Cairo University

Faculty of Engineering
Cairo University

Digital Communications

First Assignment Report

Quantization

Submitted by:

Yasmine Ashraf Ghanem

Yasmin Abdullah Nasser

ASSIGNMENT DESCRIPTION AND FIGURES

In this assignment we performed uniform and non-uniform quantization and dequantization on a ramp signal and a random signal.

REQUIREMENT ONE:

First we implemented the uniform quantizer which takes the sampled signal as a parameter, the number of bits used to quantize each sample, the maximum sample value, and whether the quantization is midrise or midtread. The uniform quantizer calculates the quantizing step (Δ), the minimum value and the quantizing error at which the values are quantized. It returns the index for which each sample belongs to.

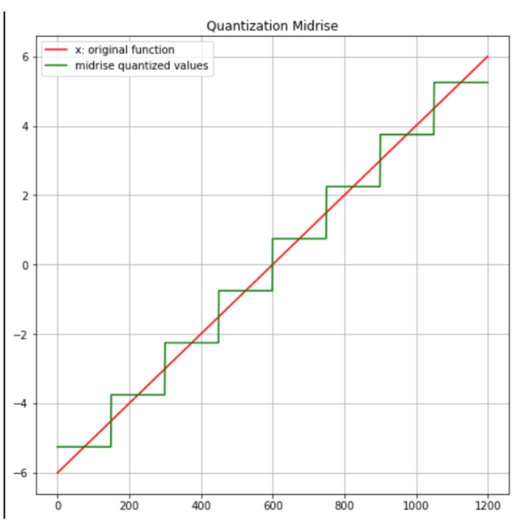
REQUIREMENT TWO:

The output of the uniform quantizer is an array of all the indices each sampled value belongs to. Then we implemented the uniform de-quantizer which takes the output of the quantizer and then converts the index of each value to an actual amplitude value for each level.

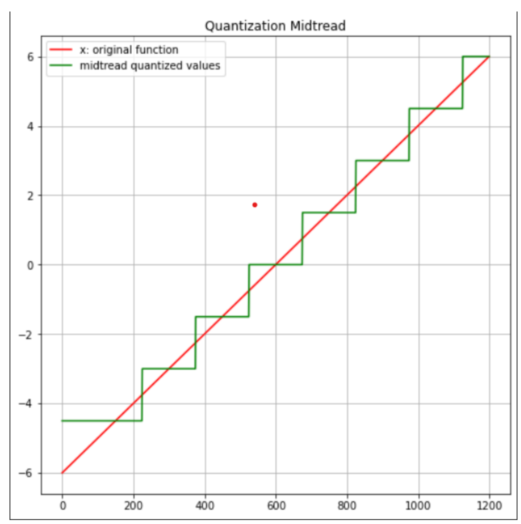
REQUIREMENT THREE:

We then generated a ramp signal and applied the quantization and dequantization functions for both midrise and midtread and displayed the resulting graphs.

Midrise:

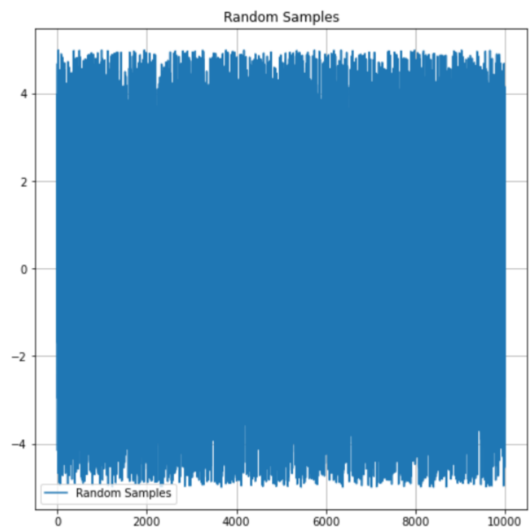


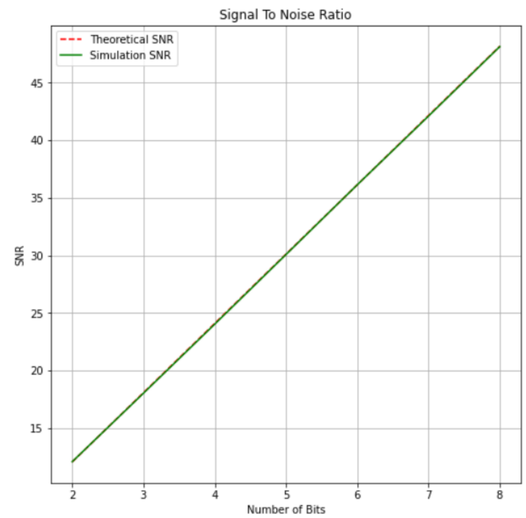
Midtread:



REQUIREMENT FOUR:

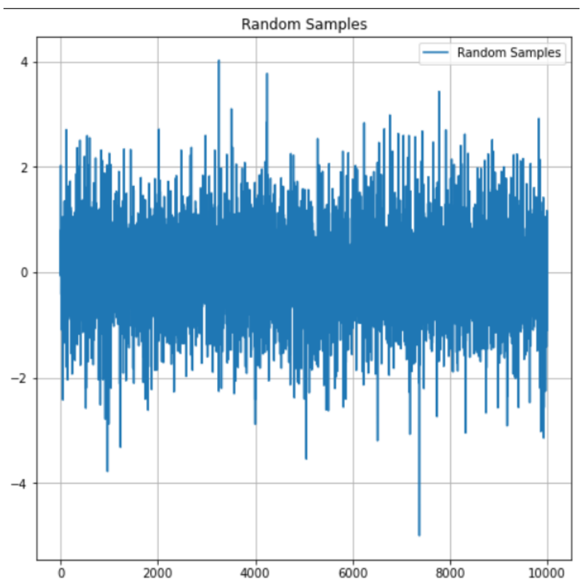
Then we generated a random uniform signal and applied the same process as before for different numbers of quantization bits (range from 2 - 8 bits) and calculated the theoretical and simulation signal to noise ratio.

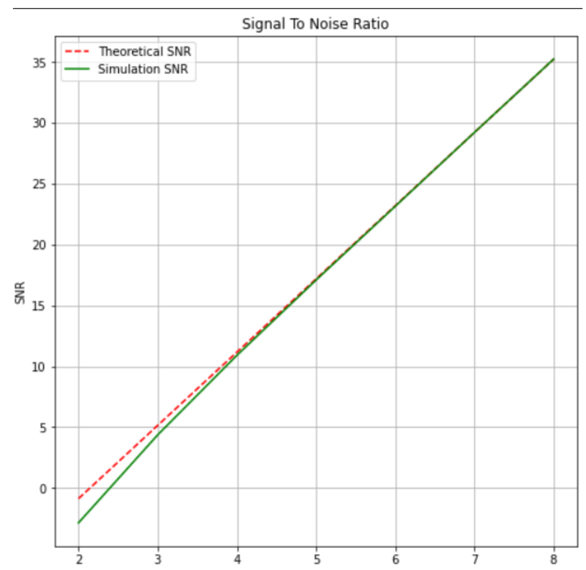




REQUIREMENT FIVE:

We repeated the previous step for a non-uniform input signal with PDF f(x)= e^(-x), and +/- with probability 0.5.

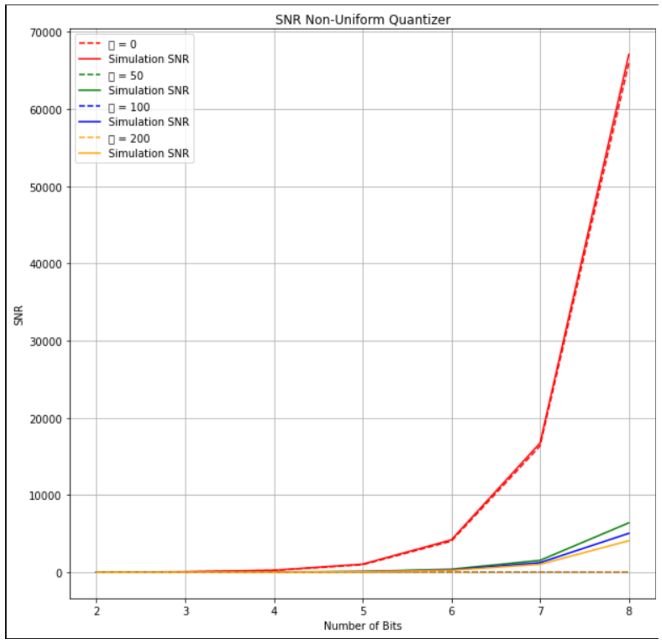




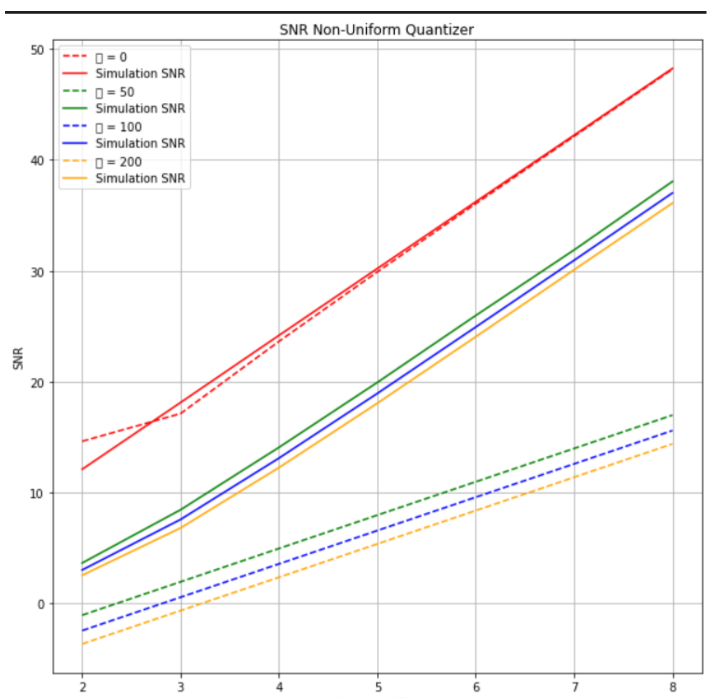
REQUIREMENT SIX:

We repeated the previous step for a non-uniform input signal with PDF f(x)= e^(-x), and +/- with probability 0.5.

Linear:



Logarithmic (dB):



CODE

```
# GENERAL FUNCTIONS

# Calculate experimental and theoretical signal to noise ratio of of the signal
# Parameters:
#   samples : vector of function samples
#   quantized_levels : vector of quantized samples
#   n_bits : number of bits to decode the level
#   max_value : max value in samples
# Output:
#   theoretical_snr : calculate the SNR with the equation
#   simulation_snr : calculate the SNR from the real values
def SNR(samples, quantized_levels, n_bits, xmax, 𝜇=0):
    quantization_error = samples - quantized_levels
    signal_power = np.mean(samples**2)
    error_power = np.mean(quantization_error**2)

    theoretical_snr = ((3* (2**n_bits)) / ((np.log(1+𝜇))**2)) if 𝜇 > 0 else ((3 * (2**n_bits)**2 * signal_power) / xmax**2)
    simulation_snr = signal_power/error_power
    return theoretical_snr, simulation_snr

# µ-Law Quantizer
# Expands signal
def ExpandSignal(samples, 𝜇):
    return np.sign(samples) * (np.log(1 + 𝜇 * np.absolute(samples)) / np.log(1 + 𝜇)) if 𝜇 > 0 else np.copy(samples)

# µ-Law Quantizer
# Compresses signal
def CompressSignal(samples, 𝜇):
    if(𝜇 > 0):
        y = ((1 + 𝜇) ** np.absolute(samples) - 1) / 𝜇
        return y * (np.sign(samples))
    else:
        return np.copy(samples)
```

```python
# Requirement 1 - Implement a uniform scalar quantizer function
# Parameters:
#     in_val : vector with the original samples
#     n_bits : number of bits available to quantize one sample
#     xmax : max value in samples values
#     m : 0 -> mibrise, 1 -> midtread
# Output:
#     q_ind :  index of the chosen quantization level
def UniformQuantizer(in_val, n_bits, xmax, m):
    q_ind = np.zeros(len(in_val))

    # number of quantized levels
    number_of_levels = 2**n_bits

    # delta is the width of each quantization level
    delta = 2 * xmax / number_of_levels

    xmin = xmax - (delta - ((1 - m) * (delta / 2)))

    i = 0
    for value in in_val:
        if(value < -xmin):
            value = -xmin
        error = (value + xmin) % delta
        q_ind[i] = (value + xmin) // delta
        if(error > delta/2):
            q_ind[i] = q_ind[i] + 1
        i = i + 1
    return q_ind
```

```python
# Requirement 2- Implement a uniform scaler de-quantizer function with the
header
# Parameters:
#     q_ind : vector with the quantized values indeces
#     n_bits : number of bits available to quantize one sample
#     xmax : max value in samples values
#     m : 0 -> mibrise, 1 -> midtread
# Output:
#     deq_value :  value of each index
def UniformDequantizer(q_ind,n_bits,xmax,m):
```

```python
    number_of_levels = 2**n_bits
    delta = 2 * xmax / number_of_levels
    xmin = xmax - (delta - ((1 - m) * (delta/2)))
    deq_val = [(value * delta - xmin) for value in q_ind]
    return deq_val
```

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

# 3.1- Midrise quantization: m = 0
x = np.linspace(-6, 6, 1200)
midrise_quantization = UniformQuantizer(x, 3, 6, 0)
midrise_dequantization = UniformDequantizer(midrise_quantization, 3, 6, 0)

# 3.2- Midtread quantization: m = 1
midrise_quantization = UniformQuantizer(x, 3, 6, 1)
midrise_dequantization = UniformDequantizer(midrise_quantization, 3, 6, 1)
```

```python
# Requirement 4 - Now test your input on a random input signal as follows:
random_samples = np.random.uniform(low=-5, high=5, size=10000)

theoretical_snr = []
simulation_snr = []

# calculate for number of bits 2:1:8
for i in range(2, 9):
    # calculate quntization level for each samples (midrise)
    quantized = UniformQuantizer(random_samples, i, 5, 0)
    dequantized = UniformDequantizer(quantized, i, 5, 0)
    snr_t, snr_s = SNR(random_samples, dequantized, i, 5)
    theoretical_snr.append(snr_t)
    simulation_snr.append(snr_s)

# convert to in db
theoretical_snr = 10*np.log10(theoretical_snr)
simulation_snr = 10*np.log10(simulation_snr)
```

```python
# Requirement 5 - Test the uniform quantizer on a non-uniform random input
```

```
random_samples = np.random.exponential(1, 10000)
random_samples = (random_samples / np.amax(random_samples)) * (5 *
np.random.choice([-1, 1], size=(10000), p=[0.5, 0.5]))
theoretical_snr = []
simulation_snr = []

# calculate for number of bits 2:1:8
for i in range(2, 9):
    # calculate quantization level for each samples (midrise)
    quantized = UniformQuantizer(random_samples, i,
np.amax(random_samples), 0)
    dequantized = UniformDequantizer(quantized, i,
np.amax(random_samples), 0)
    snr_t, snr_s = SNR(random_samples, dequantized, i,
np.amax(random_samples))
    theoretical_snr.append(snr_t)
    simulation_snr.append(snr_s)

# convert to in db
theoretical_snr = 10*np.log10(theoretical_snr)
simulation_snr = 10*np.log10(simulation_snr)
```

```
# Requirement 6 - Quantize the the non-uniform signal using a non-uniform
μ law quantizer
colors = ['red', 'green', 'blue', 'orange']
colors_counter = 0

plt.figure(figsize=(10,10))

for μ in [0, 50, 100, 200]:
    expanded_signal = ExpandSignal(random_samples, μ)
    theoretical_snr = []
```

```
    simulation_snr = []

    # calculate for number of bits 2:1:8
    for j in range(2, 9):
        # calculate quntization level for each samples (midrise)
        expanded_quantized = UniformQuantizer(expanded_signal, j,
np.amax(expanded_signal), 0)
        expanded_dequantized = UniformDequantizer(expanded_quantized, j,
np.amax(expanded_signal), 0)
        compressed_signal = CompressSignal(expanded_dequantized, μ)
        snr_t, snr_s = SNR(random_samples, compressed_signal, j,
np.amax(compressed_signal), μ)
        theoretical_snr.append(snr_t)
        simulation_snr.append(snr_s)

    # convert to in db
    theoretical_snr = 10*np.log10(theoretical_snr)
    simulation_snr = 10*np.log10(simulation_snr)
```