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
Nancy Ayman	2	27
Yara Hisham	2	31

Contents

Figures

Part 1: Implementing a uniform scalar quantizer function

```
function q_ind = UniformQuantizer(in_val, n_bits, xmax, m)
levels = 2 ^ n_bits;
delta = 2 * xmax / levels;
q_ind = floor((in_val - ((m) * (delta / 2) - xmax)) / delta);
q_ind(q_ind<0) = 0;
end
```

Comment:

- **in_val**: Input values to be quantized.
- **n_bits**: Number of bits used for quantization.
- **xmax**: Maximum absolute value of the input signal.
- **m**: Mid-tread or mid-rise quantization parameter. If **m = 0**, it's mid-tread quantization. If **m = 1**, it's mid-rise quantization.

Steps:

1. Calculate the number of quantization levels (**levels**) using the formula **2 ^ n_bits**.
2. Compute the quantization step size (**delta**) using the formula **2 * xmax / levels**.
3. Quantize the input values by subtracting **((m) * (delta / 2) - xmax)** from them, then dividing by **delta**, and taking the floor of the result. This formula ensures that the quantization levels are symmetric around zero.
4. If any quantized index is less than zero (due to rounding errors), it is set to zero.

Part 2: Implementing a uniform scaler de-quantizer function

```
function deq_val = UniformDequantizer(q_ind, n_bits, xmax, m)
levels = 2 ^ n_bits;
delta = 2 * xmax / levels;
deq_val = ((q_ind) * delta) + ((m+1) * (delta / 2) - xmax);
end
```

Comment:

- **q_ind**: Quantized indices obtained from quantization.
- **n_bits**: Number of bits used for quantization.
- **xmax**: Maximum absolute value of the input signal.
- **m**: Mid-tread or mid-rise quantization parameter. If **m = 0**, it's mid-tread quantization. If **m = 1**, it's mid-rise quantization.

Steps:

1. Calculate the number of quantization levels (**levels**) using the formula **2 ^ n_bits**.
2. Compute the quantization step size (**delta**) using the formula **2 * xmax / levels**.
3. Reconstruct the original values by multiplying the quantized indices by **delta**, then adding **((m+1) * (delta / 2) - xmax)**. This operation undoes the quantization process and brings the values back to their original scale.

Part 3: Test the quantizer/dequantizer functions on a deterministic input

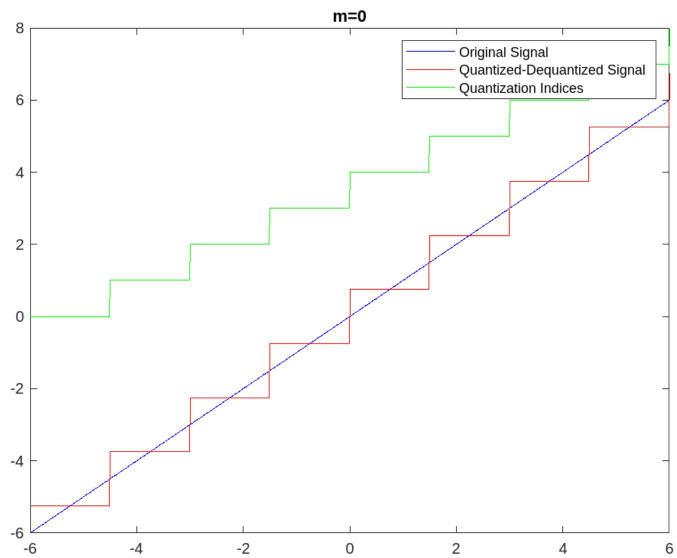

Figure 2 fig

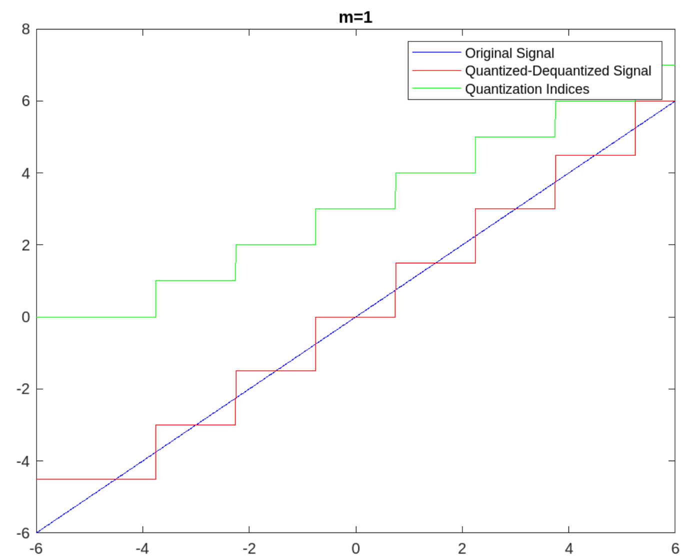

Figure 1 fig

```
n_bits = 3;
xmax = 6;
x = -6:0.01:6;
m_values = [0, 1];
for m = m_values
    q_ind = UniformQuantizer(x, n_bits, xmax, m);
    deq_val = UniformDequantizer(q_ind, n_bits, xmax, m);
    figure;
    plot(x, x, 'b', x, deq_val, 'r', x, q_ind, 'g');
    title(sprintf('m=%d', m));
    legend('Original Signal', 'Quantized-Dequantized Signal', 'Quantization Indices');
end
```

Part 4: Test the quantizer/dequantizer functions on random input

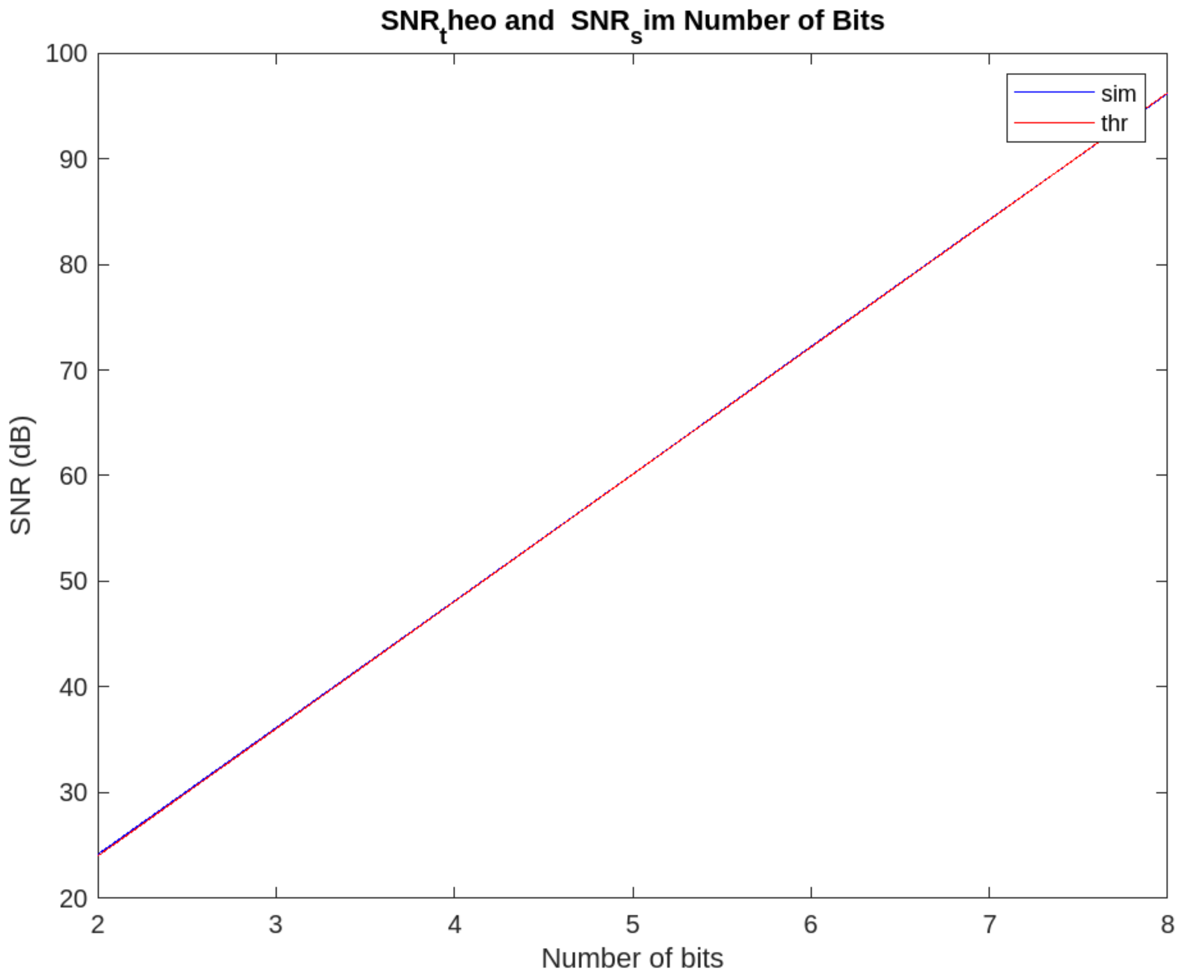

Figure 3 fig

```
sim_snr=[];
theo_snr=[];
x = random('Uniform',-5,5,1,10000);
xmax=max(abs(x));
m=0;
for n_bits= 2:1:8
  y=UniformQuantizer(x,n_bits,xmax,m);
  y_deq = UniformDequantizer(y, n_bits, xmax, m);
  error=abs(x-y_deq);
  sim_snr = [sim_snr, mean(x.^2)/mean(error.^2)];
  scale=(3*((2^n_bits)^2))/(xmax^2);
  theo_snr = [theo_snr, scale*mean(x.^2)];
end
n_bits= 2:1:8;


figure;
```

```
plot(n_bits, mag2db(sim_snr),'b', n_bits,mag2db(theo_snr),'r');
xlabel('Number of bits');
ylabel('SNR (dB)');
title('SNR_theo and  SNR_sim Number of Bits');
legend('sim','thr');
```

Comment:

The gap between simulated SNR and theoretical SNR is minimal, nearly approaching zero across all bits.

Part 5: Test the quantizer/dequantizer functions on non-uniform random input

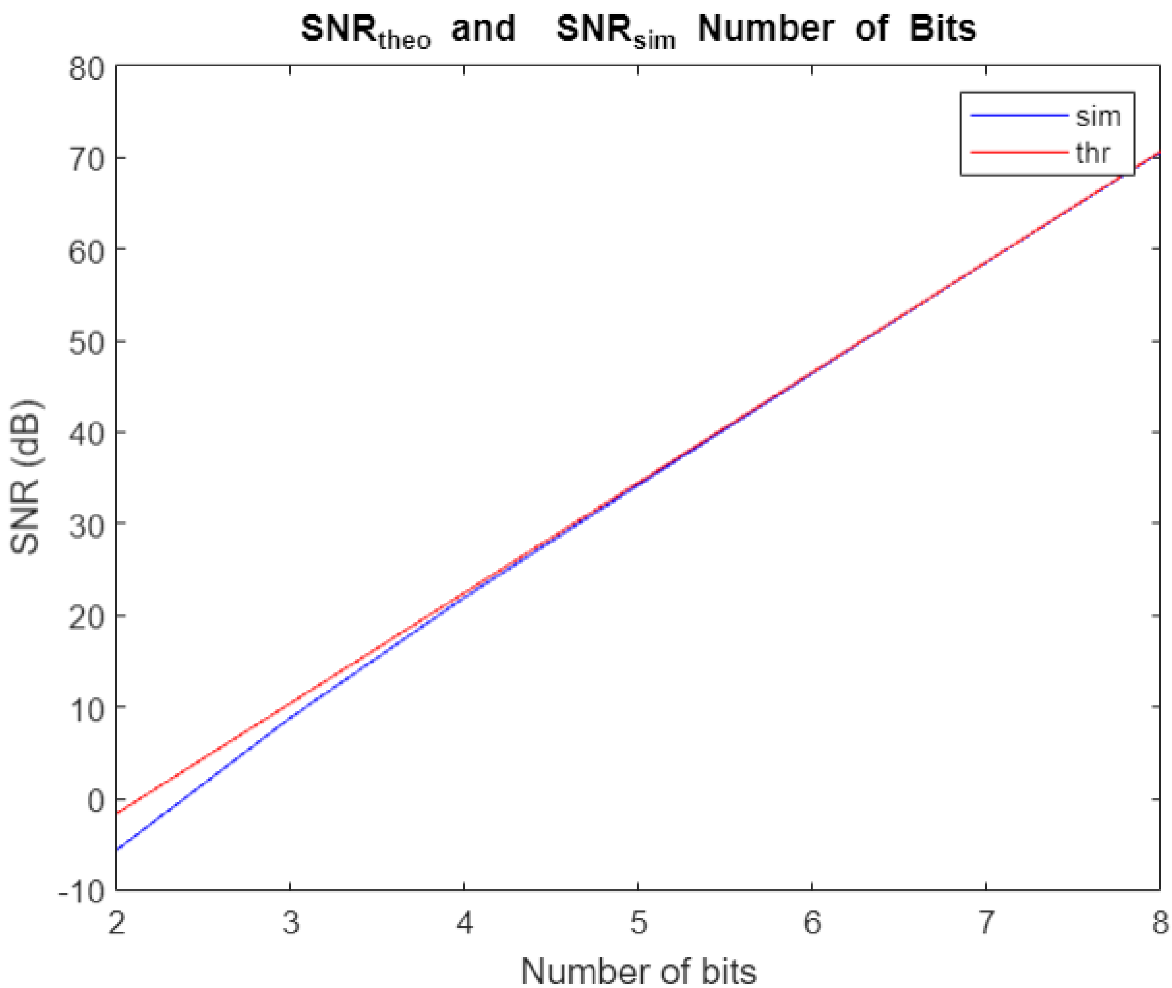

Figure 4 fig

```
sim_snr=[];
theo_snr=[];
size = [1 10000];
x_exp = exprnd(1,size);
sign = (randi([0,1],size)*2)-1;
x = x_exp.*sign;
xmax=max(abs(x));
m=0;
for n_bits= 2:1:8
y=UniformQuantizer(x,n_bits,xmax,m);
y_deq = UniformDequantizer(y, n_bits, xmax, m);
error=abs(x-y_deq);
sim_snr = [sim_snr, mean(x.^2)/mean(error.^2)];
scale=(3*((2^n_bits)^2))/(xmax^2);
theo_snr = [theo_snr, scale*mean(x.^2)];
end
```

```
n_bits= 2:1:8;
figure;
plot(n_bits, mag2db(sim_snr),'b', n_bits,mag2db(theo_snr),'r');
xlabel('Number of bits');
ylabel('SNR (dB)');
title('SNR_theo and  SNR_sim Number of Bits');
legend('sim','thr');
```

Comment:

When we apply random exponential data to the uniform quantizer and dequantizer, we notice that the gap between the simulated SNR and theoretical SNR is significant, especially with fewer bits. However, this difference decreases as we increase the number of bits.

Part 6: Test the quantizer/dequantizer functions on non-uniform signal using a non-uniform μ law quantizer

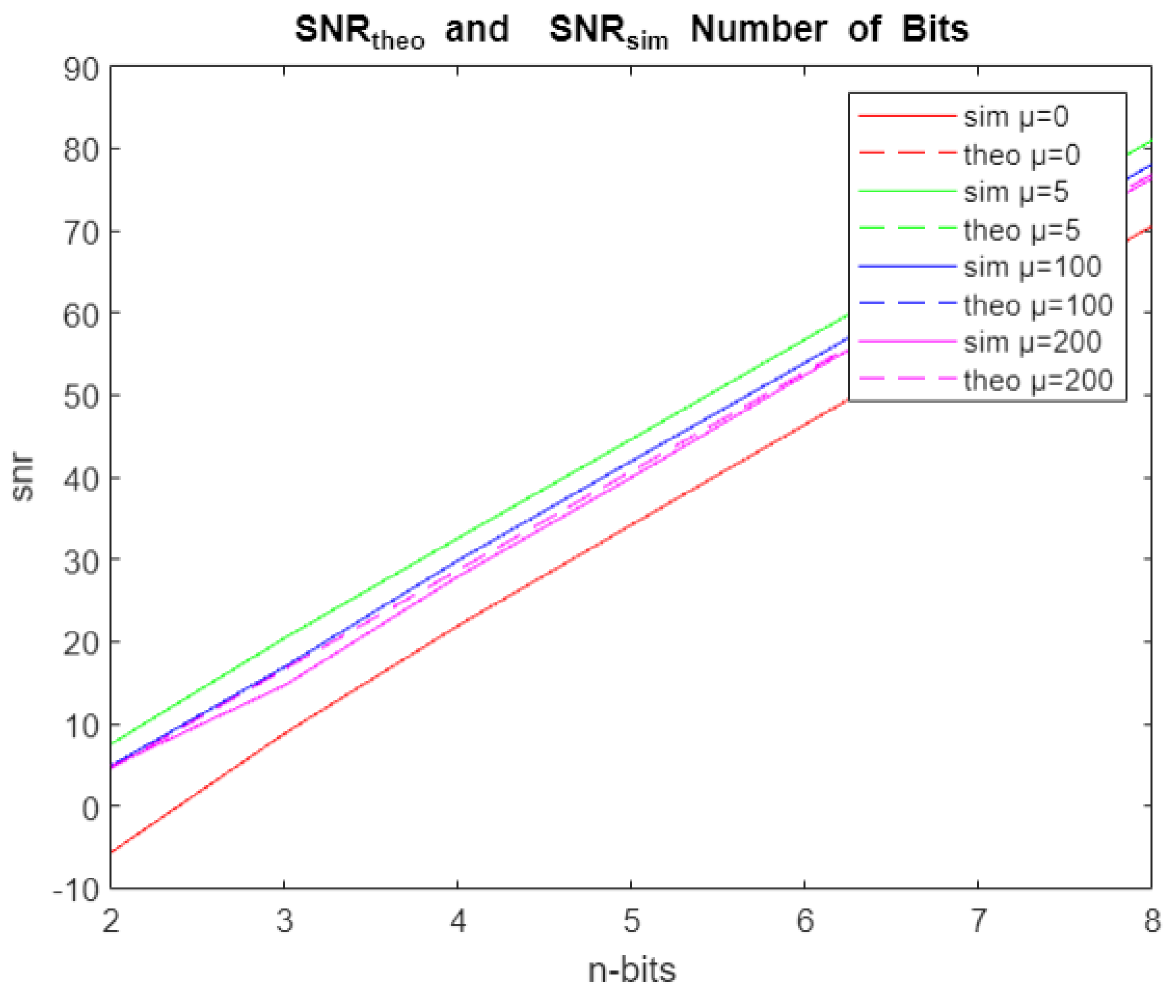

Figure 5 fig

```
figure();
x_norm=x/xmax;
c={"r",'g','b','m'};
i=1;
for mu=[0, 5, 100,200]
sim_snr=[];
theo_snr=[];
if(mu~=0)
x_comp = Compression(x_norm,mu,sign);
else
x_comp=x;
end
ymax=max(abs(x_comp));
for n_bits= 2:1:8
y = UniformQuantizer(x_comp, n_bits,ymax, m);
y_deq = UniformDequantizer(y, n_bits, ymax, m);

if(mu~=0)
```

```
y_expand = Expansion(y_deq,mu,sign);
y_deq = y_expand *xmax;
end
error=abs(x-y_deq);
sim_snr = [sim_snr, mean(x.^2)/mean(error.^2)];
if(mu~=0)
scale=(3*((2^n_bits)^2));
theo_snr = [theo_snr,scale/((log(1+mu))^2)];
else
scale=(3*((2^n_bits)^2))/(xmax^2);
theo_snr = [theo_snr, scale*mean(x.^2)];
end
end
n_bits= 2:1:8;
plot(n_bits,mag2db(sim_snr),'-','color',c{i})
hold on
plot(n_bits,mag2db(thr_snr),'--','color',c{i})
title("SNR_theo and  SNR_sim Number of Bits")
xlabel("n-bits")
ylabel("snr")
i=i+1;
end
legend('sim μ=0','theo μ=0','sim μ=5','theo μ=5','sim μ=100','theo μ=100','sim μ=200','theo μ=200');
```

Comment:

We utilize a non-uniform quantizer for random data to minimize the disparity between theoretical and simulated SNR values, as requested in requirement 5. The effectiveness of this approach depends on the chosen value of μ.

The relationship becomes evident when observing the graph: as the μ value increases for the μ-law quantizer, the gap between theoretical and simulated SNR diminishes. At μ=0, the quantization scheme aligns with requirement 5 (uniform quantization).

Steps:

1. Generate random data with a random sign (equally probable).

2. Compress the data before quantization using the compression function.

```
function y = Compression(x, u, sign)
y=sign .* (log(1+u*abs(x))/log(1+u));
end
```

3. Apply uniform quantization.

4. Perform uniform dequantization.

5. Expand the data using the expansion function.

```
function y = Expansion(x, u, sign)
y= sign .*(((1+u).^abs(x)-1)/u);
end
```

Index: Code using MATLAB

```
% 3- Test the quantizer/dequantizer functions
n_bits = 3;
xmax = 6;
x = -6:0.01:6;
m_values = [0, 1];
for m = m_values
    q_ind = UniformQuantizer(x, n_bits, xmax, m);
    deq_val = UniformDequantizer(q_ind, n_bits, xmax, m);
    figure;
    plot(x, x, 'b', x, deq_val, 'r', x, q_ind, 'g');
    title(sprintf('m=%d', m));
    legend('Original Signal', 'Quantized-Dequantized Signal', 'Quantization Indices');
end

% 4- Test on random input signal
sim_snr=[];
theo_snr=[];
x = random('Uniform',-5,5,1,10000);
xmax=max(abs(x));
m=0;
for n_bits= 2:1:8
  y=UniformQuantizer(x,n_bits,xmax,m);
  y_deq = UniformDequantizer(y, n_bits, xmax, m);
  error=abs(x-y_deq);
  sim_snr = [sim_snr, mean(x.^2)/mean(error.^2)];
  scale=(3*((2^n_bits)^2))/(xmax^2);
  theo_snr = [theo_snr, scale*mean(x.^2)];
end
n_bits= 2:1:8;


figure;
plot(n_bits, mag2db(sim_snr),'b', n_bits,mag2db(theo_snr),'r');
xlabel('Number of bits');
ylabel('SNR (dB)');
title('SNR_theo and SNR_sim Number of Bits');
legend('sim','thr');

% 5- Test on non-uniform random input
sim_snr=[];
theo_snr=[];
size = [1 10000];
x_exp = exprnd(1,size);
sign = (randi([0,1],size)*2)-1;
x = x_exp.*sign;
xmax=max(abs(x));
m=0;
for n_bits= 2:1:8
y=UniformQuantizer(x,n_bits,xmax,m);
y_deq = UniformDequantizer(y, n_bits, xmax, m);
error=abs(x-y_deq);
sim_snr = [sim_snr, mean(x.^2)/mean(error.^2)];
scale=(3*((2^n_bits)^2))/(xmax^2);
theo_snr = [theo_snr, scale*mean(x.^2)];
end
```

```matlab
n_bits= 2:1:8;
figure;
plot(n_bits, mag2db(sim_snr),'b', n_bits,mag2db(theo_snr),'r');
xlabel('Number of bits');
ylabel('SNR (dB)');
title('SNR_theo and  SNR_sim Number of Bits');
legend('sim','thr');
% 6- Non-uniform Mu-law quantization

figure();
x_norm=x/xmax;
c={"r",'g','b','m'};
i=1;
for mu=[0, 5, 100,200]
sim_snr=[];
theo_snr=[];
if(mu~=0)
x_comp = Compression(x_norm,mu,sign);
else
x_comp=x;
end
ymax=max(abs(x_comp));
for n_bits= 2:1:8
y = UniformQuantizer(x_comp, n_bits,ymax, m);
y_deq = UniformDequantizer(y, n_bits, ymax, m);
if(mu~=0)
y_expand = Expansion(y_deq,mu,sign);
y_deq = y_expand *xmax;
end
error=abs(x-y_deq);
sim_snr = [sim_snr, mean(x.^2)/mean(error.^2)];
if(mu~=0)
scale=(3*((2^n_bits)^2));
theo_snr = [theo_snr,scale/((log(1+mu))^2)];
else
scale=(3*((2^n_bits)^2))/(xmax^2);
theo_snr = [theo_snr, scale*mean(x.^2)];
end
end
n_bits= 2:1:8;
plot(n_bits,mag2db(sim_snr),'-','color',c{i})
hold on
plot(n_bits,mag2db(thr_snr),'--','color',c{i})
title("SNR_theo and  SNR_sim Number of Bits")
xlabel("n-bits")
ylabel("snr")
i=i+1;
end
legend('sim μ=0','theo μ=0','sim μ=5','theo μ=5','sim μ=100','theo μ=100','sim μ=200','theo
μ=200');
function y = Compression(x, u, sign)
y=sign .* (log(1+u*abs(x))/log(1+u));
end
function y = Expansion(x, u, sign)
y= sign .*(((1+u).^abs(x)-1)/u);
end
function q_ind = UniformQuantizer(in_val, n_bits, xmax, m)
levels = 2 ^ n_bits;
```

```
delta = 2 * xmax / levels;
q_ind = floor((in_val - ((m) * (delta / 2) - xmax)) / delta);
q_ind(q_ind<0) = 0;
end

function deq_val = UniformDequantizer(q_ind, n_bits, xmax, m)
levels = 2 ^ n_bits;
delta = 2 * xmax / levels;
deq_val = ((q_ind) * delta) + ((m+1) * (delta / 2) - xmax);
end
```