digital telephony, where simplicity and bandwidth efficiency are critical. Understanding its limitations helps in designing systems that mitigate its drawbacks.

Pulse Code Modulation is a foundational technique in digital communication, allowing the conversion of analogue signals into a digital form for efficient transmission and processing. Through laboratory experiments, the principles of sampling, quantization, and encoding can be effectively demonstrated, providing a deeper understanding of how PCM underpins modern communication systems. The successful implementation and analysis of PCM in the lab highlight its importance and versatility in various real-world applications

In summary, the lab exercise provides practical insights into the PCM and delta modulation process, illustrating its critical role in modern digital communication systems.