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
Eslam Ashraf	1	13
Nour Ziad	2	31

Contents

Figures

Introduction:

This report is mainly talking about using uniform quantizer and see the affect of this quantizer on both uniform signal and non-uniform ones with small modification on the quantizer also calculate the signal to noise ratio for each to compare results when different number of bits are used to transmit the signal which means a more or less resolution on transmitting, for the non-uniform signal we used smaller quantization steps for smaller signal amplitudes this achieved through compressing the signal using the **μ-Law** Companding Technique then applying uniform quantizer which is equivalent to non-uniform quantization

Part 1: Uniform Quantizer

```
function q_ind = UniformQuantizer(in_val, n_bits, xmax, m)
    delta=(2*xmax)/(2^n_bits);
    xmin = m*(delta/2)-xmax;

    % change values less than xmin to meet xmin in case of mid-tread
    internal_val = xmin .* ones(1,length(in_val));
    cond = (in_val >= xmin);
    internal_val(cond) = in_val(cond);

    % get the level of each point in the signal
    q_ind = min(floor((internal_val - xmin)/delta)+1 , (2^n_bits));
end
```

Figure 1 Quantization Code

Comment:

Function is responsible to quantize and encode the transmitted signal to a specific number of levels specified by the number of bits it takes so the block outputs the zero-based index of the associated region.

The algorithm depended on calculating the number of levels to get the delta (step) value then calculating the minimum value (x_min) can the quantized signal take depending on the mode of the function whether it is mid-rise or mid-tread as the quantization levels in the mid-rise type are even in number where they are odd at mid-tread type so we had to get over that and transmit the signal in exactly **n** number of bits then we mapped the values to their levels

Part 2: Uniform De Quantizer

```
function deq_val = UniformDequantizer(in_val, n_bits, xmax, m)
    delta=(2*xmax)/(2^n_bits);
    x_min = xmax - ((1 - m) * 0.5*delta);
    deq_val = (in_val - (1 - m)) * delta - x_min;
end
```

Figure 2 Dequantization Code

Comment:

- Dequantizer block: which was mainly responsible to decode the quantized signal and get the range back to the originally transmitted signal using previously calculated delta and x_min values

Part 3: Quantize Dequantize Ramp Signal

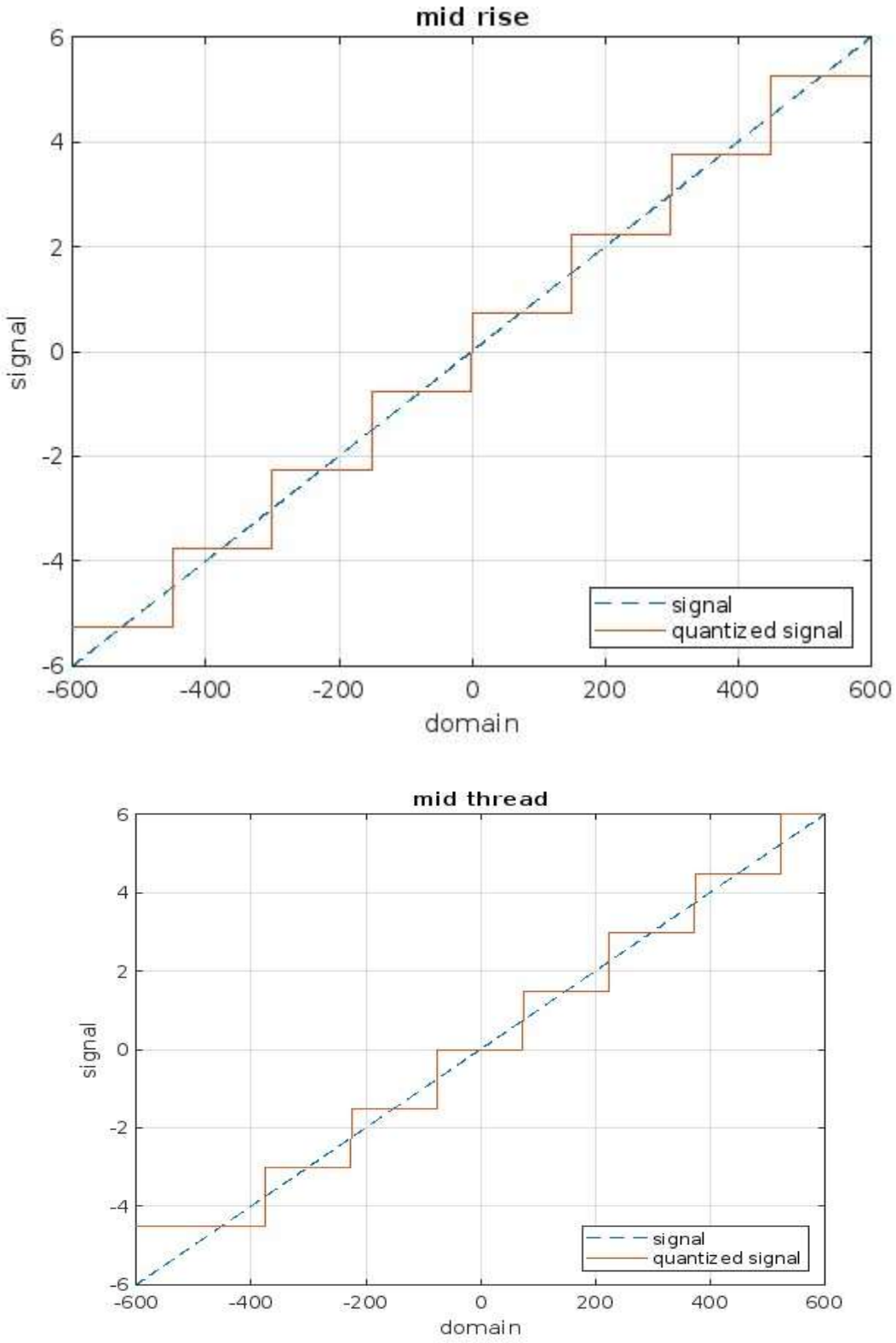

Figure 3 Ramp Signal Midrise and Midtread

Comment:

- **Midrise Figure**

  The figure shows a ramp function (dashed) which values ranges from -6 to 6 and the output of the quantizer and dequantizer for this signal where we used the mid-rise staircase for transmitting and using number of bits = 3 which leads to have 8 levels of values. the origin lies in the middle of a raising part of the stair-case like graph.

- **Midtread Figure**

  This figure shows the same ramp function (dashed) but when we used the mid-tread staircase for transmitting the signal, as shown the origin lies in the middle of a tread of the stair-case like graph.

Part 4: Uniform Quantizer and De Quantizer on uniform random Signal

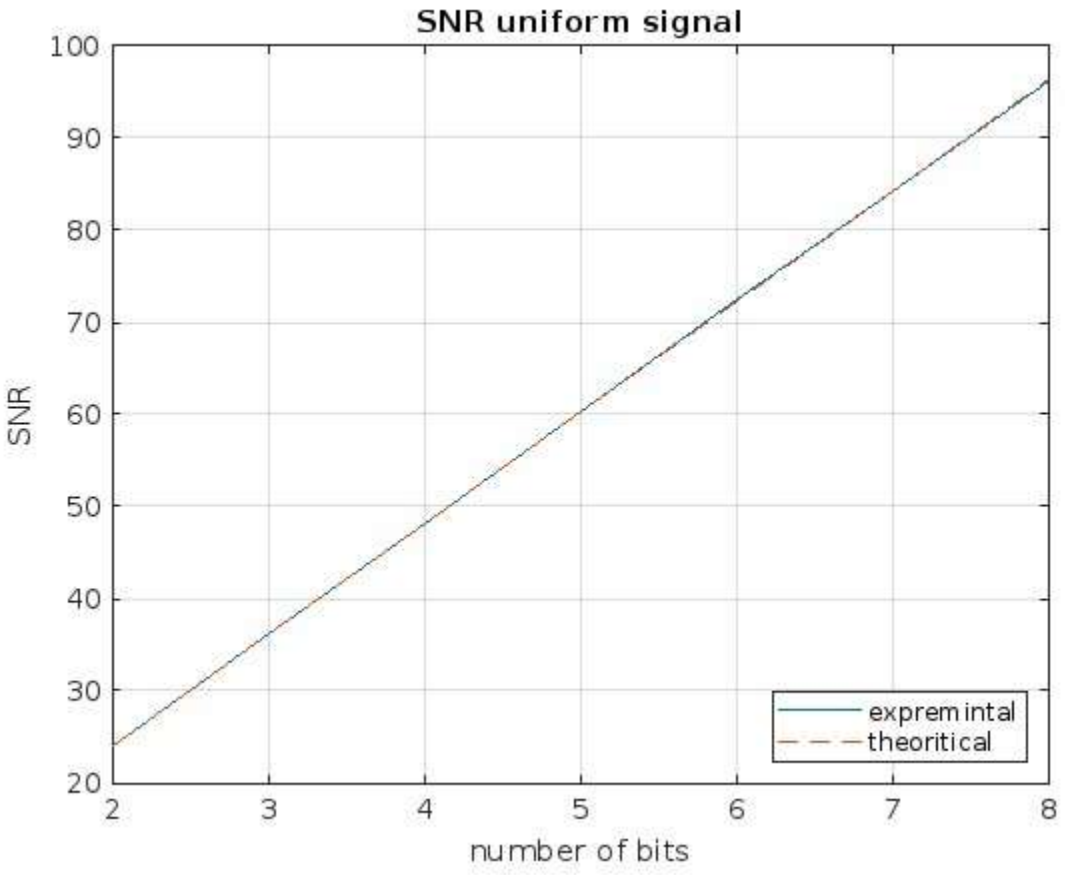

Figure 4 SNR theoretical vs Experimental Uniform Random Signal

Comment:

The signal-to-noise ratio (SNR) is one of the performance measures used to describe communication systems.

In this particular figure we did generate a uniform random function and tried to transmit it at different number of levels to see the affect of quantizer and dequantizer on it

For this we used SNR block to calculate the signal-to-noise ratio at different number of bits for both the theoretical side of the error using the equation provided with the lecture slides and the experimental side of it by calculating the power of the input signal divided on the power of the quantization error

We can see that two graphs are identical as it is a uniform quantizer applied on a uniform signal and didn't expect to give different values for SNR from the theoretically proved functions

Part 5: Uniform Quantizer and De Quantizer on non-uniform random Signa

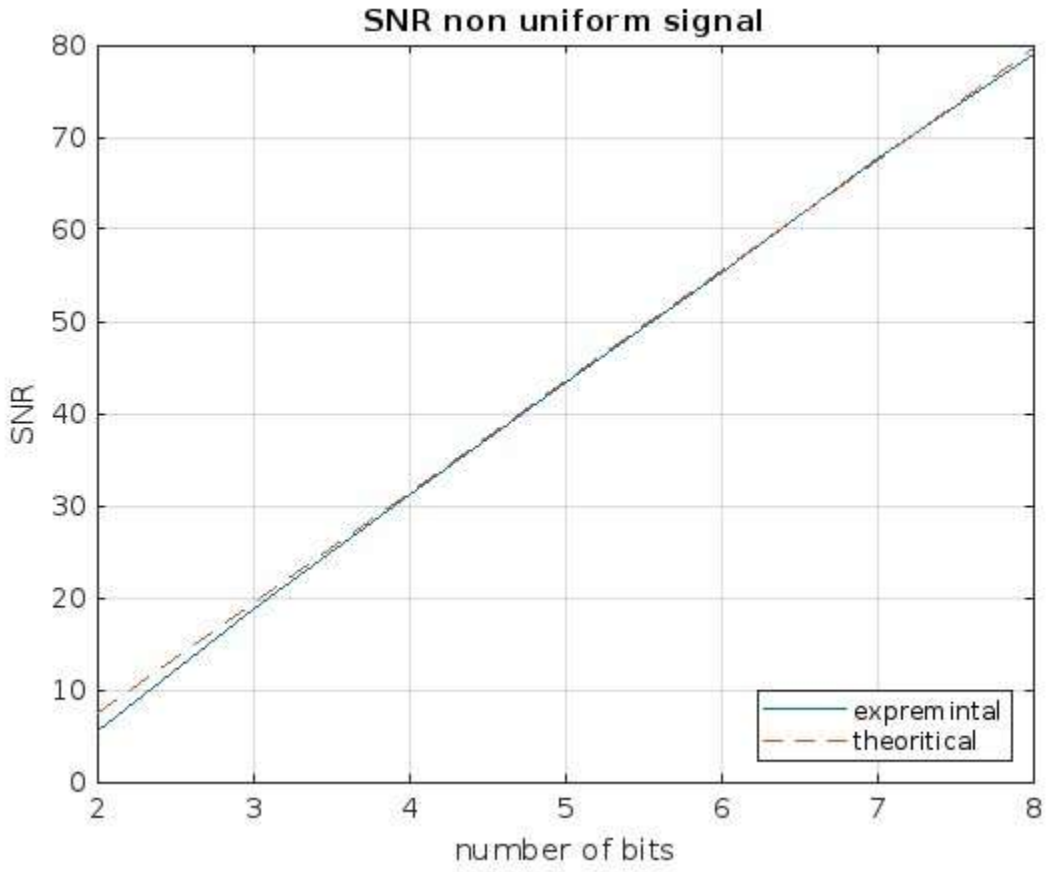

Figure 5 SNR theoretical vs Experimental non-Uniform Random Signal with Uniform Quantizer

Comment:

For this figure we generated a non-uniform random function and tried the uniform quantizer on it and we discovered that for small number of bits we transmit at, the SNR value was less than expected and theoretically proved in equations as it is a uniform quantizer applied on a non-uniform signal, in the other hand at higher number of bits we can hardly see a difference

Part 6: Using a non-uniform μ law quantizer

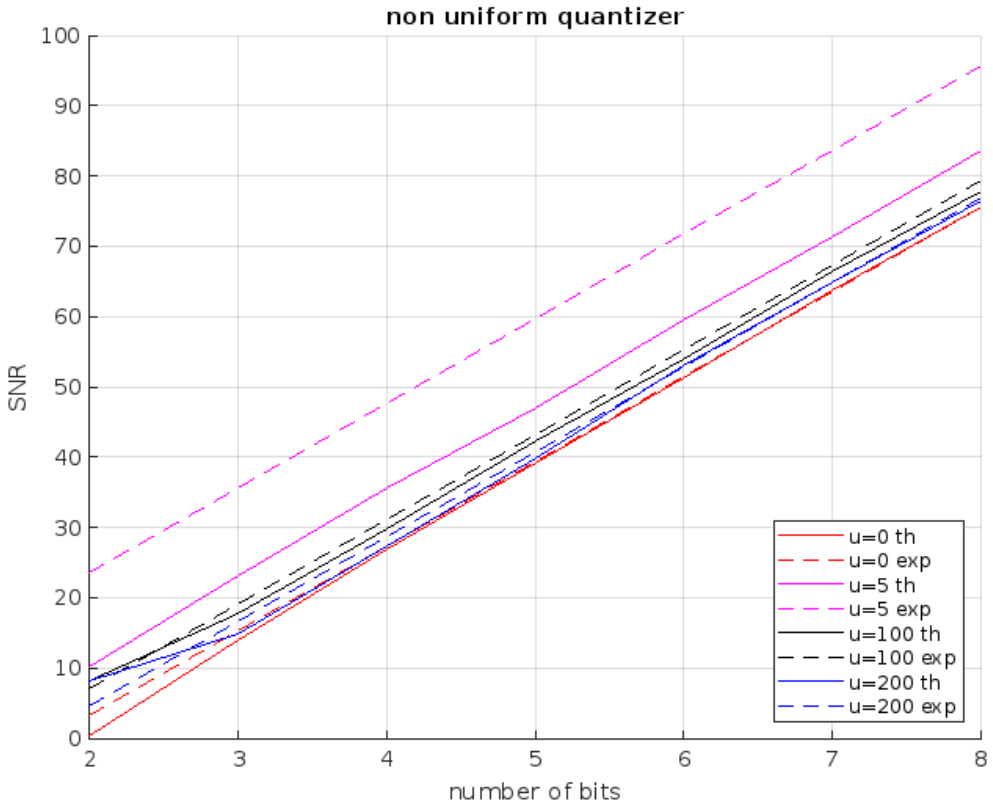

Figure 6 SNR theoretical vs Experimental non-Uniform Random Signal with Uniform Quantizer non-uniform μ law quantize

Comment:

For this part we used non-uniform **μ-Law** quantizer for different **μ-val** using zero value for the **μ** means uniform quantizing the signal the blocks output exactly the same graph as last mentioned one and for different **μ-values** we could obviously see the difference between theoretically calculated SNR and experimentally calculated one

Index:

```
% difining the signal
x = -6:0.01:6;
t = -fix(length(x)/2):1:fix(length(x)/2);


% req3 a: quantized signal when midrise algorithm used
encoded_signal = UniformQuantizer(x, 3, 6, 0);
quantized_signal = UniformDequantizer(encoded_signal, 3, 6, 0);

figure(1);
subplot(1,1,1);
plot(t,x,'--',t,quantized_signal)
title('mid rise');
xlabel 'domain';
ylabel 'signal';
legend({'signal','quantized signal'},'Location','southeast')
grid on


% req3 b: quantized signal when midtread algorithm used
encoded_signal = UniformQuantizer(x, 3, 6, 1);
quantized_signal = UniformDequantizer(encoded_signal, 3, 6, 1);

figure(2);
subplot(1,1,1);
plot(t,x,'--',t,quantized_signal)
title('mid thread');
xlabel 'domain';
ylabel 'signal';
legend({'signal','quantized signal'},'Location','southeast')
grid on


% req4: generate a uniform rendom signal
random_signal = random('Uniform',-5, 5,1, 10000);

% calculate SNR for the uniform random signal using midrise and n_bits [2:8]
SNR_exp_arr = zeros(7);
SNR_th_arr = zeros(7);
for i=2:8 % n_bits from 2:8
    encoded_signal = UniformQuantizer(random_signal, i, 5, 0);
    quantized_signal = UniformDequantizer(encoded_signal, i, 5, 0);
    [SNR_exp, SNR_th] = SNR(random_signal, quantized_signal, i, 0,
max(abs(random_signal)));
    SNR_exp_arr(i-1) = SNR_exp;
    SNR_th_arr(i-1) = SNR_th;
end
figure(3);
subplot(1,1,1);
n = 2:1:8;
plot(n,mag2db(SNR_exp_arr),n,mag2db(SNR_th_arr),'--')
title('SNR uniform signal');
xlabel 'number of bits';
ylabel 'SNR';
legend({'expremintal','theoritical'},'Location','southeast')
```

```
grid on
% req5: generate a non uniform random signal and test uniform quantizer
size = [1 1000];
exp_signal = exprnd(1,size);
sin = (randi([0,1],size)*2)-1;
rand_sig = exp_signal.*sin;
max_rand = max(abs(rand_sig));
SNR_exp_arr = zeros(7);
SNR_th_arr = zeros(7);
for i=2:8
    encoded_signal = UniformQuantizer(rand_sig, i, max_rand, 0);
    quantized_signal = UniformDequantizer(encoded_signal, i, max_rand, 0);
    [SNR_exp, SNR_th] = SNR(rand_sig, quantized_signal, i, 0, max_rand);
    SNR_exp_arr(i-1) = SNR_exp;
    SNR_th_arr(i-1) = SNR_th;
end
figure(4);
subplot(1,1,1);
n = 2:1:8;
plot(n,mag2db(SNR_exp_arr),n,mag2db(SNR_th_arr),'--')
title('SNR non uniform signal');
xlabel 'number of bits';
ylabel 'SNR';
legend({'experimental','theoretical'},'Location','southeast')
grid on


% req6: generate a non-uniform random signal and test uniform quantizer
% with different u
size = [1 1000];
exp_signal = exprnd(1,size);
sin = (randi([0,1],size)*2)-1;
rand_sig = exp_signal.*sin;
u = [0 5 100 200];
figure(5);
subplot(1,1,1);
n = 2:1:8;
colors = ['r','m','k','b'];
index = 1;
hold on
for ele = u
    SNR_exp_arr = zeros(7);
    SNR_th_arr = zeros(7);
    y = compressor_block(rand_sig, ele, sin);
    for i=2:8
        encoded_signal = UniformQuantizer(y, i, max(abs(y)), 0);
        quantized_signal = UniformDequantizer(encoded_signal, i, max(abs(y)), 0);
        expanded = expansion_block(quantized_signal, ele, sin);
        if(ele > 0) % we normalized the function in compressor
            expanded = expanded*max(abs(rand_sig));
        end
        [SNR_exp, SNR_th] = SNR(rand_sig, expanded, i, ele, max(abs(rand_sig)));
        SNR_exp_arr(i-1) = SNR_exp;
        SNR_th_arr(i-1) = SNR_th;
    end
    plot(n,mag2db(SNR_exp_arr),colors(index),n,mag2db(SNR_th_arr),strcat(colors(index),'--
'));
    index = index +1;
```

```
end
title('non uniform quantizer');
xlabel 'number of bits';
ylabel 'SNR';
legend({'u=0 th','u=0 exp','u=5 th','u=5 exp','u=100 th','u=100 exp','u=200 th','u=200
exp'},'Location','southeast')
grid on
hold off


function q_ind = UniformQuantizer(in_val, n_bits, xmax, m)
    delta=(2*xmax)/(2^n_bits);
    xmin = m*(delta/2)-xmax;

    % change values less than xmin to meet xmin in case of mid-tread
    internal_val = xmin .* ones(1,length(in_val));
    cond = (in_val >= xmin);
    internal_val(cond) = in_val(cond);

    % get the level of each point in the signal
    q_ind = min(floor((internal_val - xmin)/delta)+1 , (2^n_bits));
end

function deq_val = UniformDequantizer(in_val, n_bits, xmax, m)
    delta=(2*xmax)/(2^n_bits);
    x_min = xmax - ((1 - m) * 0.5*delta);
    deq_val = (in_val - (1 - m)) * delta - x_min;
end


function [SNR_exp, SNR_th] = SNR(signal, quantized, n_bits, u, xmax)
    quantization_error = signal - quantized;
    input_power = mean(signal.^2);
    SNR_exp = input_power / mean(quantization_error.^2);
    if(u > 0)
        SNR_th = (3 * (2^n_bits)^2) / (log(1 + u)^2);
    else
        SNR_th = (3 * (2^n_bits)^2 * input_power) / xmax^2;
    end
end


function compressed = compressor_block(x, u, sin)
    compressed = x;
    if(u>0)
        compressed = sin .* (log(1+u*abs(x/max(x)))/log(1+u));
    end
end


function expanded = expansion_block(x, u, sin)
    expanded = x;
    if(u>0)
        % inverse of compressed function
        expanded= sin .*(((1+u).^abs(x)-1)/u);
    end
end
```