Cairo University
Faculty of Engineering

Department of Computer Engineering

ELC 325B – Spring 2023

Digital Communications

Assignment #1

Quantization

Submitted to

Dr. Mai

Dr.Hala

Eng.Mohamed Khaled

Submitted by

Name	Sec	BN
Sara Bisheer Fekry	1	18
Menna Mohamed Abdelbaset	2	26

Contents

Figures

Part 1: Uniform Quantizer

```
def UniformQuantizer(in_val, n_bits, xmax, m):
    # Calculate the step size (quantization interval width)
    L = 2**n_bits
    delta = 2 * xmax / L

    # Quantize the input samples
    q_ind = ((in_val - ((m) * (delta / 2) - xmax)) / delta).astype(int)
    return q_ind
```

Figure 1 Fig

Comment:
The UniformQuantizer function performs uniform quantization on input samples.

1. It calculates the quantization interval width delta based on the number of bits:
   delta = 2 * xmax / (2 ** n_bits)
2. It quantizes the input samples using the formula:

q_ind = ((in_val - ((m) * (delta / 2) - xmax)) / delta).astype(int)

Part 2: Uniform Dequantizer

```
def UniformDequantizer(q_ind, n_bits, xmax, m):
    # Calculate the step size (quantization interval width)
    L = 2**n_bits
    delta = 2 * xmax / L

    # Reconstruct the de-quantized value
    deq_val = ((q_ind) * delta) + ((m+1) * (delta / 2) - xmax);

    return deq_val
```

Figure 2 Fig

Comment:
The `UniformDequantizer` function reconstructs de-quantized values from quantized indices.

1. It calculates the quantization interval width `delta` based on the number of bits: delta = 2 * xmax / (2 ** n_bits)
2. It reconstructs the de-quantized values using the formula:
   deq_val = ((q_ind) * delta) + ((m+1) * (delta / 2) - xmax)

so it complements the UniformQuantizer function by converting quantized indices back into approximate continuous values.

Part 3: test input on a random input signal

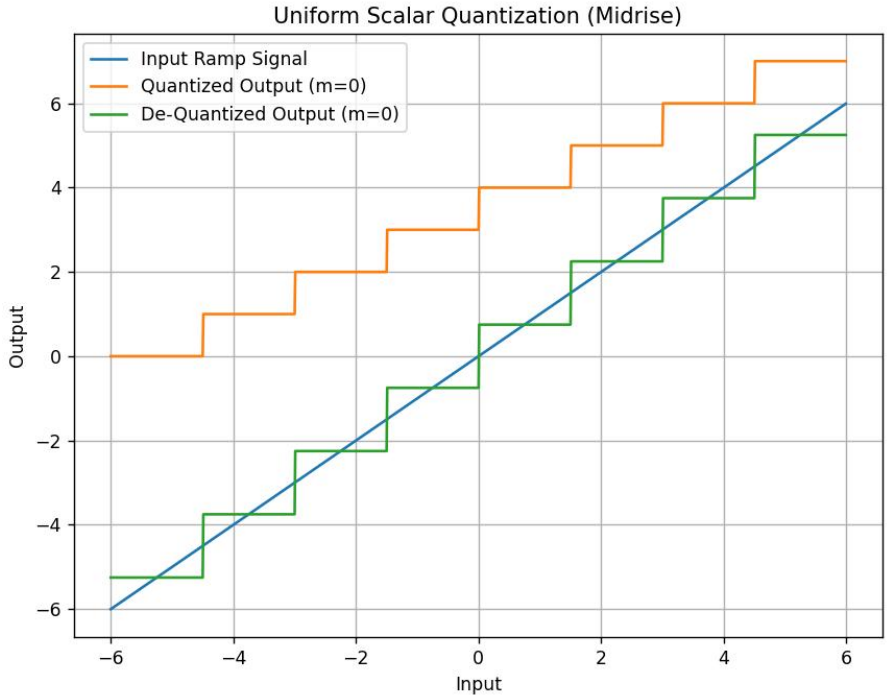

Figure 3 Fig

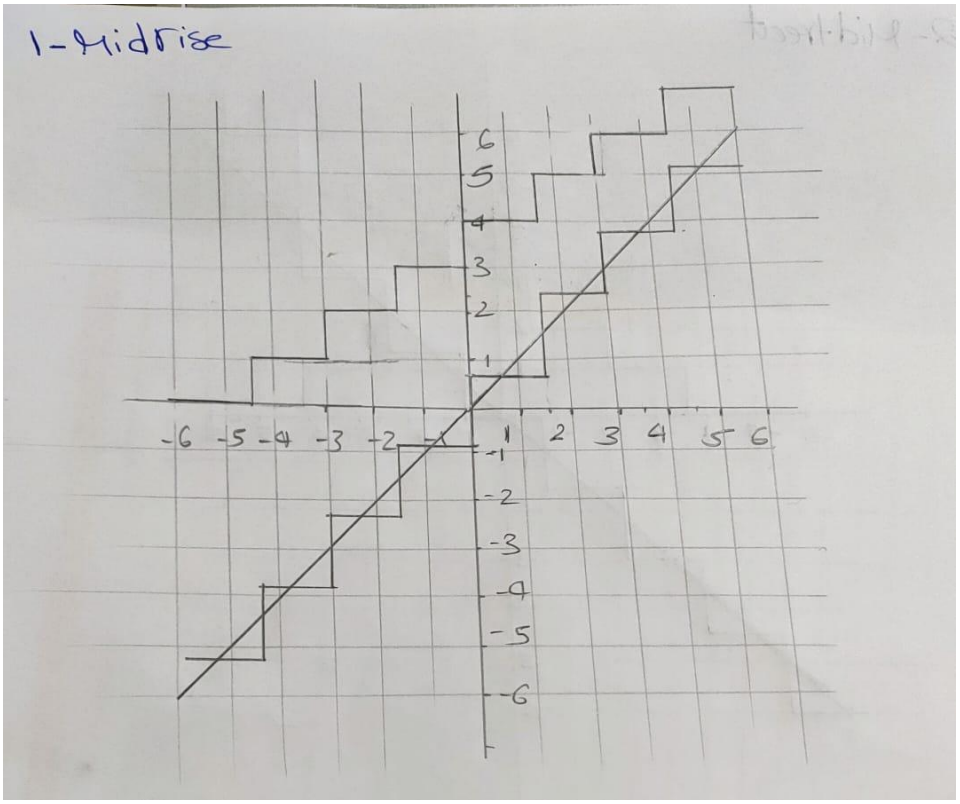

Figure 4 Fig

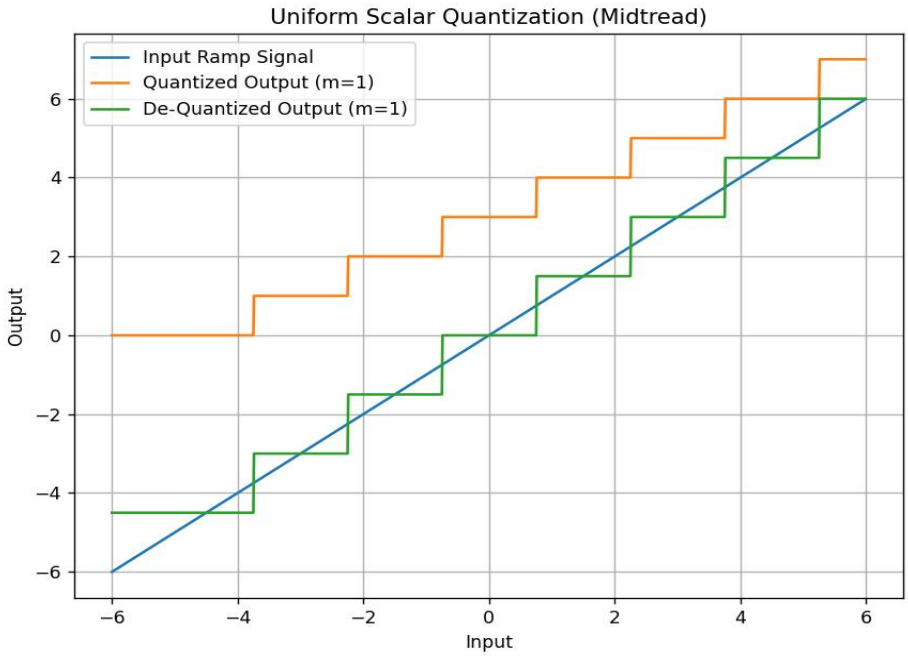

Figure 5 Fig

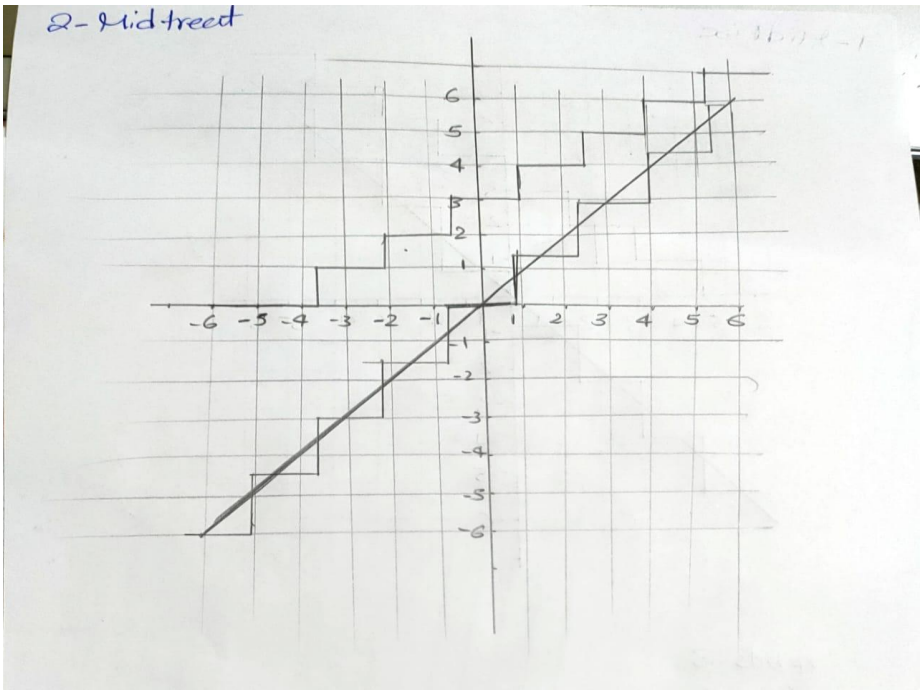

Figure 6 Fig

Comment:

In both preceding figures, the quantized and dequantized signals are displayed alongside the original signal to visualize the quantization error.

Part 4:Testing input on random input signal

```python
# Generate random input signal (10,000 samples)
np.random.seed(0)  # For reproducibility
input_signal = np.random.uniform(low=-5, high=5, size=10000)

# Parameters
xmax = 5
m = 0
n_bits_range = range(2, 9)
# Initialize arrays for SNR and quantization error
snr_values = []
theoretical_snr =  [ ]
# Calculate SNR and quantization error for each n_bits
for n_bits in n_bits_range:
    # Quantize the input signal
    q_ind = UniformQuantizer(input_signal, n_bits, xmax, m)
    # De-quantize the quantized signal
    deq_val = UniformDequantizer(q_ind, n_bits, xmax, m)
    # Calculate quantization error
    quant_error = input_signal - deq_val
    # Calculate SNR
    snr = np.mean(input_signal**2) / np.mean(quant_error**2)
    snr_values.append(snr)  # Convert to dB

    input_power = np.mean(input_signal**2)
    theoretical_snr.append((3 * (2 ** n_bits) ** 2 * input_power) /xmax**2)
# Plot SNR vs n_bits
plot_SNR(n_bits_range, [10 * np.log10(snr_values)],[10 * np.log10(theoretical_snr)],
    ['SNR'],['Theoretical SNR'], 'Number of Bits (n_bits)', 'SNR (dB)', 'Simulation SNR vs. Number of Bits (Uniform Input)')
plot_SNR(n_bits_range, [snr_values],[theoretical_snr],
    ['SNR'],['Theoretical SNR'], 'Number of Bits (n_bits)', 'SNR (linear)', 'Simulation SNR vs. Number of Bits (Uniform Input)')
```

Figure 7 Fig

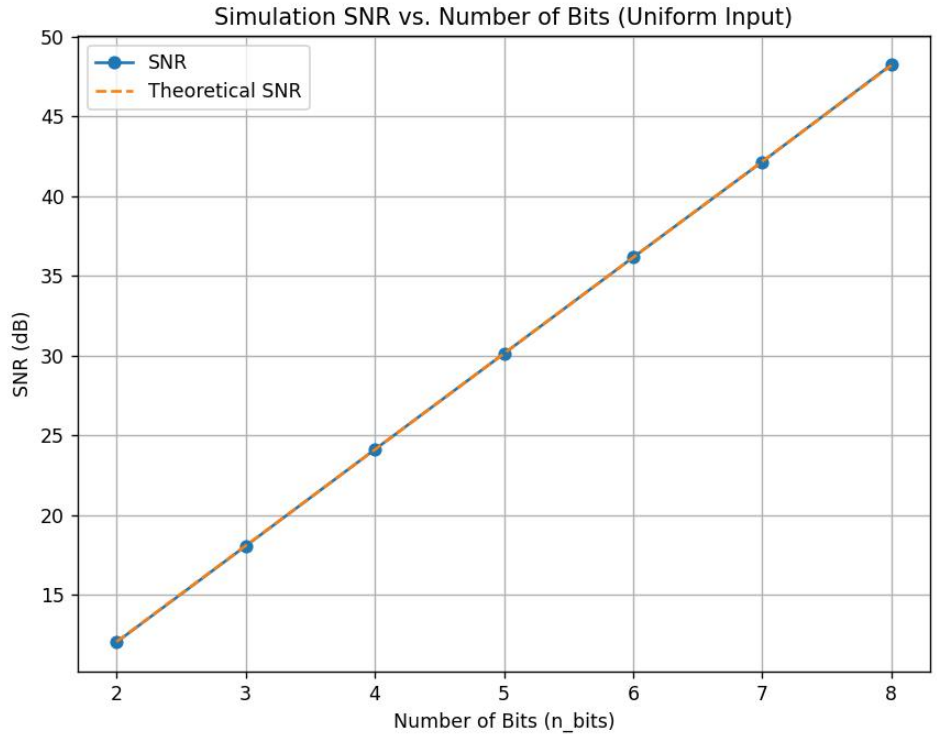

Figure 8 Fig

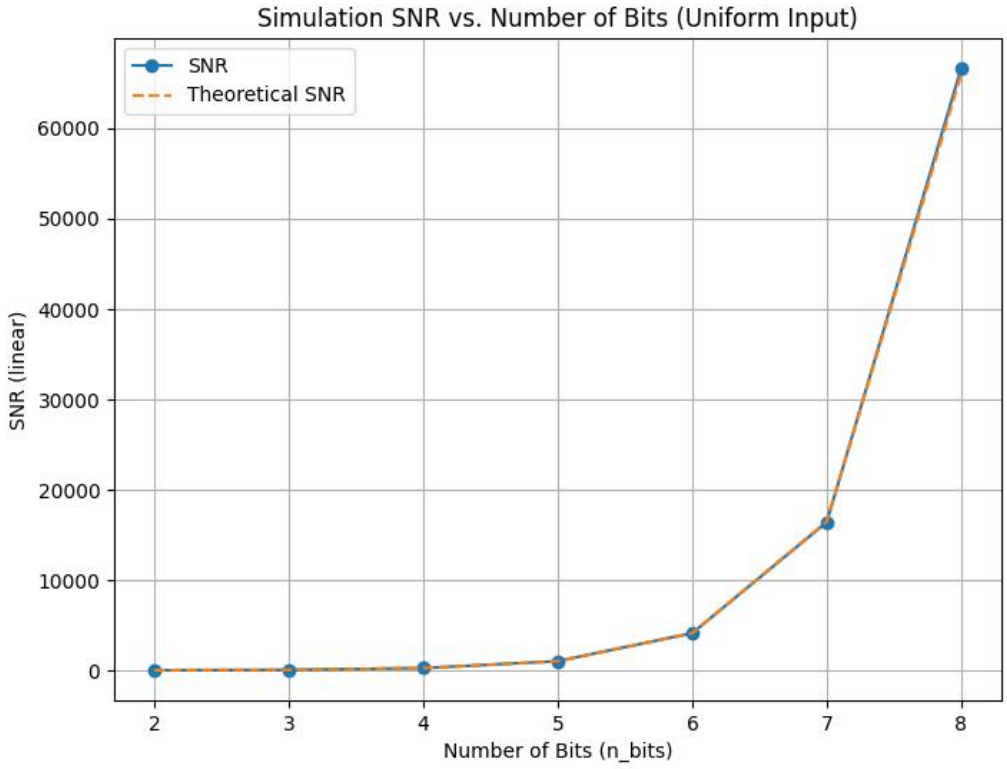

Figure 9 Fig

Comment:

We generated a uniform random variable and applied a uniform quantizer to it. Upon comparison, we found that the resulting Signal-to-Noise Ratio (SNR) from the uniform quantizer was identical to the theoretical SNR. This indicates that the quantizer's performance aligns closely with the expected theoretical values, demonstrating its accuracy.

Part 5: test the uniform quantizer on a non-uniform random input

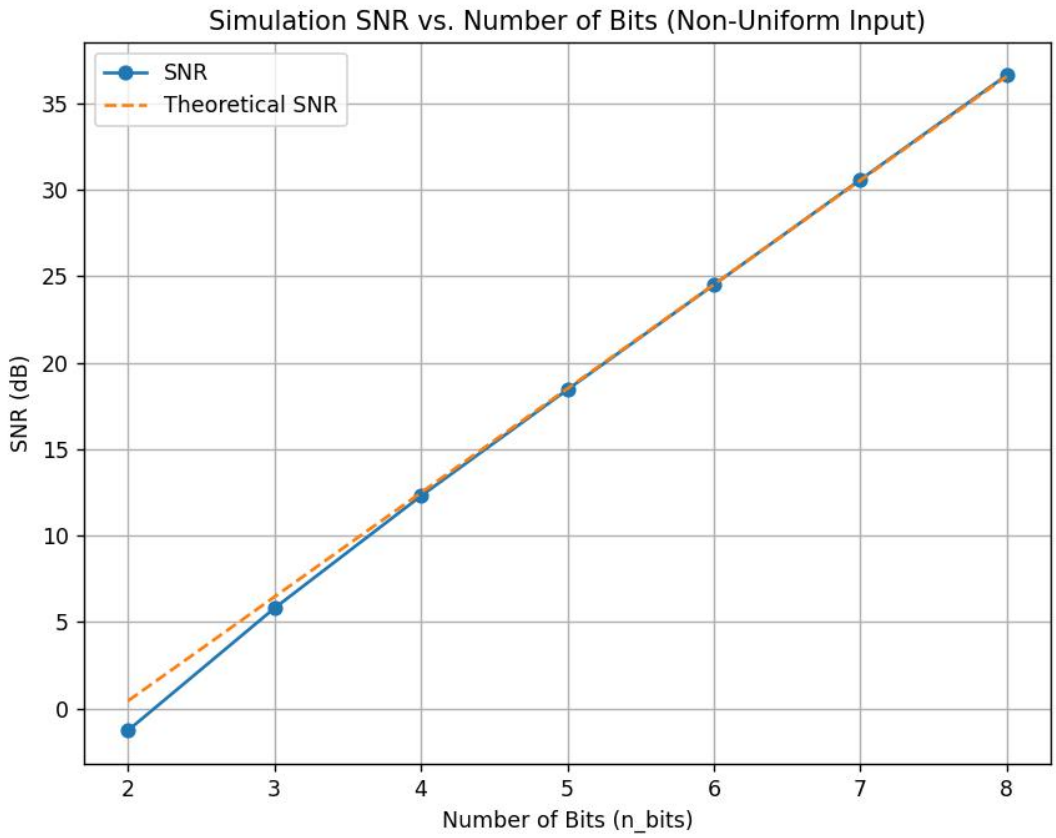

Figure 10 Fig

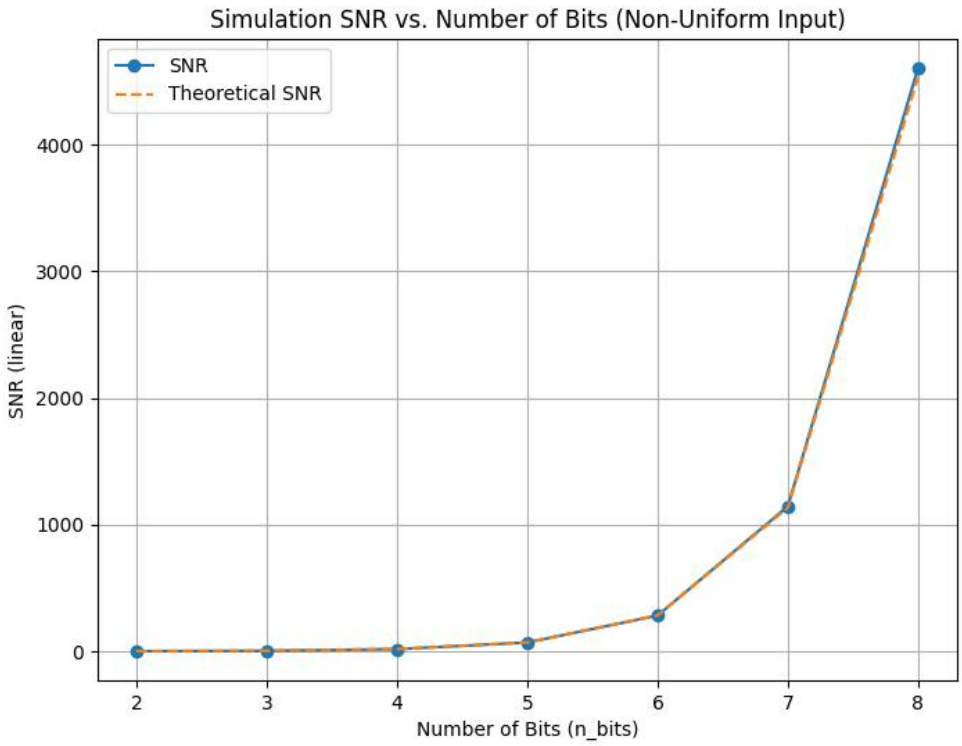

Figure 11 Fig

Comment:

In this part, we generated a non-uniform random variable and used it as the input to both the uniform quantizer and dequantizer. Upon analysis, we observed that the SNR values of the output signal from the quantizer were lower than the theoretical values, especially noticeable at lower values of n_bits. This was not the case with the previous section, where using a uniform random variable resulted in SNR values closely matching the theoretical values.

Part 6: quantize the the non-uniform signal using a non-uniform μ law quantizer

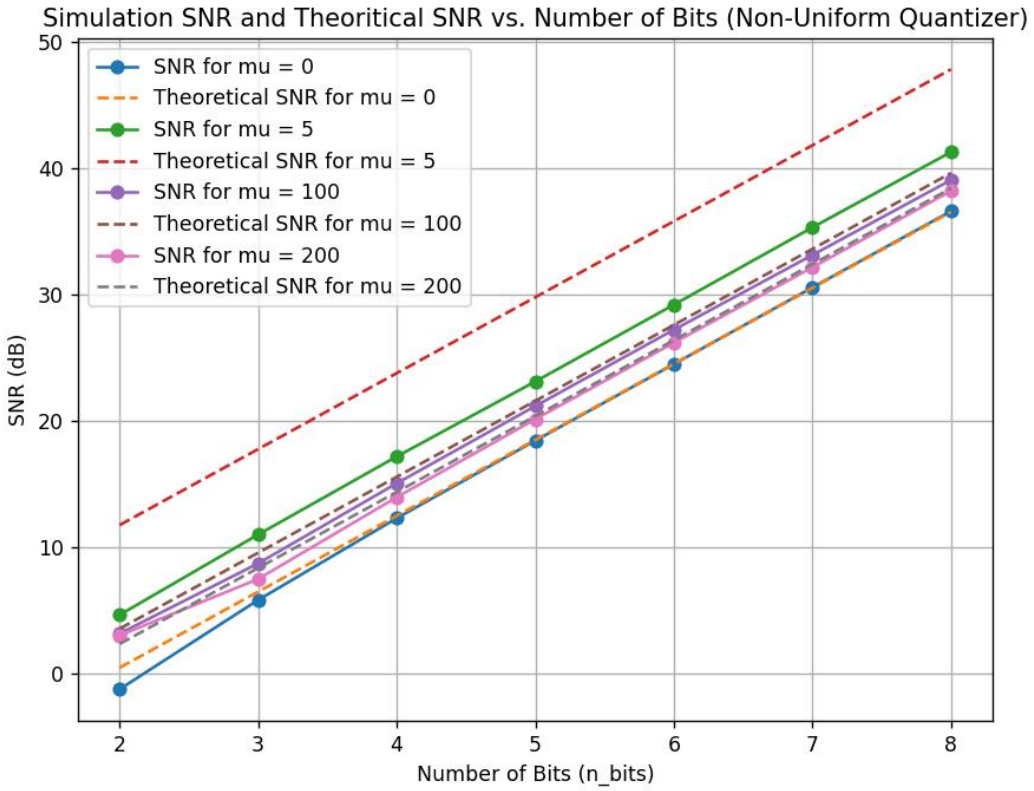

Figure 12 Fig

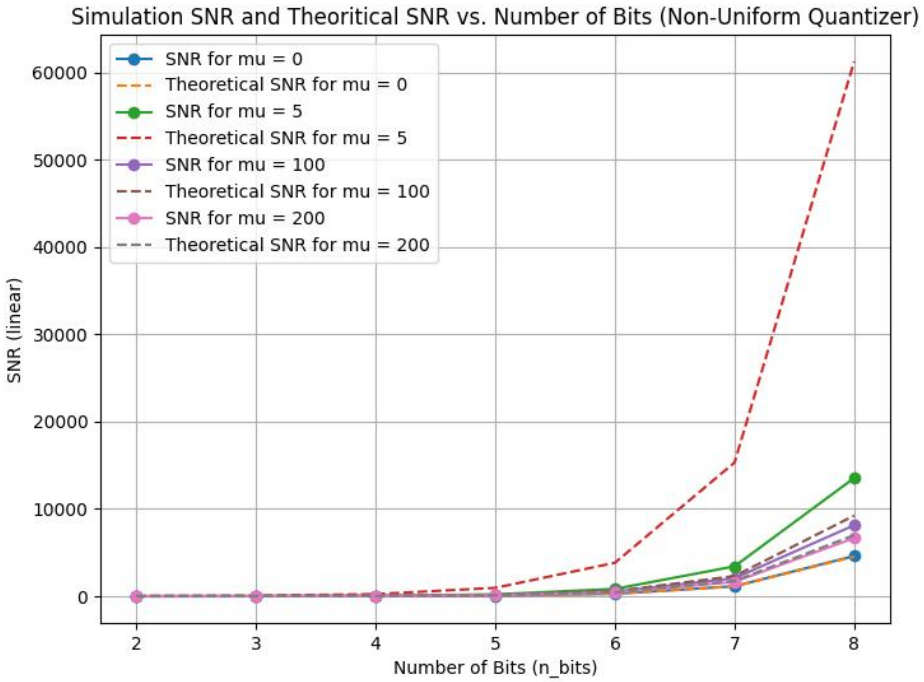

Figure 13 Fig

Comment:

In this part, we first passed the signal through a compressor before applying the uniform quantizer, followed by the dequantizer and then an expander. We compared the resulting Signal-to-Noise Ratio (SNR) with the theoretical SNR values for different values of μ

Index:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import expon

# Plot the input and de-quantized output
def plot(x,Y,labels,xlabel,ylabel,title):
    plt.figure(figsize=(8, 6))
    for i in range(len(labels)):
        plt.plot(x, Y[i], label=labels[i])
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.grid(True)
    plt.legend()
    plt.show()

def plot_SNR(x,Sim,Theo,Sim_labels,Theo_labels,xlabel,ylabel,title):
    plt.figure(figsize=(8, 6))
    for i in range(len(Sim_labels)):
        plt.plot(x, Sim[i],marker='o', label=Sim_labels[i])
        plt.plot(x, Theo[i],linestyle='--',label=Theo_labels[i])
        i = i + 1
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.grid(True)
    plt.legend()
    plt.show()

#############################################################################################
#                                      Requirement 1,2                                      #
#     Implement a uniform scalar quantizer and de-quantizer function with the header       #
#############################################################################################

def UniformQuantizer(in_val, n_bits, xmax, m):
    # Calculate the step size (quantization interval width)
    L = 2**n_bits
    delta = 2 * xmax / L

    # Quantize the input samples
    q_ind = ((in_val - ((m) * (delta / 2) - xmax)) / delta).astype(int)
    return q_ind

def UniformDequantizer(q_ind, n_bits, xmax, m):
    # Calculate the step size (quantization interval width)
    L = 2**n_bits
    delta = 2 * xmax / L
```

```
    # Reconstruct the de-quantized value
    deq_val = ((q_ind) * delta) + ((m+1) * (delta / 2) - xmax);

    return deq_val


##############################################################################################
#                                    Requirement 3                                           #
#           Test the quantizer/dequantizer functions on a deterministic input                #
##############################################################################################

# Generate the input ramp signal
x = np.arange(-6, 6, 0.01)

# Quantize the input signal (Midrise)
q_ind_midrise = UniformQuantizer(x, n_bits=3, xmax=6, m=0)

# De-quantize the quantized signal
deq_val_midrise = UniformDequantizer(q_ind_midrise, n_bits=3, xmax=6, m=0)

# Plot the input and de-quantized output
plot(x, [x, q_ind_midrise, deq_val_midrise], ['Input Ramp Signal', 'Quantized Output (m=0)', 'De-
Quantized Output (m=0)'], 'Input', 'Output', 'Uniform Scalar Quantization (Midrise)')


# Quantize the input signal with m = 1 (Midread)
q_ind_midtread = UniformQuantizer(x, n_bits=3, xmax=6, m=1)

# De-quantize the quantized signal with m = 1
deq_val_midtread = UniformDequantizer(q_ind_midtread, n_bits=3, xmax=6, m=1)

# Plot the input and de-quantized output for m = 1
plot(x, [x, q_ind_midtread, deq_val_midtread], ['Input Ramp Signal', 'Quantized Output (m=1)', 'De-
Quantized Output (m=1)'], 'Input', 'Output', 'Uniform Scalar Quantization (Midtread)')

##############################################################################################
#                                    Requirement 4                                           #
#                        test input on a random input uniform signal                         #
##############################################################################################

# Generate random input signal (10,000 samples)
np.random.seed(0)  # For reproducibility
input_signal = np.random.uniform(low=-5, high=5, size=10000)

# Parameters
xmax = 5
m = 0
n_bits_range = range(2, 9)
# Initialize arrays for SNR and quantization error
snr_values = []
theoretical_snr =  [ ]
# Calculate SNR and quantization error for each n_bits
```

```
for n_bits in n_bits_range:
    # Quantize the input signal
    q_ind = UniformQuantizer(input_signal, n_bits, xmax, m)
    # De-quantize the quantized signal
    deq_val = UniformDequantizer(q_ind, n_bits, xmax, m)
    # Calculate quantization error
    quant_error = input_signal - deq_val
    # Calculate SNR
    snr = np.mean(input_signal**2) / np.mean(quant_error**2)
    snr_values.append(snr)  # Convert to dB

    input_power = np.mean(input_signal**2)
    theoretical_snr.append((3 * (2 ** n_bits) ** 2 * input_power) /xmax**2)
# Plot SNR vs n_bits
plot_SNR(n_bits_range, [10 * np.log10(snr_values)],[10 * np.log10(theoretical_snr)],
     ['SNR'],['Theoretical SNR'], 'Number of Bits (n_bits)', 'SNR (dB)', 'Simulation SNR vs. Number
of Bits (Uniform Input)')
plot_SNR(n_bits_range, [snr_values],[theoretical_snr],
     ['SNR'],['Theoretical SNR'], 'Number of Bits (n_bits)', 'SNR (linear)', 'Simulation SNR vs.
Number of Bits (Uniform Input)')

##########################################################################################
#                                    Requirement 5                                       #
#                  test the uniform quantizer on a non-uniform random input              #
##########################################################################################

# Generate random input signal (10,000 samples)
np.random.seed(42)
num_samples = 10000
polarity = np.random.choice([-1, 1], size=num_samples, p=[0.5, 0.5])
magnitude = np.random.exponential(size=num_samples)
input_signal = polarity * magnitude

# Parameters
xmax = max(abs(input_signal))
m = 0
n_bits_range = range(2, 9)

# Initialize arrays for SNR and quantization error
snr_values = []
theoretical_snr = []
# Calculate SNR and quantization error for each n_bits
for n_bits in n_bits_range:
    # Quantize the input signal
    q_ind = UniformQuantizer(input_signal, n_bits, xmax, m)

    # De-quantize the quantized signal
    deq_val = UniformDequantizer(q_ind, n_bits, xmax, m)
```

```
    # Calculate quantization error
    quant_error = input_signal - deq_val

    # Calculate SNR manually
    snr = np.mean(input_signal**2) / np.mean(quant_error**2)
    snr_values.append(snr)  # Convert to dB
    input_power = np.mean(input_signal**2)
    theoretical_snr.append((3 * (2 ** n_bits) ** 2 * input_power) /xmax**2)

# Plot SNR vs n_bits
plot_SNR(n_bits_range, [10 * np.log10(snr_values)],[10 * np.log10(theoretical_snr)],
['SNR'],['Theoretical SNR'], 'Number of Bits (n_bits)', 'SNR (dB)', 'Simulation SNR vs. Number of
Bits (Non-Uniform Input)')
plot_SNR(n_bits_range, [snr_values],[theoretical_snr], ['SNR'],['Theoretical SNR'], 'Number of Bits
(n_bits)', 'SNR (linear)', 'Simulation SNR vs. Number of Bits (Non-Uniform Input)')

##############################################################################################
#                                    Requirement 6                                           #
#      quantize the the non-uniform signal using a non-uniform   law quantizer               #
##############################################################################################


# Define the compressor function
def compressor(x, mu):
    return np.sign(x) * (np.log(1 + mu * np.abs(x/xmax)) / np.log(1 + mu))


# Define the expander function
def expander(y, mu):
    return np.sign(y) * ((1 + mu) ** np.abs(y) - 1) / mu


# Parameters
mu_values = [0, 5, 100, 200]  # Different mu values


# Quantize and expand for each mu value
Sim_SNR = []
Theor_SNR = []
for mu in mu_values:
    snr_values = []
    theoretical_snr =  []
    for n_bits in n_bits_range:
        in_sig = input_signal
        if (mu > 0):
            in_sig = compressor(input_signal, mu)
        q_ind = UniformQuantizer(in_sig, n_bits, np.max(abs(in_sig)), 0)
        deq_val = UniformDequantizer(q_ind, n_bits, np.max(abs(in_sig)), 0)
        if (mu > 0):
            deq_val = expander(deq_val, mu)*xmax
```

```
        # Calculate quantization error
        quant_error = input_signal - deq_val

        # Calculate SNR
        snr = np.mean(input_signal**2) / np.mean(quant_error**2)
        snr_values.append(snr)  # Convert to dB
        input_power = np.mean(input_signal**2)
        if (mu > 0):
            theoretical_snr.append((3 * (2 ** n_bits) ** 2) / (np.log(1 + mu) ** 2))
        else:
            theoretical_snr.append((3 * (2 ** n_bits) ** 2 * input_power) /xmax**2)
    Sim_SNR.append(snr_values)
    Theor_SNR.append(theoretical_snr)

# Plot SNR vs n_bits
Sim_labels = ['SNR for mu = 0', 'SNR for mu = 5', 'SNR for mu = 100', 'SNR for mu = 200']
Theo_labels = ['Theoretical SNR for mu = 0', 'Theoretical SNR for mu = 5', 'Theoretical SNR for mu = 100', 'Theoretical SNR for mu = 200']
plot_SNR(n_bits_range, 10 * np.log10(Sim_SNR), 10 * np.log10(Theor_SNR), Sim_labels, Theo_labels, 'Number of Bits (n_bits)', 'SNR (dB)', 'Simulation SNR and Theoritical SNR vs. Number of Bits (Non-Uniform Quantizer)')
plot_SNR(n_bits_range, Sim_SNR, Theor_SNR, Sim_labels, Theo_labels, 'Number of Bits (n_bits)', 'SNR (linear)', 'Simulation SNR and Theoritical SNR vs. Number of Bits (Non-Uniform Quantizer)')
```