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

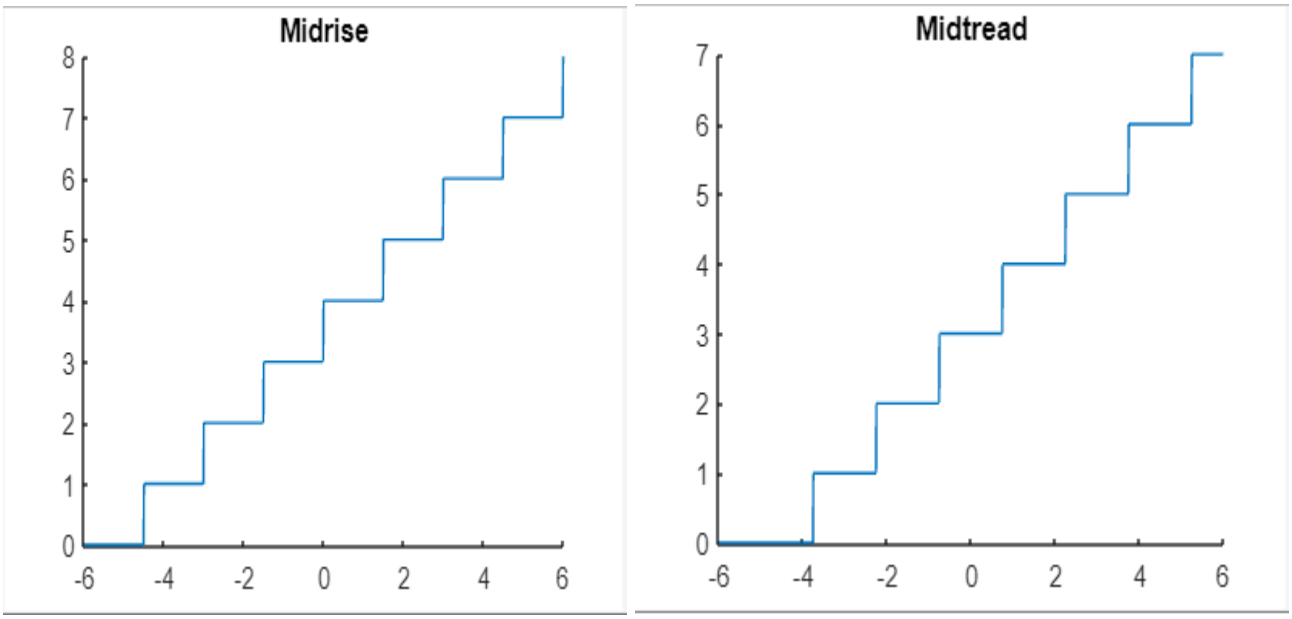
Submitted by

Name	Sec	BN
Asmaa Adel Abdelhamed	1	14
Samaa Hazem Mohamed	1	32

Contents

Figures

Part 1:.....



Quantization function output

Comment:

In this part we create a quantization function which calculates the indices of the input signal.

First, we calculate delta (Δ) which consists of the step of each level, then we calculate the index of each level.

m indicates if it is midtread or midrize, if m = 0, it's midrize, else if m = 1 it's midtread.

The function returns the indices, then when we call it, we are able to plot the indices with the actual signal.

In the figure, it's the output of the function in two cases, midrize and midtread, where x in x-axis, indices in y-axis.

In midrize, The quantization levels in this type are even in number, the origin lies in the middle of a rising part of the staircase- like graph.

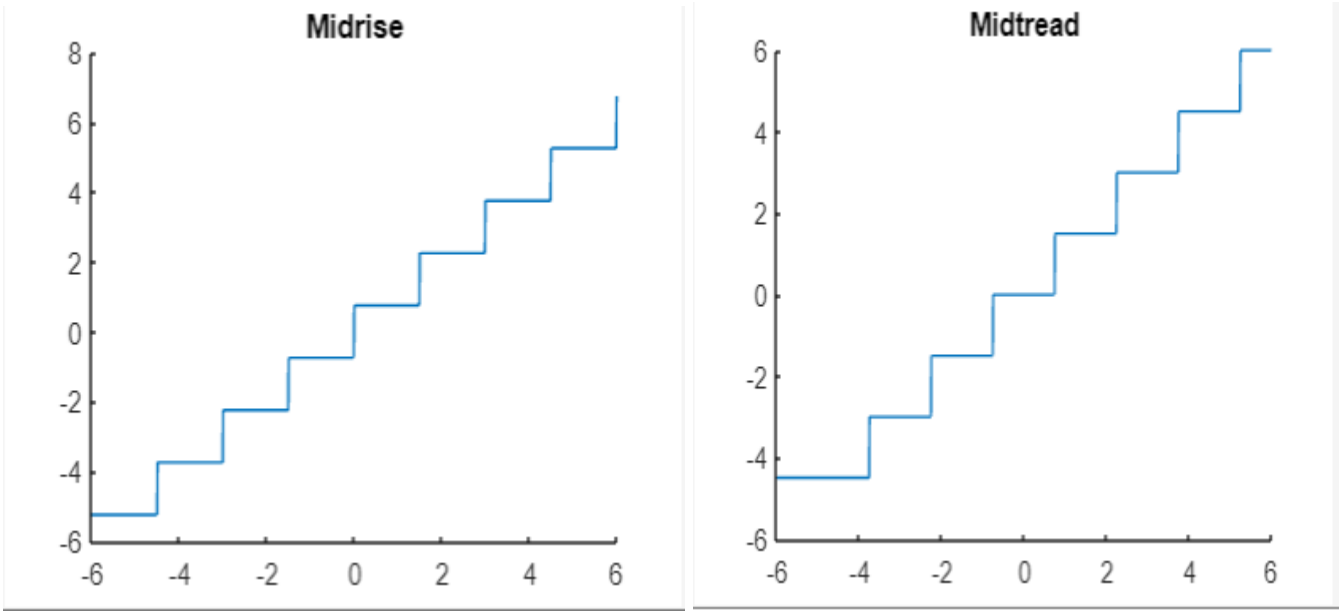
In midtread, The quantization levels in this type are odd in number, the origin lies in the middle of a tread of the staircase- like graph.

The output of the functions is between 0 and number of levels - 1

Our Function for quantization:

```
= floor((in_val - ((m) * (delta / 2) - xmax)) / delta)
```

Part 2:……



De-quantization function output

Comment:

In this part we create a de-quantization function which calculates the dequantized value for the input signal.

First, we calculate delta (Δ) which consists of the step of each level, then we calculate the dequantized value.

m indicates if it is midtread or midrize, if m = 0, it's midrize, else if m = 1 it's midtread.

The function returns the dequantized value, then when we call it, we are able to plot the dequantized value with the actual signal.

In the figure, it's the output of the function in two cases, midrize and midtread, where x in x-axis, dequantized value in y-axis.

In midrize, The quantization levels in this type are even in number, the origin lies in the middle of a rising part of the staircase- like graph.

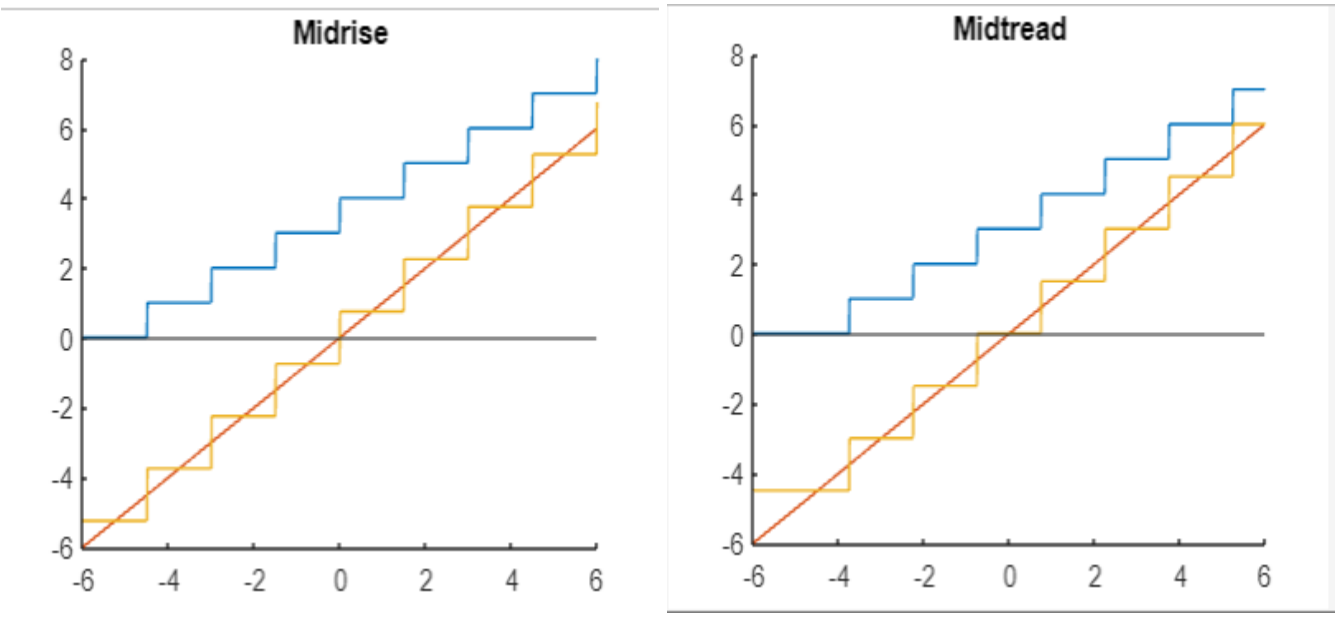
In midtread, The quantization levels in this type are odd in number, the origin lies in the middle of a tread of the staircase- like graph.

The output of the functions is between the two values of input signal.

Our Function for de-quantization:

```
= ((q_ind) * delta) + ((m + 1) * (delta / 2) - xmax)
```

Part 3:……



Test the quantizer/de-quantizer

Comment:

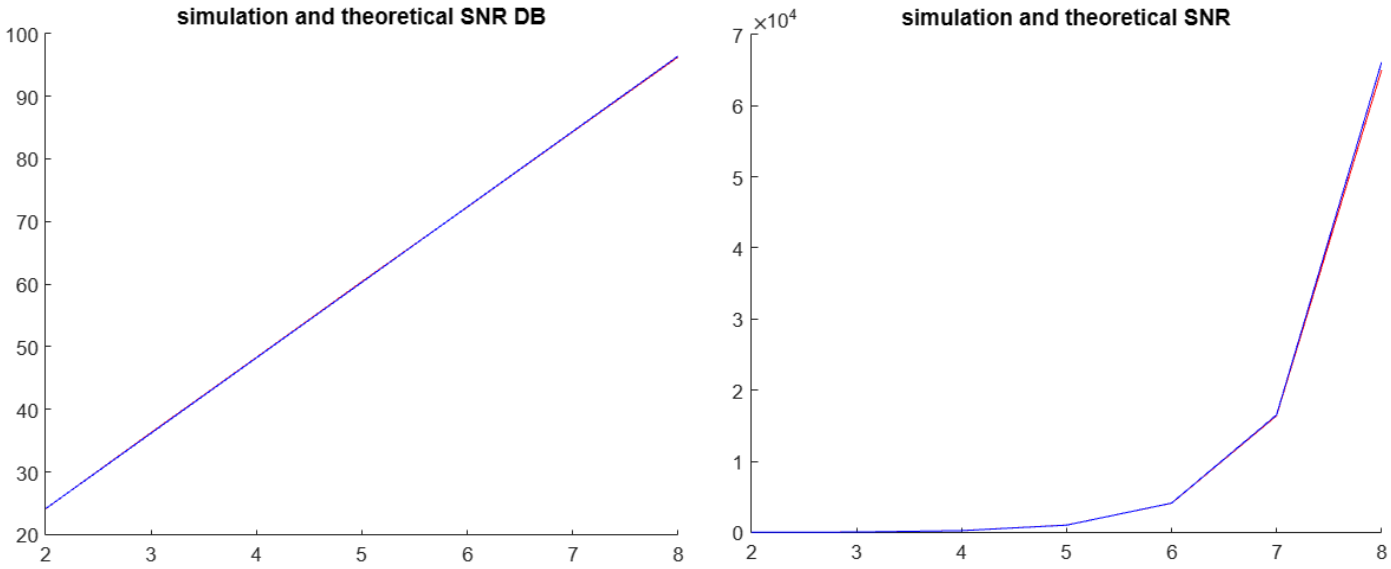
As we know, Quantization is the process of mapping input values (magnitude of signal) from a large set (often a continuous set) to output values in a (countable) smaller set. Rounding and truncation are typical examples of quantization processes.

To test our functions, We generate input signal X from -6 to 6 with step size 0.01, then we pass it to quantization, and de-quantization functions, and put n_bits = 3, xmax = 6

then we plot it for m = 0, m = 1.

we expected that, the midrise uniform quantizers do not have a zero output value – their minimum output magnitude is half the step size, but the mid-tread quantizers do have a zero output level, and can reach arbitrarily low bit rates per sample for input distributions that are symmetric and taper off at higher magnitudes.

Part 4:.....



SNR_SIM_dB –> 'red'

SNR_THER_dB –> 'blue'

Figure 4 Fig

Comment:

NOTE : for req4&5&6 ,the code has the commands for plotting SNR before it gets its db form. I have commented on them just for simplicity of tracking the outputs as there will not be many figures.

From this requirement to the end of the requirements , we are working to get the SNR theoretical and SNR from simulation, getting their values for 7-values to the number of bits (n_bits) used in quantization,then comparing their values in db.

Using the following rule from the lecture for SNR theoretical:

$$SNR = \frac{\widetilde{m^2}}{N_q} = \frac{3L^2}{m_p^2}P$$

and using this definition of error to calculate SNR of simulation:

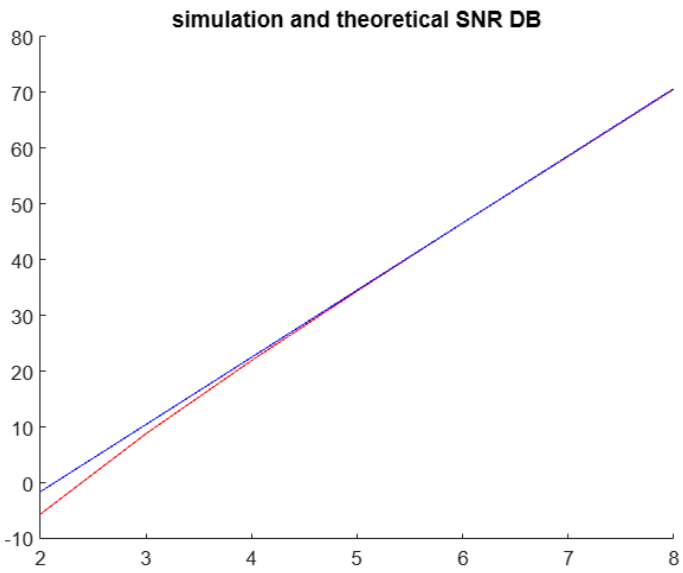
$$q = m - \nu$$

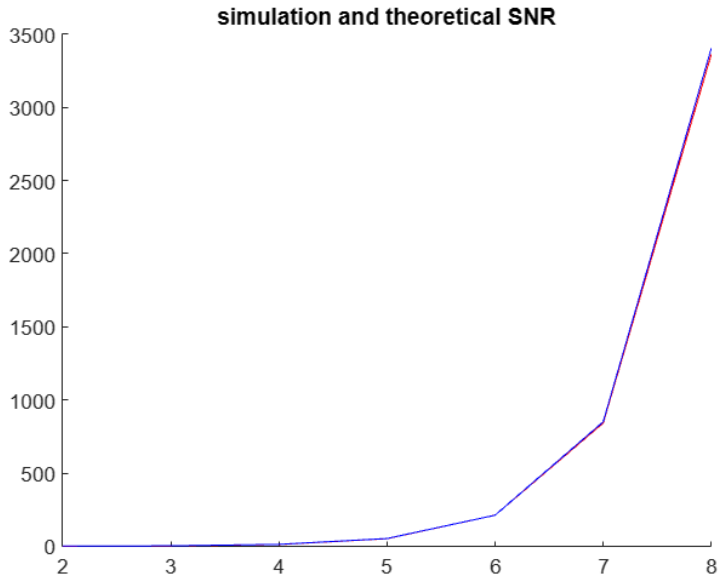
q : quantization error

```
SNR_SIM(i) = mean(x.^2)/mean(error.^2)
```

As we notice in the plots above , they almost have the same values.

Part 5:……






```
SNR_SIM_dB –> 'red'
SNR_THER_dB –> 'blue'
```

Figure 5 Fig

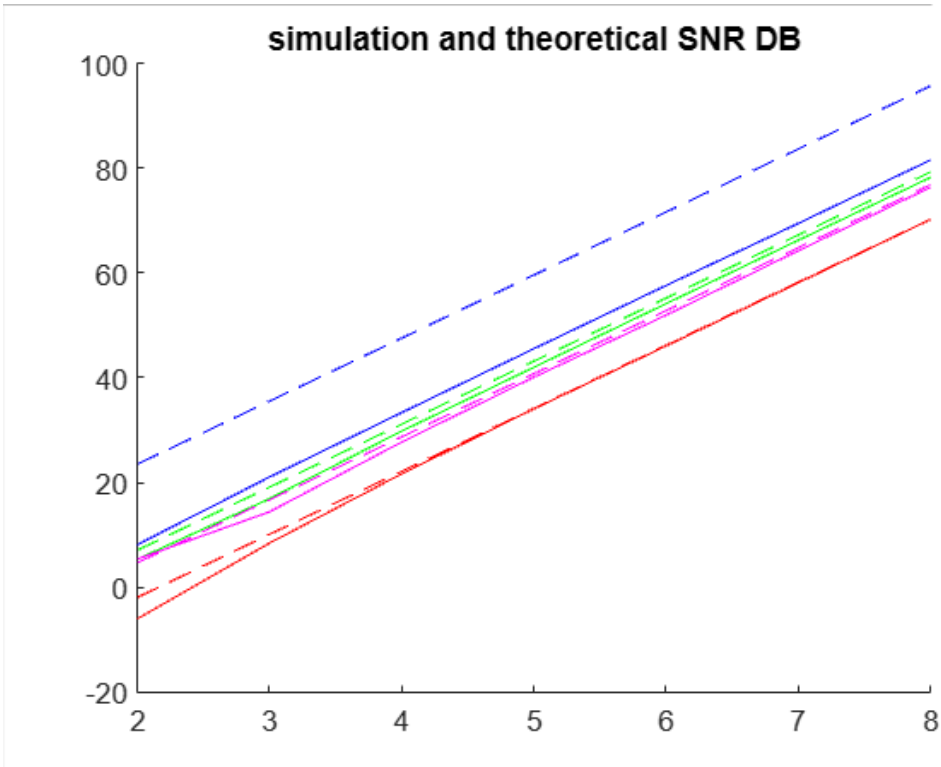
Comment:

Using the same rules as requirement 4 , we are trying to perform uniform quantization for non-uniform exponential signal , we notice a difference between SNR theoretical and SNR from simulation,and it is getting reduced with increasing the value of number of bits used in quantization

Noting that, for generating random sign with equal distribution we use:

```
M=1;
N=10000;
random_sign_1 = mod( reshape(randperm(M*N), M, N), 2 );
```

Part 6:……

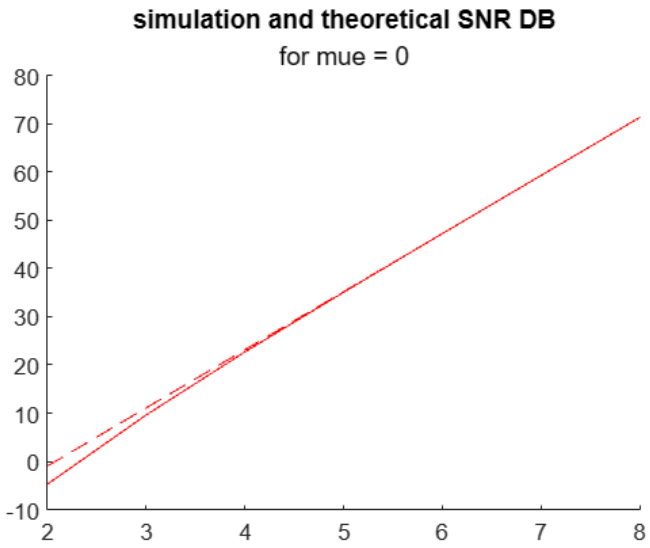


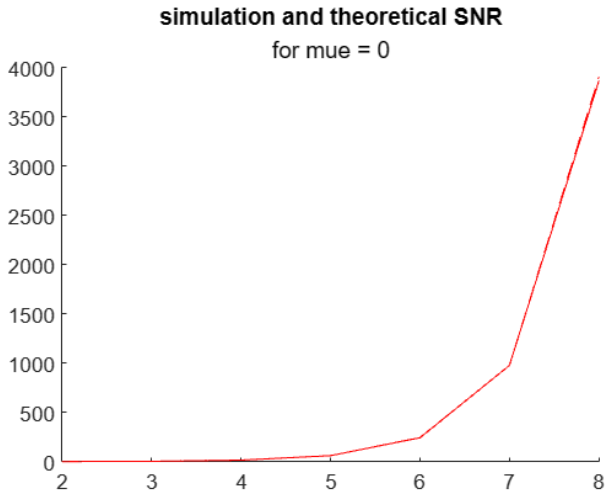
μ = 0 → `'red'`

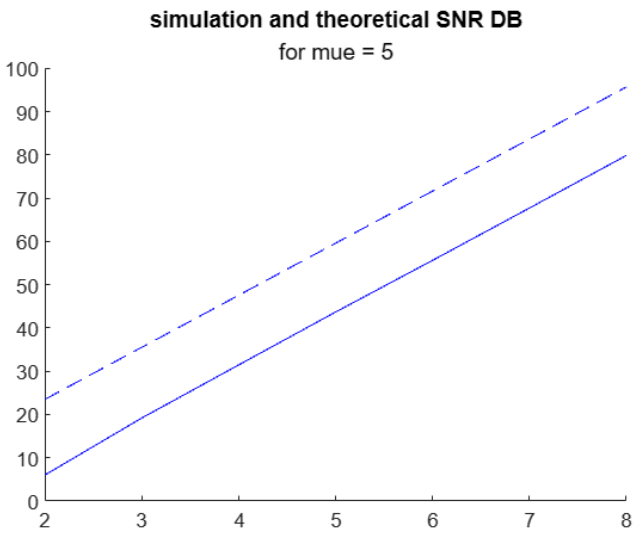
μ = 5 → `'blue'`

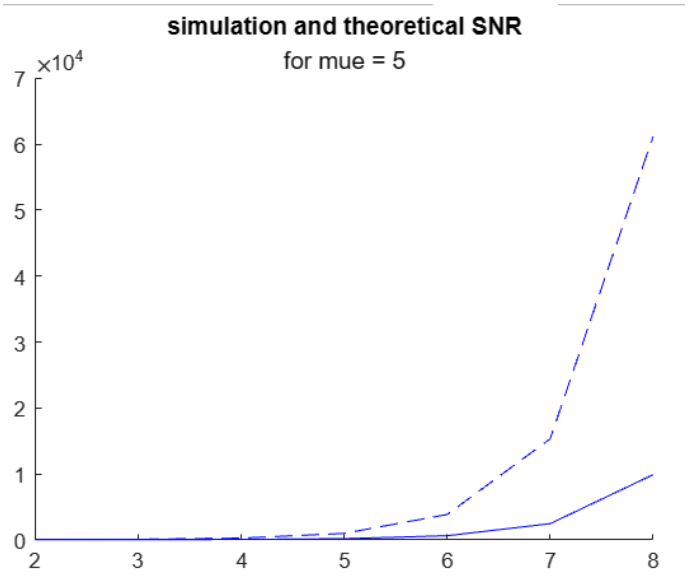
μ = 100 → `'green'`

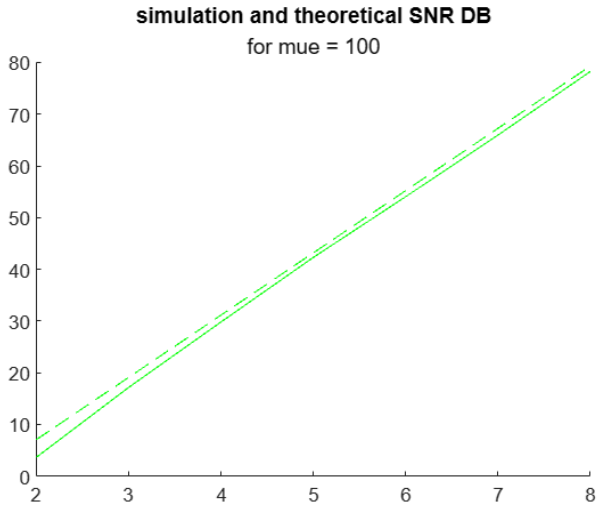
μ = 200 → `'magenta'`

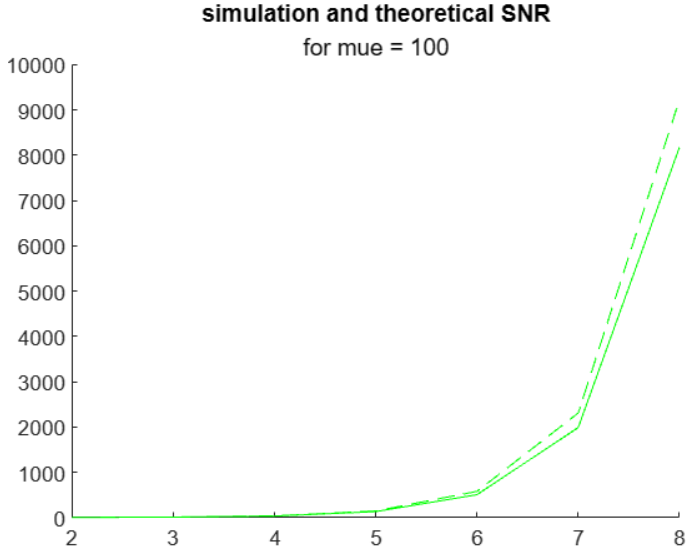


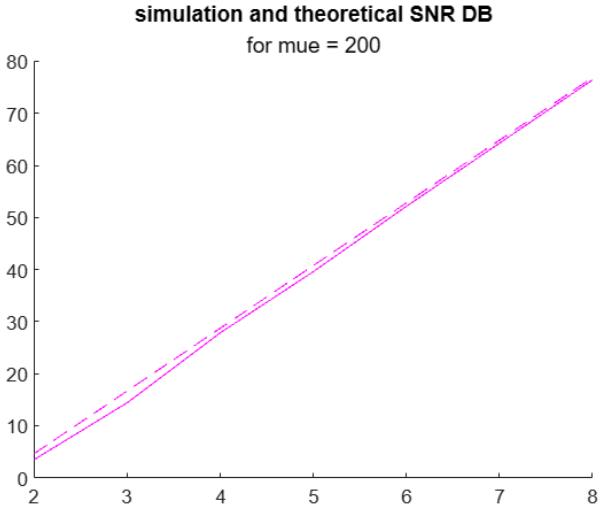



simulation and theoretical SNR DB
for mue = 5


simulation and theoretical SNR
for mue = 5
×10^4


simulation and theoretical SNR DB
for mue = 100


simulation and theoretical SNR
for mue = 100


simulation and theoretical SNR DB
for mue = 200

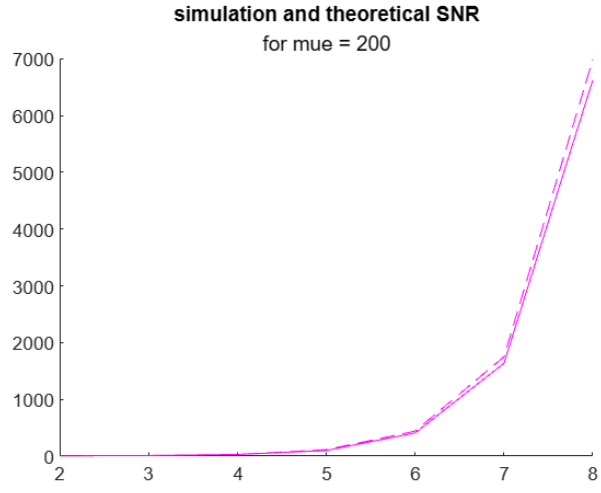

simulation and theoretical SNR
for mue = 200

Figure 6 Fig

Comment:

Compress → Quantize → Dequantize → Expand

Using the same concepts of calculating SNR from simulation, and using this rule from lecture to calculate SNR theoretical:

$$SNR \simeq \frac{3L^2}{[\ln(1+\mu)]^2}$$

we are trying to use our uniform quantizer to quantize the non-uniform signal in requirement 5 for making better result for SNR simulation, so we have compressed the signal before quantization the expand it after the dequantization,using the μ-Law concept , with different μ values.

Given that for each value of μ,we use 7-values to the number of bits (n_bits).

We notice that increasing the value of the μ , decrease the difference between SNR theoretical and SNR from simulation,also increasing the value of n_bits for the sane value of μ , decrease the difference between them.

Noting that: for μ=0 , it gives the same result as requirement 5, as the main equations of μ-law has (ln(1+μ))in the denominator, instead of adding some conditions before those divisions ,we could just start from μ=5 and delete all thse if-conditions, and take the results from requirement 5 to plot it with all μ result

Index:

NOTE: it is not the actual order of the code , as the defined functions have to be at the end , we just reorder it to specify which code for which requirement.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 1- UniformQuantizer Function%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function q_ind = UniformQuantizer(in_val, n_bits, xmax, m)
   delta = deltaFunction(n_bits,xmax);
   q_ind = floor((in_val - ((m) * (delta / 2) - xmax)) / delta);
   q_ind(q_ind < 0) = 0;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 2- UniformDequantizer Function%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function deq_val = UniformDequantizer(q_ind, n_bits, xmax, m)
   delta = deltaFunction(n_bits,xmax);
   deq_val = ((q_ind) * delta) + ((m + 1) * (delta / 2) - xmax);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% deltaFunction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function delta = deltaFunction(n_bits,xmax)
num_levels = 2 ^ n_bits; %2** n_bits;
delta = 2*xmax /num_levels;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 3- Test the quantizer/dequantizer %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x = -6:0.01:6;
n_bits= 3;
xmax = 6;
m = 1;
% midtread
q_indcall1 = UniformQuantizer(x, n_bits, xmax, m);
```

```
deq_valcall1 = UniformDequantizer(q_indcall1, n_bits, xmax, m);
figure();
title("Midtread")
hold on
plot( x, q_indcall1 );
plot( x, x );
plot( x, deq_valcall1 );
yline(0);
%axis([-8 8 -8 8])
hold off
m = 0;
% midrise
q_indcall2 = UniformQuantizer(x, n_bits, xmax, m);
deq_valcall2 = UniformDequantizer(q_indcall2, n_bits, xmax, m);
figure();
title("Midrise")
hold on
plot( x, q_indcall2 );
plot( x, x );
yline(0);
plot( x, deq_valcall2 );
%axis([-8 8 -8 8])
%axis padded
hold off
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% calc4_5 for req 4 and 5
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function SNR_S_T = calc4_5(x, xmax, m)
SNR_S_T = zeros(2, 7);
n_bits = 2:1:8;
SNR_SIM = zeros(1, 7);
SNR_SIM_dB = zeros(1, 7);
SNR_THER = zeros(1, 7);
SNR_THER_dB = zeros(1, 7);
```

```
for i = 1:7
   q_indcall3 = UniformQuantizer(x, n_bits(i), xmax, m);
   deq_valcall3 = UniformDequantizer(q_indcall3, n_bits(i), xmax, m);
   error = deq_valcall3 - x;
   SNR_SIM(i) = mean(x.^2)/mean(error.^2);
   SNR_SIM_dB(i) = db(SNR_SIM(i));
   num_levels = 2 ^ n_bits(i);
   SNR_THER(i) = (mean(x.^2)*(3* num_levels^2 / xmax^2));
   SNR_THER_dB(i) = db(SNR_THER(i));

end
figure();
title("simulation and theoretical SNR DB")
hold on
plot(n_bits, SNR_SIM_dB,'r');
plot(n_bits, SNR_THER_dB,'b');
hold off
SNR_S_T(1,:) = SNR_SIM_dB;
SNR_S_T(2,:) = SNR_THER_dB;
% figure();
% title("simulation and theoretical SNR")
% hold on
% plot(n_bits, SNR_SIM,'r');
% plot(n_bits, SNR_THER,'b');
% hold off
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  4- test your input on a random input signal  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x = -5 + (5+5)*rand(10000,1);
xmax = 5;
m = 0;
calc4_5(x, xmax, m);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  5- test your input on a random input signal  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
exp_signal = exprnd(3,[1 10000]);
M=1;
N=10000;
random_sign_1 = mod( reshape(randperm(M*N), M, N), 2 );
random_sign = random_sign_1*2 - 1;
req_exp = random_sign.*exp_signal;
xmax = max(abs(req_exp));
m= 0;
SNR_S_T = calc4_5(req_exp, xmax, m);
signal_x = mean(req_exp.^2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%6- Now quantize the the non-uniform signal using a non-uniform
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
exp_signal = exprnd(3,[1 10000]);
M=1;
N=10000;
random_sign_1 = mod( reshape(randperm(M*N), M, N), 2 );
random_sign = random_sign_1*2 - 1;
req_exp = random_sign.*exp_signal;
xmax = max(abs(req_exp));
m= 0;
req_exp_normalized = req_exp / xmax;
mue = [5;100;200];
n_bits = 2:1:8;
SNR_SIM = zeros(1, 7);
SNR_SIM_dB = zeros(1, 7);
SNR_THER = zeros(1, 7);
SNR_THER_dB = zeros(1, 7);
colors_sim = ['b','g','m'];
colors_ther = ["--b","--g","--m"];
figure();
```

```
title("simulation and theoretical SNR DB")
hold on
plot(n_bits, SNR_S_T(1,:),'r');
plot(n_bits, SNR_S_T(2,:),'--r');
for k=1:numel(mue)
   compress = random_sign.*(log(1 + mue(k)*abs(req_exp_normalized))/log(1 + mue(k)));
   compress_max = max(abs(compress));
   for i = 1:7
       q_indcall6 = UniformQuantizer(compress, n_bits(i), compress_max, m);
       deq_valcall6 = UniformDequantizer(q_indcall6, n_bits(i), compress_max, m);
       expanded = random_sign .*(((1+mue(k)).^abs(deq_valcall6)-1)/mue(k));
       expanded_denormalized = expanded * xmax;
       error = expanded_denormalized - req_exp ;
       SNR_SIM(i) = mean(req_exp.^2)/mean(error.^2);
       SNR_SIM_dB(i) = db(SNR_SIM(i));

       num_levels = 2 ^ n_bits(i);
       SNR_THER(i) = 3*(num_levels)^2 / (log(1+mue(k)))^2 ;
       SNR_THER_dB(i) = db(SNR_THER(i));
   end
% figure();
% title("simulation and theoretical SNR DB","for mue = "+mue(k))
% hold on
plot(n_bits, SNR_SIM_dB,colors_sim(k));
plot(n_bits, SNR_THER_dB,colors_ther(k));
% hold off
% figure();
% title("simulation and theoretical SNR","for mue = "+mue(k))
% hold on
% plot(n_bits, SNR_SIM,colors_sim(k));
% plot(n_bits, SNR_THER,colors_ther(k));
% hold off
end
hold off
```