Faculty of Engineering – Cairo University
Computer Engineering Department

ELC 325B – Spring 2025

Digital Communications

Assignment #1

Quantization

Submitted to

Dr. Mai Badawi

Dr. Hala Abdelkader

Eng. Mohammed Khaled

Submitted by

Name	Sec	BN
Ahmed Hamdy Mohammed	1	4
Abdelrahman Mostafa Gomaa	1	31

Contents

Figures

Requirement – 1 Explanation

```python
def UniformQuantizer(in_val, n_bits, xmax, m):
    L = 2**n_bits  # Number of levels
    delta = 2 * xmax / L  # Quantization step size
    q_ind = (in_val - (m * (delta / 2) - xmax)) / delta  # Quantization index
    q_ind = q_ind.astype(int)  # Convert to integer
    return q_ind
```

Figure 1: Uniform Quantizer Function

- The **UniformQuantizer** function performs uniform quantization on input samples as follows:
  1- It calculates the quantization interval width using the formula: $\Delta = \frac{2x_{max}}{2^{n\_bits}}$
  2- It uses the formula: $q_{ind} = \frac{V_{in} - \frac{1}{2}m\Delta + x_{max}}{\Delta}$ to quantize the input samples correctly

Requirement – 2 Explanation

```python
def UniformDequantizer(q_ind, n_bits, xmax, m):
    L = 2**n_bits  # Number of levels
    delta = 2 * xmax / L  # Quantization step size
    deq_val = delta * (q_ind + 0.5 * (m + 1)) - xmax  # Dequantized value
    return deq_val
```

Figure 2: Uniform De-Quantizer Function

- The **UniformDequantizer** function gets dequantized values from quantized ones as follows:
  1- It calculates the dequantization level using the formula: $\Delta = \frac{2x_{max}}{2^{n\_bits}}$
  2- It reconstructs the values using the formula: $val_{deq} = \Delta\left(q_{ind} + \frac{1}{2}(m+1)\right) - x_{max}$

Requirement – 3 Explanation

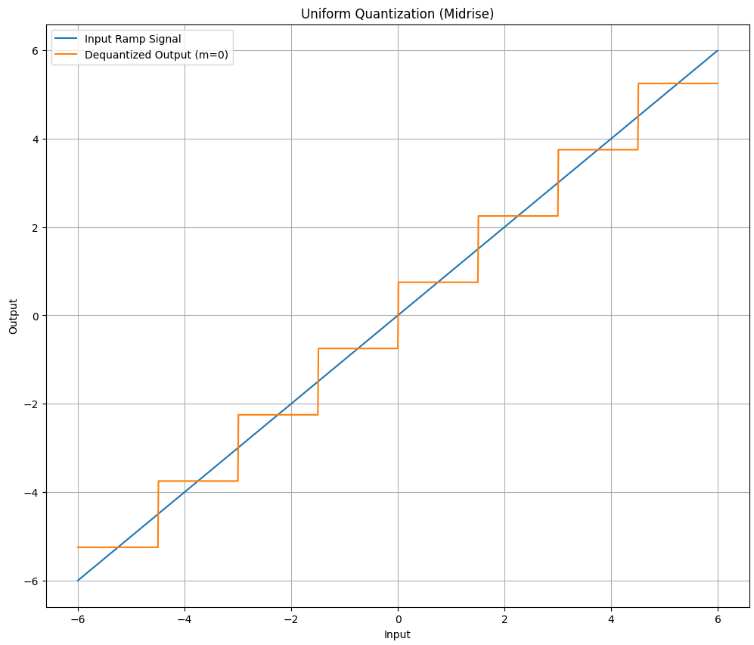

Figure 3: Midrise Plot

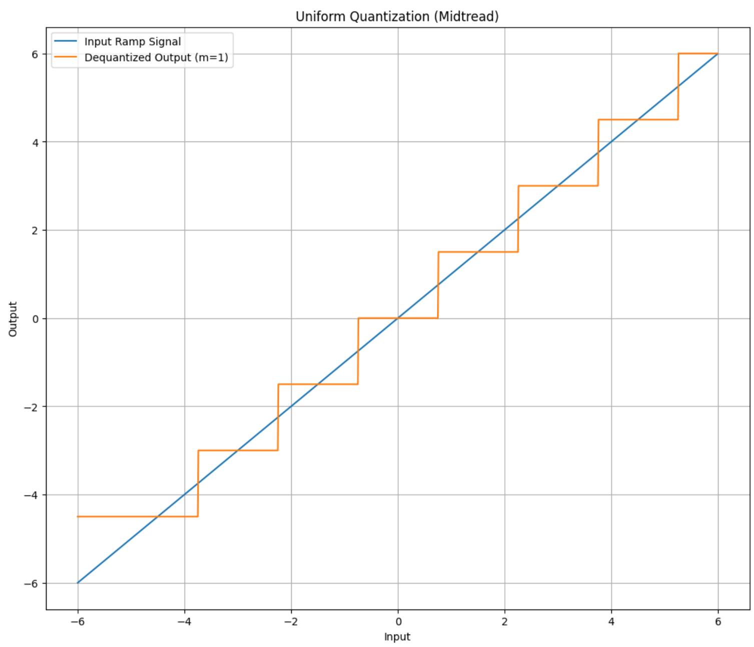

Figure 4: Midtread Plot

- As we see that the maximum quantization error in both cases for the midrise, and the midtread is $\frac{\Delta}{2} = 0.75V$
- In midtread quantization, the input value at zero corresponds to 0, whereas in midrise quantization, it rises instantly from $-\frac{\Delta}{2}$ to $\frac{\Delta}{2}$

Requirement – 4 Explanation

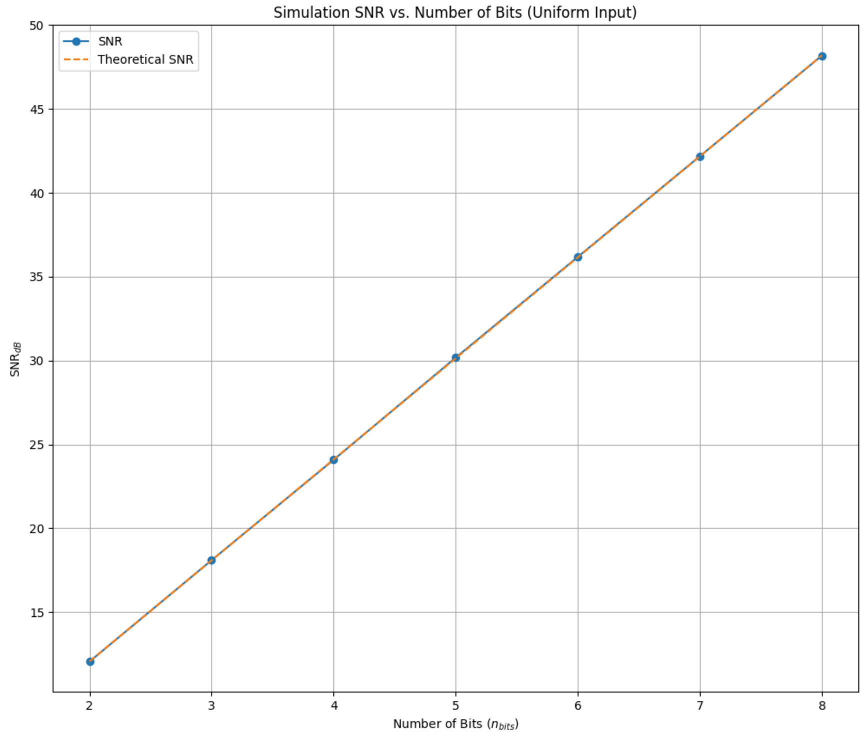

Figure 5: Response to a random i.i.d signal

- A uniform quantizer was applied to a uniform random variable that we had created. We discovered that the uniform quantizer's output Signal-to-Noise Ratio (SNR) matched the theoretical SNR exactly. This shows that the quantizer is accurate because its performance roughly matches the predicted theoretical values

Requirement – 5 Explanation

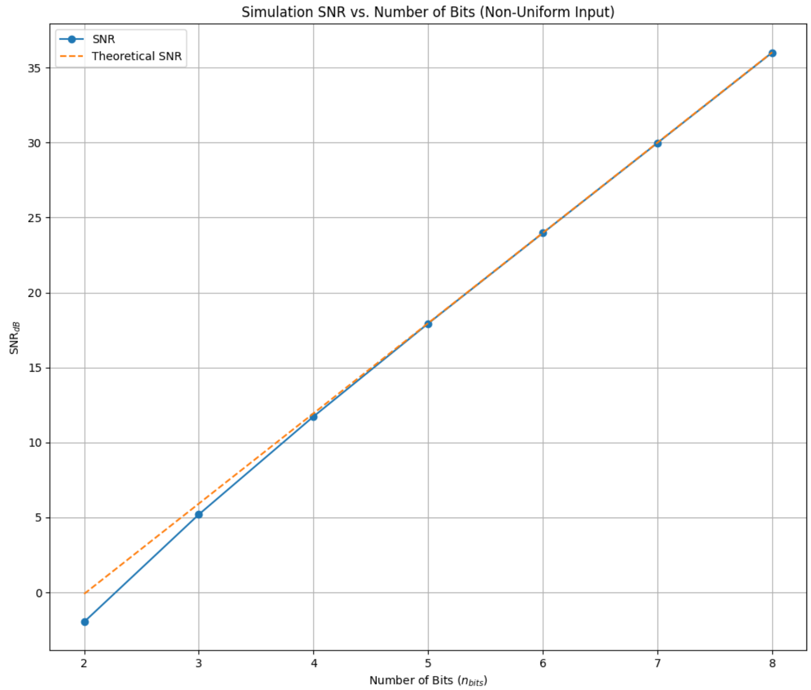

Figure 6: Response to a non-uniform random input signal

- A non-uniform random variable was created in this section and sent into the uniform quantizer and de-quantizer. After examination, we found that the quantizer's output signal's SNR values were below the theoretical values, particularly at lower ***n_bit*** values. This was not the case in the preceding part, where SNR values nearly matched the theoretical values due to the use of a uniform random variable.

Requirement – 6 Explanation

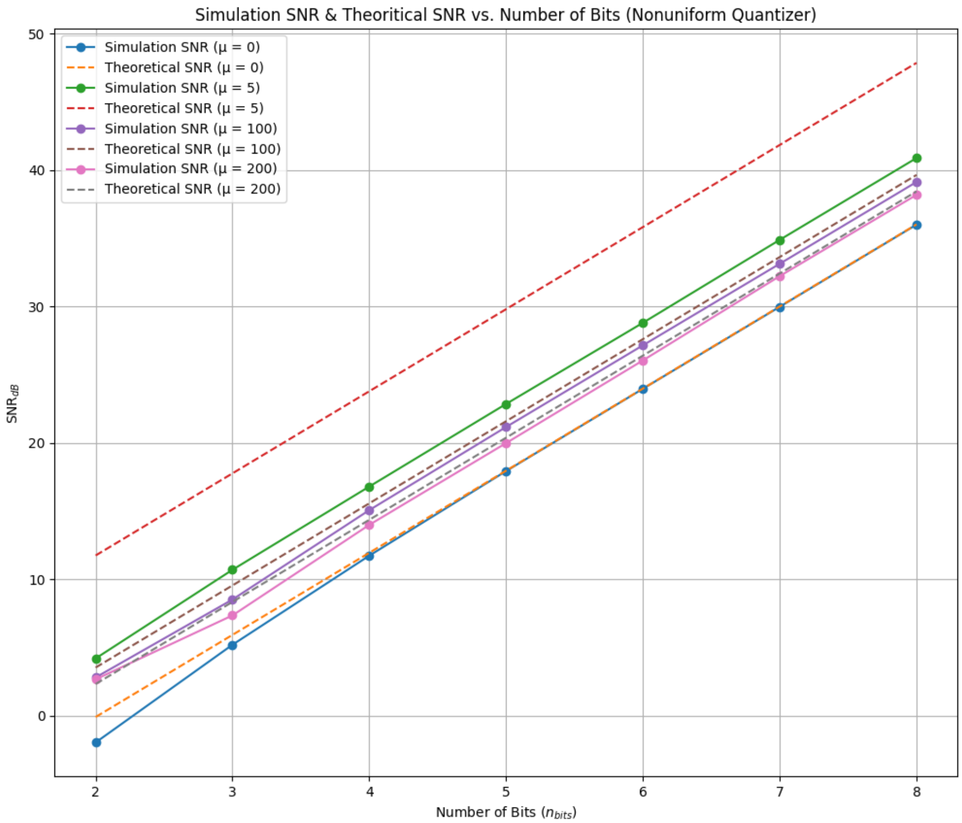

Figure 7: Responses of different µ-values

```python
def mu_law_compression(normalized_x, mu):
    return np.sign(normalized_x) * (np.log1p(mu * np.abs(normalized_x)) / np.log1p(mu))


def mu_law_expansion(y, mu):
    return np.sign(y) * ((1 + mu) ** np.abs(y) - 1) / mu
```

Figure 8: Compressor, and expansion function

- To handle the previous problem, we used nonuniform µ quantizer. We started with a µ compressor, then used a uniform quantizer, de-quantizer, and expander. We compared the Signal-to-Noise Ratio (SNR) to theoretical values for various µ values.

Index

Requirement – 1 Code

```
def UniformQuantizer(in_val, n_bits, xmax, m):
    L = 2**n_bits  # Number of levels
    delta = 2 * xmax / L  # Quantization step size
    q_ind = (in_val - (m * (delta / 2) - xmax)) / delta  # Quantization index
    q_ind = q_ind.astype(int)  # Convert to integer
    return q_ind
```

Figure 9: Requirement - 1 Flow

Requirement – 2 Code

```
def UniformDequantizer(q_ind, n_bits, xmax, m):
    L = 2**n_bits  # Number of levels
    delta = 2 * xmax / L  # Quantization step size
    deq_val = delta * (q_ind + 0.5 * (m + 1)) - xmax  # Dequantized value
    return deq_val
```

Figure 10: Requirement - 2 Flow

Requirement – 3 Code

```
def ramp_test():
    # Generate the input ramp signal
    n_bits = 3
    x_max = 6
    m = 0
    x = np.arange(-6, 6, 0.01)

    # Quantize the input signal using Midrise
    q_ind_midrise = UniformQuantizer(x, n_bits, x_max, m)

    # Dequantize the quantized signal
    deq_val_midrise = UniformDequantizer(q_ind_midrise, n_bits, x_max, m)

    # Plot the input and dequantized output
    plot(
        x,
        [x, deq_val_midrise],
        ["Input Ramp Signal", "Dequantized Output (m=0)"],
        "Input",
        "Output",
        "Uniform Quantization (Midrise)",
    )

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

    # Dequantize the quantized signal
    deq_val_midtread = UniformDequantizer(q_ind_midtread, n_bits=3, xmax=6, m=1)

    # Plot the input and dequantized output
    plot(
        x,
        [x, deq_val_midtread],
        ["Input Ramp Signal", "Dequantized Output (m=1)"],
        "Input",
        "Output",
        "Uniform Quantization (Midtread)",
    )


ramp_test()
```

Figure 11: Requirement - 3 Flow

Requirement – 4 Code

```python
def random_test():
    # Generate random input signal
    input_signal = np.random.uniform(low=-5, high=5, size=10000)

    xmax = 5
    m = 0
    n_bits_range = range(2, 9)
    snr_values = []
    theoretical_snr = []
    # Calculate SNR and quantization error for each n_bits
    for n_bits in n_bits_range:
        q_ind = UniformQuantizer(
            input_signal, n_bits, xmax, m
        )  # Quantize the input signal
        deq_val = UniformDequantizer(
            q_ind, n_bits, xmax, m
        )  # Dequantize the quantized signal
        quantization_error = input_signal - deq_val  # Calculate quantization error

        E_x2 = np.mean(input_signal**2)  # Calculate E[x^2]
        E_error2 = np.mean(quantization_error**2)  # Calculate E[error^2]
        snr = E_x2 / E_error2  # Calculate SNR
        snr_values.append(snr)
        theoretical_snr.append(
            (3 * (2**n_bits) ** 2 * E_x2) / xmax**2
        )  # Calculate Theoretical SNR
    # 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)",
    )


random_test()
```

Figure 12: Requirement - 4 Flow

Requirement – 5 Code

```
num_samples = 10000
polarity = np.random.choice(
    [-1, 1], size=num_samples, p=[0.5, 0.5]
)  # Randomly select polarity
magnitude = np.random.exponential(size=num_samples)  # Generate exponential magnitude
input_signal = polarity * magnitude  # Generate input signal

# Parameters
xmax = max(abs(input_signal))  # Calculate maximum value of input signal
m = 0
n_bits_range = range(2, 9)


def exponential_test():
    snr_values = []
    theoretical_snr = []
    for n_bits in n_bits_range:
        q_ind = UniformQuantizer(
            input_signal, n_bits, xmax, m
        )  # Quantize the input signal
        deq_val = UniformDequantizer(
            q_ind, n_bits, xmax, m
        )  # Dequantize the quantized signal
        quantization_error = input_signal - deq_val  # Calculate quantization error
        E_x2 = np.mean(input_signal**2)  # Calculate E[x^2]
        E_error2 = np.mean(quantization_error**2)  # Calculate E[error^2]
        snr = E_x2 / E_error2  # Calculate SNR
        snr_values.append(snr)
        # Calculate Theoretical SNR
        theoretical_snr.append((3 * (2**n_bits) ** 2 * E_x2) / 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)",
    )


exponential_test()
```

Figure 13: Requirement - 5 Flow

Requirement – 6 Code

```python
def mu_law_compression(normalized_x, mu):
    return np.sign(normalized_x) * (np.log1p(mu * np.abs(normalized_x)) / np.log1p(mu))


def mu_law_expansion(y, mu):
    return np.sign(y) * ((1 + mu) ** np.abs(y) - 1) / mu


def final_test():

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

    simulation_SNR_list = []

    theoritical_SNR_list = []

    for mu in mu_values:

        simulaton_snr = []  # Initialize SNR values

        theoretical_snr = []  # Initialize Theoretical SNR values

        for n_bits in n_bits_range:

            in_sig = input_signal  # Initialize input signal

            if mu > 0:  # If mu > 0, compress the input signal

                in_sig = mu_law_compression(input_signal / xmax, mu)

            # Quantize the input signal

            q_ind = UniformQuantizer(in_sig, n_bits, np.max(abs(in_sig)), 0)

            # Dequantize the quantized signal

            deq_val = UniformDequantizer(q_ind, n_bits, np.max(abs(in_sig)), 0)

            if mu > 0:

                deq_val = (
                    mu_law_expansion(deq_val, mu) * xmax
                )  # Expand the dequantized signal agian


            quantization_error = input_signal - deq_val  # Calculate quantization error

            E_x2 = np.mean(input_signal**2)  # Calculate E[x^2]

            E_error2 = np.mean(quantization_error**2)  # Calculate E[error^2]

            snr = E_x2 / E_error2  # Calculate SNR

            simulaton_snr.append(snr)

            # Calculate Theoretical SNR

            if mu > 0:

                theoretical_snr.append((3 * (2**n_bits) ** 2) / (np.log1p(mu) ** 2))

            else:

                theoretical_snr.append((3 * (2**n_bits) ** 2 * E_x2) / xmax**2)

        simulation_SNR_list.append(simulaton_snr)

        theoritical_SNR_list.append(theoretical_snr)


    # Plot SNR vs n_bits

    simulaton_labels = [
        "Simulation SNR (μ = 0)",
        "Simulation SNR (μ = 5)",
        "Simulation SNR (μ = 100)",
        "Simulation SNR (μ = 200)",
    ]

    theoretical_labels = [
        "Theoretical SNR (μ = 0)",
        "Theoretical SNR (μ = 5)",
        "Theoretical SNR (μ = 100)",
        "Theoretical SNR (μ = 200)",
    ]

    plot_SNR(
        n_bits_range,
        10 * np.log10(simulation_SNR_list),
        10 * np.log10(theoritical_SNR_list),
        simulaton_labels,
        theoretical_labels,
        "Number of Bits ($n_{bits}$)",
        "SNR$_{dB}$",
        "Simulation SNR & Theoritical SNR vs. Number of Bits (Nonuniform Quantizer)",
    )



final_test()
```

Figure 14: Requirement - 6 Flow