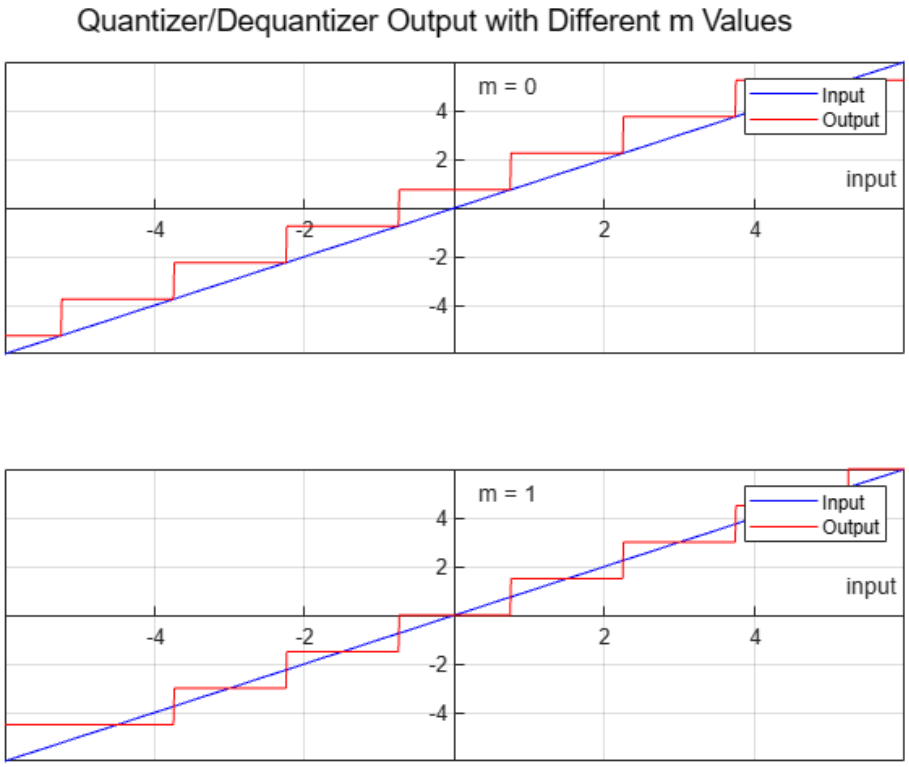
ASSIGNMENT 1

By:

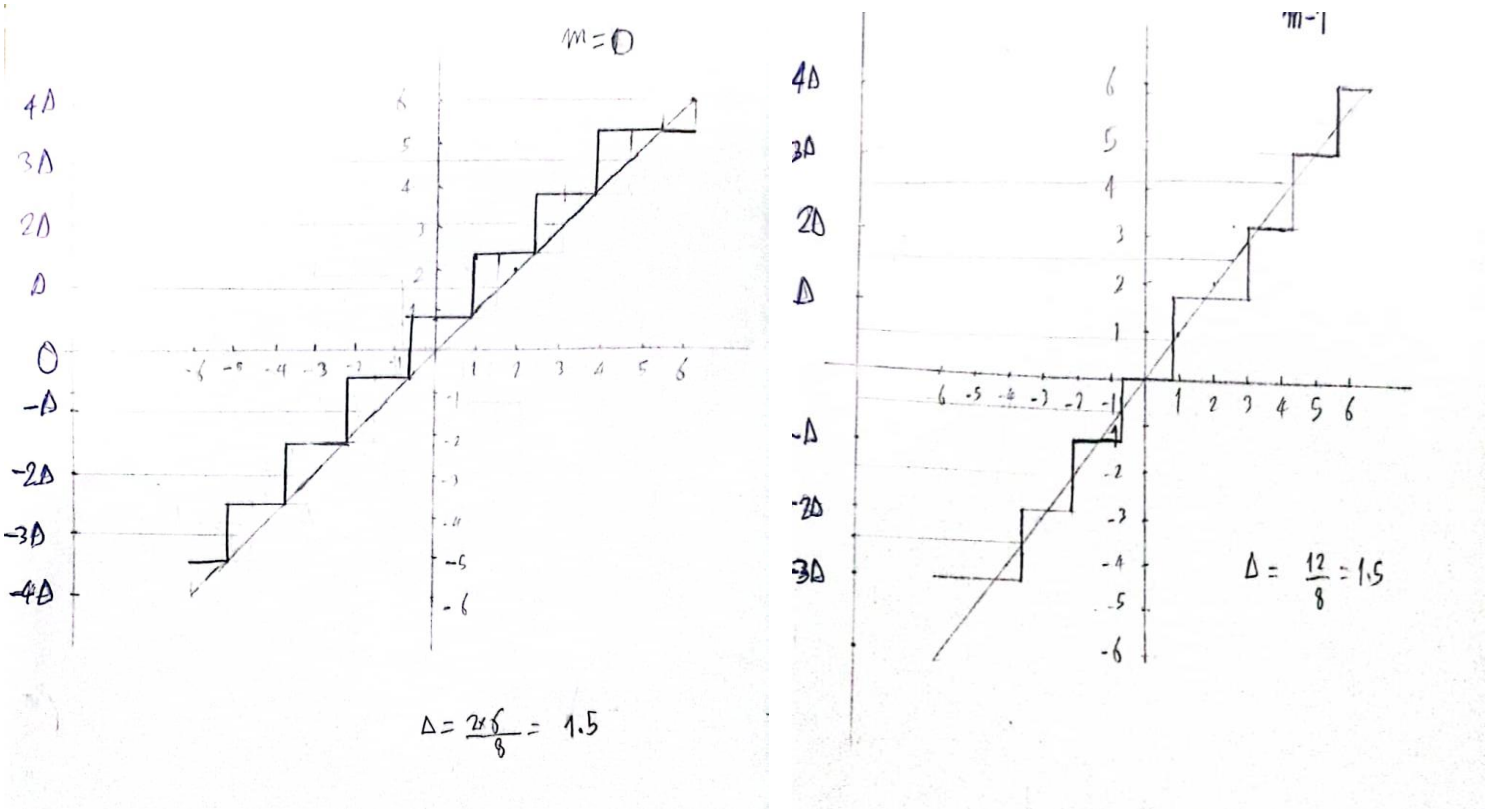
Omar Said ➔ Sec: 1 Bn: 27

Ahmed Samy ➔ Sec: 1 Bn: 8

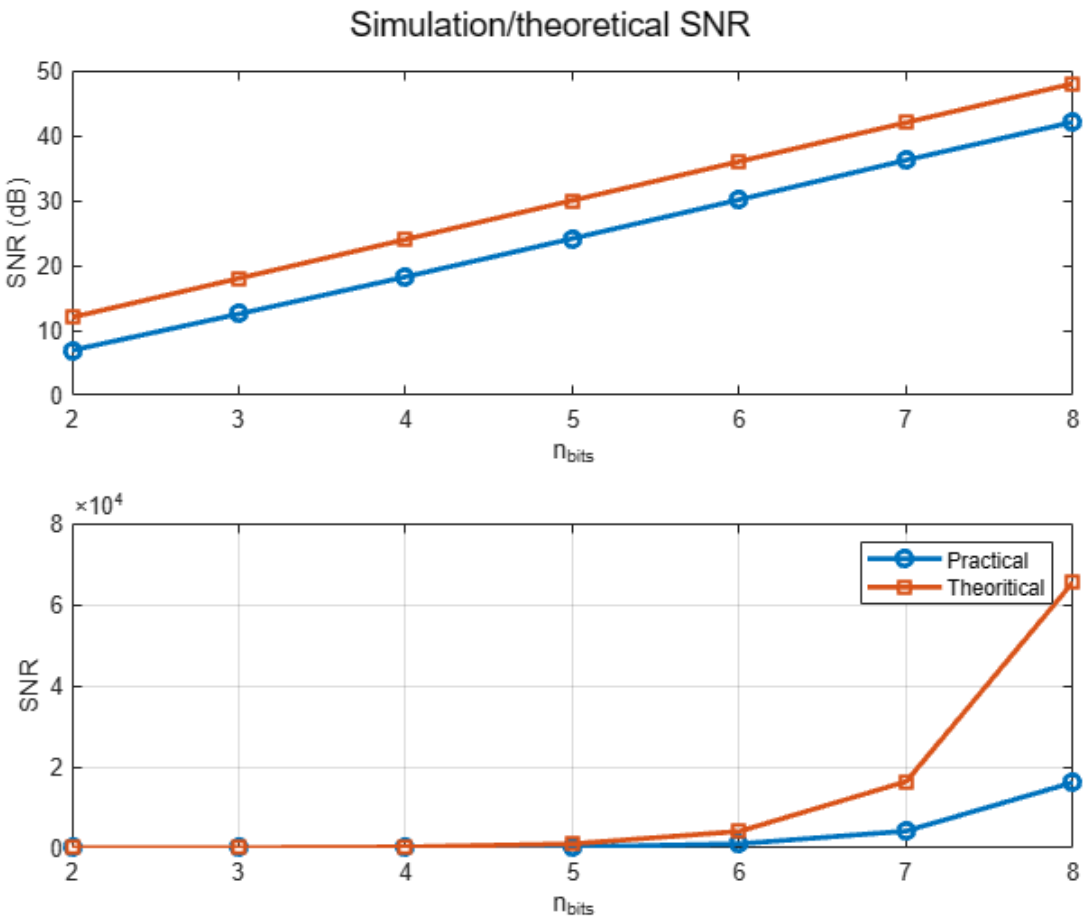
Third Question:



Hand Analysis:

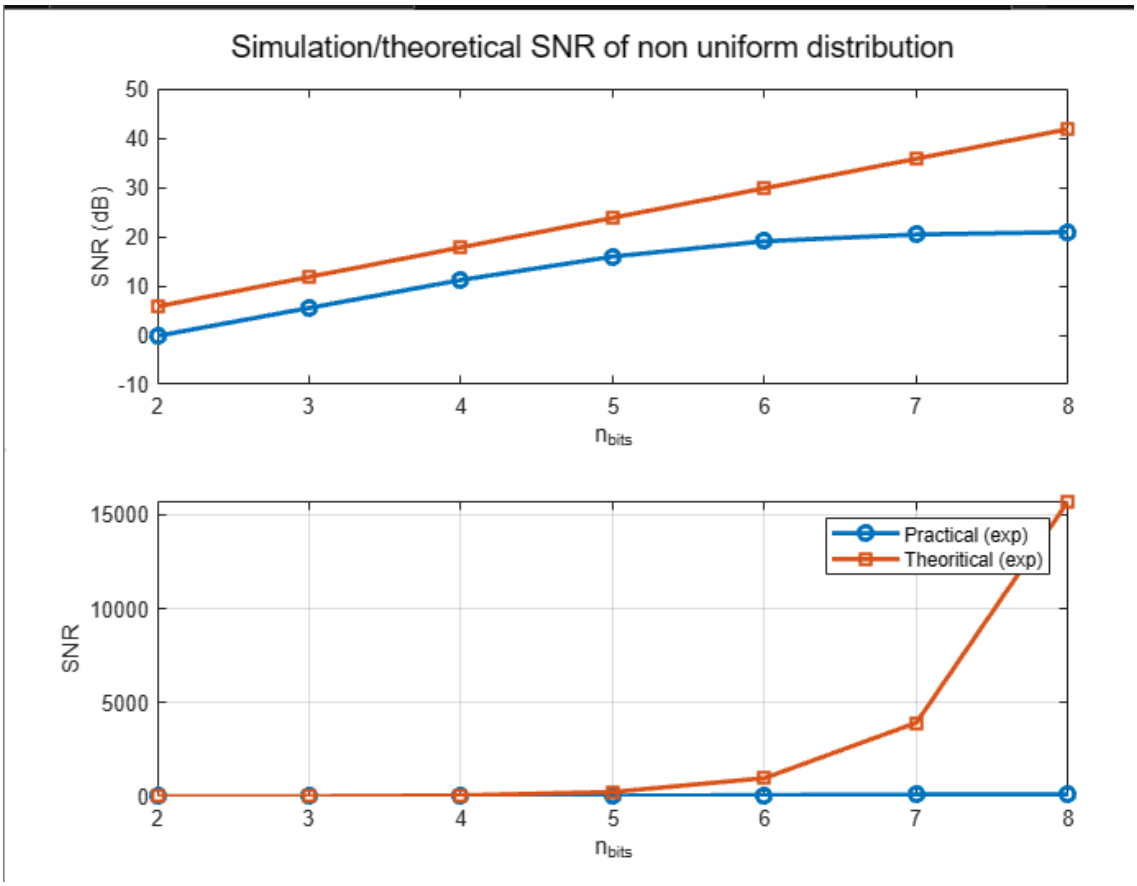


Fourth Question:



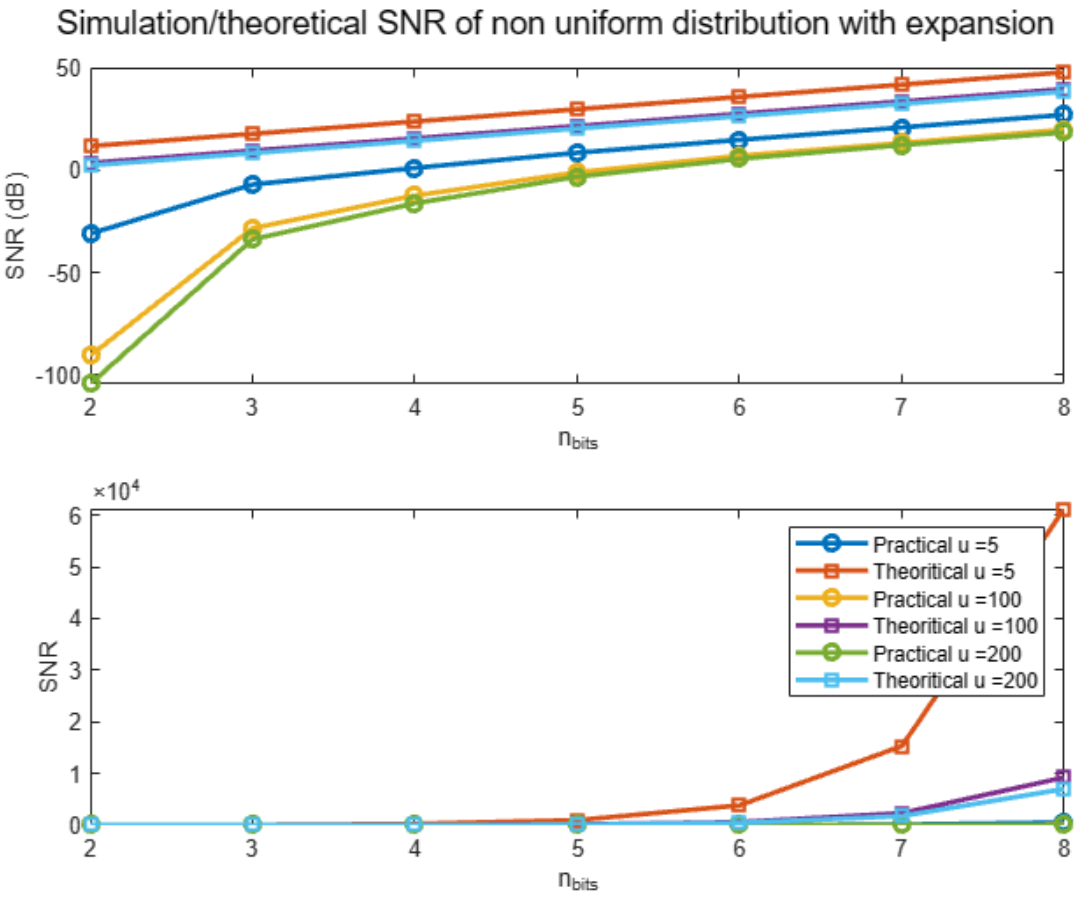
SNR theoretical vs practical for input ramp

Fifth Question:



SNR theoretical vs practical for non-uniform signal input using uniform quantizer

Sixth Question:



SNR theoretical vs practical for non-uniform signal input using non-uniform quantizer

Code:

```
clear all;
close all;
clc;
% Test 1
x = -6:0.01:6;
n_bits = 3;
xmax = 6;
q_ind1 = UniformQuantizer(x, n_bits, xmax, 0);
deq_val1=UniformDequantizer(q_ind1,n_bits,xmax,0);
q_ind2 = UniformQuantizer(x, n_bits, xmax, 1);
deq_val2=UniformDequantizer(q_ind2,n_bits,xmax,1);
fig1=figure;
subplot(2, 1, 1);
plot(x, x, 'b', x, deq_val1, 'r');
xlabel('input');
ylabel(' m = 0');
legend('Input', 'Output');
grid on;
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
subplot(2, 1, 2);
plot(x, x, 'b', x, deq_val2, 'r');
xlabel('input');
ylabel(' m = 1');
legend('Input', 'Output');
grid on;
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
sgtitle('Quantizer/Dequantizer Output with Different m Values');
set(fig1, 'Name', 'Quantizer/Dequantizer Output');

% Test 2
lower_bound = -5;
upper_bound = 5;
num_samples = 10000;
uniform_samples = rand(1, num_samples);
uniform_variables = lower_bound + (upper_bound - lower_bound) * uniform_samples;
quant_error = zeros(7, num_samples);
n_bits_range=2:1:8;
for idx = 1:length(n_bits_range)
   n_bits = n_bits_range(idx);
   q_ind=UniformQuantizer(uniform_variables, n_bits, 5, 0);
   q_deq=UniformDequantizer(q_ind, n_bits, 5, 0);
   quant_error(n_bits-1,:)=uniform_variables-q_deq;
end
input_squared_mean = mean(uniform_variables.^2);
quant_error_squared_mean = mean(quant_error.^2, 2); % Calculate mean along rows
SNR_pract = input_squared_mean ./ quant_error_squared_mean;
SNR_pract_db=10 * log10(SNR_pract);
SNR_theor_db=6*n_bits_range;
SNR_theor=2.^(2*n_bits_range);
fig2=figure;
subplot(2,1,1);
plot(n_bits_range, SNR_pract_db, 'o-', 'LineWidth', 2, 'DisplayName', 'Simulation');
hold on;
plot(n_bits_range, SNR_theor_db, 's-', 'LineWidth', 2, 'DisplayName', 'Theory');
hold off;
xlabel('n_{bits}');
ylabel('SNR (dB)');
subplot(2,1,2);
plot(n_bits_range, SNR_pract, 'o-', 'LineWidth', 2, 'DisplayName', 'Simulation');
hold on;
plot(n_bits_range, SNR_theor, 's-', 'LineWidth', 2, 'DisplayName', 'Theory');
hold off;
xlabel('n_{bits}');
ylabel('SNR');
sgtitle('Simulation/theoretical SNR');
set(fig2, 'Name', 'Simulation/theoretical SNR');
legend('Practical', 'Theoritical');
grid on;
```

```
% 5

% Generate random polarities (+/- with equal probability)
polarity = rand(1, num_samples) > 0.5;
polarity = 2*polarity - 1; % Convert logical array to +/- 1 values
% Generate magnitudes from exponential distribution
magnitudes = exprnd(1, 1, num_samples);
% Combine polarity and magnitude
samples = polarity .* magnitudes;
% Calculate the energy of the input signal
input_squared_mean = mean(samples.^2);
quant_error = zeros(7, num_samples);
for idx = 1:length(n_bits_range)
   n_bits = n_bits_range(idx);
   q_ind=UniformQuantizer(samples, n_bits, 5, 0);
   q_deq=UniformDequantizer(q_ind, n_bits, 5, 0);
   quant_error(n_bits-1,:)=samples-q_deq;
end
quant_error_squared_mean = mean(quant_error.^2, 2);
SNR_pract = input_squared_mean ./ quant_error_squared_mean;
SNR_pract_db=10 * log10(SNR_pract);
% Var(XY)=E(X2^Y^2)-(E(XY))^2=Var(X)Var(Y)+Var(X)(E(Y))^2+Var(Y)(E(X))^2 as
% they are independent
SNR_theor=2.^(2*n_bits_range)*0.24;
SNR_theor_db=10 * log10(0.24)+6*n_bits_range;

fig3=figure;
subplot(2,1,1);
plot(n_bits_range, SNR_pract_db, 'o-', 'LineWidth', 2, 'DisplayName', 'Simulation');
hold on;
plot(n_bits_range, SNR_theor_db, 's-', 'LineWidth', 2, 'DisplayName', 'Theory');
hold off;
xlabel('n_{bits}');
ylabel('SNR (dB)');
subplot(2,1,2);
plot(n_bits_range, SNR_pract, 'o-', 'LineWidth', 2, 'DisplayName', 'Simulation');
hold on;
plot(n_bits_range, SNR_theor, 's-', 'LineWidth', 2, 'DisplayName', 'Theory');
hold off;
xlabel('n_{bits}');
ylabel('SNR');
sgtitle('Simulation/theoretical SNR of non uniform distribution');
set(fig3, 'Name', 'Simulation/theoretical SNR of non uniform distribution');
legend('Practical (exp)', 'Theoritical (exp)');
grid on;
```

```
% 6

% TODO: is n const or changed
fig4=figure;
grid on;
mius = [5, 100, 200];
for idx = 1:length(mius)
   miu = mius(idx);
   quant_error = zeros(7, num_samples);
   for idx2 = 1:length(n_bits_range)
        n_bits=n_bits_range(idx2);
        compressed_vals = Compressor(samples, miu);
        q_ind = UniformQuantizer(compressed_vals, n_bits, 5, 0);
        q_deq = UniformDequantizer(q_ind, n_bits, 5, 0);
        expanded_vals = Expander(q_deq, miu);
        quant_error(n_bits-1, :) = samples - expanded_vals;

   end
   quant_error_squared_mean = mean(quant_error.^2, 2);
   SNR_pract = input_squared_mean ./ quant_error_squared_mean;
   SNR_pract_db=10 * log10(SNR_pract);
   SNR_theor=2.^(2*n_bits_range)*3/((log(1+miu))^2);
   SNR_theor_db=10 * log10(3)+6*n_bits_range-10*log10((log(1+miu))^2);
   subplot(2,1,1);
   plot(n_bits_range, SNR_pract_db, 'o-', 'LineWidth', 2, 'DisplayName', 'Simulation');
   hold on;
   plot(n_bits_range, SNR_theor_db, 's-', 'LineWidth', 2, 'DisplayName', 'Theory');
   hold on;
   xlabel('n_{bits}');
   ylabel('SNR (dB)');
   subplot(2,1,2);
   plot(n_bits_range, SNR_pract, 'o-', 'LineWidth', 2, 'DisplayName', 'Simulation');
   hold on;
   plot(n_bits_range, SNR_theor, 's-', 'LineWidth', 2, 'DisplayName', 'Theory');
   hold on;
   xlabel('n_{bits}');
   ylabel('SNR');
end
legend('Practical u =5', 'Theoritical u =5','Practical u =100', 'Theoritical u =100','Practical u =200',
'Theoritical u =200');
sgtitle('Simulation/theoretical SNR of non uniform distribution with expansion');
set(fig4, 'Name', 'Simulation/theoretical SNR of non uniform distribution');
function q_ind = UniformQuantizer(in_val, n_bits, xmax, m)
     L = 2^n_bits; % Number of quantization intervals
     delta = 2 * xmax / L; % Width of each quantization interval
     q_levels_output = ((1-m) * ((-L+1)* delta / 2) + (m*(-L*0.5+1)*delta)):delta:((1-m) * ((L-1)* delta
/ 2) + (m*L*0.5*delta));
     rounded_in_val = round(in_val / delta);
     midrise_out = (1 - m) * (((rounded_in_val + 0.5) * delta) + m);
     mid_tread_out = m * (rounded_in_val) * delta;
     out_val = mid_tread_out + midrise_out;
      % Ensure out_val is within the range of q_levels_output
     out_val(out_val < q_levels_output(1)) = q_levels_output(1);
     out_val(out_val > q_levels_output(L)) = q_levels_output(L);
     [~, indices] = ismember(out_val, q_levels_output); % Find indices of matching values
     q_ind = indices - 1; % Convert to index - 1
     q_ind(indices == 0) = NaN; % Handle values not found in q_levels_output
end

function deq_val=UniformDequantizer(q_ind,n_bits,xmax,m)
     L = 2^n_bits; % Number of quantization intervals
     delta = 2 * xmax / L; % Width of each quantization interval
     q_levels_output = ((1-m) * ((-L+1)* delta / 2) + (m*(-L*0.5+1)*delta)):delta:((1-m) * ((L-1)* delta
/ 2) + (m*L*0.5*delta));
     deq_val=q_levels_output(q_ind+1);
end
function compressed_vals = Compressor(in,miu)

compressed_vals=sign(in).*(log(1+abs(in)*miu)/log(1+miu));
end

function expanded_vals=Expander(in,miu)

expanded_vals=sign(in).*(exp(abs(in) * log(1 + miu)) - 1) / miu;
end
```