Assignment 1

Name	Section	BN	Student code
Abdelrahman Mohamed Salem Hassan	1	38	9202794
Ahmed Fawzy Mohamed Ibrahim	1	8	9202153

Table of Contents

Table of Contents

Abstract

This assignment was all about doing some uniform/non-uniform quantization and dequantization on some uniform/random input, so

- in **Part1** of the assignment: we implemented a uniform quantizer that preform uniform quantization on the input given the max value for the quantization , number of bits used to construct our levels of quantization and whether to quantize using midtread or midrise signal and the quantizer return the index of the level for different signals values.
- In **Part2** of the assignment: we implement a uniform dequantizer that takes the index of the level returned by the uniform quantizer and map it to actual value
- In **Part3** of the assignment: we made a simple code to test out quantizer/dequantizer with uniform ramp signal that changes from -6 to 6 with step 0.01 and we tested this signal one time using midrise representation and another time using midtread representation.
- In **part4** of the assignment: we made uniform random variable that changes from -5 to 5 and pass it the quantizer and we inspected the value that came from the dequantizer in order to compare the theoretical SNR of original signal versus actual SNR of the dequantized one for number of bits changing from 2 to 8, using midrise representation.
- In **part5** of the assignment: we repeated Part4 of the assignment but for a non-uniform random input that changes very rapidly and we tried to input it to a uniform quantizer and notices the difference between theoretical SNR before and actual SNR after as in part4 of the assignment
- In **Part6** of the assignment: we repeated part5 but we build a non-uniform quantizer by making a block that contains “compressor and uniform quantizer” and another block that contains “dequantizer and expander”, the compressor and expander are using μ-law to quantize and dequantize non-uniform signals and then we inspected the theoretical SNR of original and actual SNR dequantized signals as in part5.

Codes:

1. Codes for Part 1 of the assignment:

```
1.
2. %---------------------------------------%
3. % Requirement 1
4. %---------------------------------------%
5. % this is our quantizer that converts signals to levels (0 -> 2^n-1)
6. function [q_out] = UniformQuantizer(in_val, n_bits, xmax, m)
7.
8.     % get the number of the levels
9.     numOfLevels = 2^n_bits;
10.
11.    % calculate the step size
12.    stepSize = (2*xmax) / numOfLevels;
13.
14.    % get the min and max values of the signals
15.    minVal = (-m * stepSize /2) - xmax + stepSize /2;
16.    maxVal = (-m * stepSize /2) + xmax - stepSize /2;
17.
18.    % get the possible the values
19.    values = minVal:stepSize:maxVal;
20.
21.    % repeat the vectors so that we can subtract them
22.    repeatedValues = repmat(values, [length(in_val) 1]);
23.    repeatedIn_vals = repmat(in_val', [1 length(values)]);
24.
25.    % subtract the each value in the repeatedIn_val from values to get the
26.    % least non-negative number
27.    subtractedVal = abs(repeatedValues' - repeatedIn_vals');
28.
29.    % get the index of the least non-negative number
30.    [~, closestIndex] = min(subtractedVal, [], 'omitnan');
31.
32.    % get the closests values to the readings
33.    q_out = closestIndex - 1;
34.
35. end
```

So this is the function responsible for unform-quantization, that takes input as follow

```
6. function [q_out] = UniformQuantizer(in_val, n_bits, xmax, m)
```

in_val	This is the input signal to be quantized
n_bits	This is the number of bits available to express each level
xmax	This is the max magnitude value of input signal
m	0: means constructed signal will be midrise 1: means constructed signal will be midtread

```
% get the number of the levels
numOfLevels = 2^n_bits;

% calculate the step size
stepSize = (2*xmax) / numOfLevels;

% get the min and max values of the signals
minVal = (-m * stepSize /2) - xmax + stepSize /2;
maxVal = (-m * stepSize /2) + xmax - stepSize /2;

% get the possible the values
values = minVal:stepSize:maxVal;

```

In these 4 lines we are calculating the needed parameters for calculating our vector that will represent our available levels of values.

```
% repeat the vectors so that we can subtract them
repeatedValues = repmat(values, [length(in_val) 1]);
repeatedIn_vals = repmat(in_val', [1 length(values)]);

% subtract the each value in the repeatedIn_val from values to get the
% least non-negative number
subtractedVal = abs(repeatedValues' - repeatedIn_vals');

% get the index of the least non-negative number
[~, closestIndex] = min(subtractedVal, [], 'omitnan');

% get the closests values to the readings
q_out = closestIndex - 1;
```

In these line we are actually calculating the index of level the every value of the input signal by subtracting each value of the signal from the constructed vector that represents our available levels to get the nearest level to that value of the signal and then return the index of the value that's nearest to the value of the signal.

2. Codes for Part 2 of the assignment:

```
%---------------------------------------%
% Requirement 2
%---------------------------------------%
% this is the dequantizer that converts levels back to amplitude
function [deq_val] = UniformDequantizer(q_ind, n_bits, xmax, m)

    % get the number of the levels
    numOfLevels = 2^n_bits;

    % calculate the step size
    stepSize = (2*xmax) / numOfLevels;

    % get the min and max values of the signals
    minVal = (-m * stepSize /2) - xmax + stepSize /2;

    % calculate the values
    deq_val = q_ind * stepSize + minVal;

end
```

This function is to map the index of the level returned by the quantizer to actual value, so we are constructing the vector that represents our available values and then map the given indexes given by the quantizer to actual values.

3. Codes for Part 3 of the assignment:

```
%----------------------------------------%
% Requirement 3
%----------------------------------------%

% creating signal
x = -6:0.01:6;

% quantizing and dequantizing the signal using m = 0
quantizedVals = UniformQuantizer(x, 3, 6, 0);
disp(quantizedVals);
dequantizedVals = UniformDequantizer(quantizedVals, 3, 6, 0);
disp(dequantizedVals);

% plot the graph when m = 0
figure
title('midrise');
hold on
plot(x, '--','DisplayName','original X value');
plot(dequantizedVals, 'DisplayName','after quantization / dequantization');
legend;
hold off

% quantizing and dequantizing the signal using m = 1
quantizedVals = UniformQuantizer(x, 3, 6, 1);
disp(quantizedVals);
dequantizedVals = UniformDequantizer(quantizedVals, 3, 6, 1);
disp(dequantizedVals);

% plot the graph when m = 1
figure
title('midtread');
hold on
plot(x, '--','DisplayName','original X value');
plot(dequantizedVals, 'DisplayName','after quantization / dequantization');
legend;
hold off

```

So here we are creating the actual signal called `x` which is uniform random variable that changes from -6 to 6 with step size 0.01 and then we are trying to quantize and dequantize it one time using midrise representation and another time using midtread representation and then plot the original signal versus midrise representation one time and another time original signal versus midtread signal, where the number of bits to quantize with is **3 bits** and **xmax** is 6.

```
1.
2. % quantizing and dequantizing the signal using m = 0
3. quantizedVals = UniformQuantizer(x, 3, 6, 0);
4. disp(quantizedVals);
5. dequantizedVals = UniformDequantizer(quantizedVals, 3, 6, 0);
6. disp(dequantizedVals);
7.
8. % plot the graph when m = 0
9. figure
10. title('midrise');
11. hold on
12. plot(x, '--','DisplayName','original X value');
13. plot(dequantizedVals, 'DisplayName','after quantization / dequantization');
14. legend;
15. hold off
```

In the above lines of code, we quantize and dequantized the signal called **x** using number of bits of **3 bits**, **xmax** with value 6 and **m** with value 0 to get a midrise representation of the signal and then we plot the original signal versus the dequantized signal.

```
1. % quantizing and dequantizing the signal using m = 1
2. quantizedVals = UniformQuantizer(x, 3, 6, 1);
3. disp(quantizedVals);
4. dequantizedVals = UniformDequantizer(quantizedVals, 3, 6, 1);
5. disp(dequantizedVals);
6.
7. % plot the graph when m = 1
8. figure
9. title('midtread');
10. hold on
11. plot(x, '--','DisplayName','original X value');
12. plot(dequantizedVals, 'DisplayName','after quantization / dequantization');
13. legend;
14. hold off
```

In the above lines of code, we quantize and dequantized the signal called **x** using number of bits of **3 bits**, **xmax** with value 6 and **m** with value 1 to get a midtread representation of the signal and then we plot the original signal versus the dequantized signal.

4. Codes for Part 4 of the assignment:

```
%----------------------------------------%
% Requirement 4
%----------------------------------------%

% generating random signal
x = -5 + 10 * rand(1, 10000);

% number of bits to be tested on
n_bits = 2:1:8;

% create 2 vectors for storing both theoritcal and actual SNR
SNR_theortical = zeros(1, length(n_bits));
SNR_actual = zeros(1, length(n_bits));

% quantizing/dequantizing over different number of bits but same signal,
% also calculating the theortical
for i = 1:length(n_bits)
    % quantizing and deqantizing
    quantizedVals = UniformQuantizer(x, n_bits(i), 5, 0);
    dequantizedVals = UniformDequantizer(quantizedVals, n_bits(i), 5, 0);
    % calculate actual SNR
    differenceInSigs = x - dequantizedVals;
    SNR_actual(i) = mean(x.^2) / mean(differenceInSigs.^2);
    % calculate theoritcal SNR
    LevelsNum = 2^n_bits(i);
    SNR_theortical(i) = 3 * (LevelsNum^2) * mean(x.^2) / (25);
end

% plot the graph , mag2db -> 20log(SNR)
figure
title('SNR comparison(uniform random input)');
hold on
xlabel('n-bits');
ylabel('SNR (in db)');
plot(n_bits, mag2db(SNR_theortical), '--','DisplayName','theoritical SNR');
plot(n_bits, mag2db(SNR_actual), 'DisplayName','actual SNR');
legend;
hold off
```

In the above lines of code, we generate a (i.i.d) random signal which is called **x** and we tried to input this signal to our uniform quantizer/dequantizer and then we compared the theoretical SNR from the law $\frac{3\,L^2}{m^2}P$ and compare it with the actual SNR that can be calculate from : $\frac{\mathrm{E}(input^2)}{\mathrm{E}(\mathrm{quantization\_error}^2)}$ for different values of number of bits used in quantization where we used number of bits ranging from 2 to 8 and we plot a graph of theoretical SNR versus actual SNR.

5. Codes for Part 5 of the assignment:

```
%----------------------------------------%
% Requirement 5
%----------------------------------------%

% array to choose random phase values from
arr = [-1 1];

% generate the magnitude of the signal
x_mag = exprnd(1, 1, 10000);

% generate phase of the signal
index = randi(2, 1, 10000);
x_phase = arr(index);

% calculate the actual random signal
x = x_mag .* x_phase;

% quantizing/dequantizing over different number of bits but same signal,
% also calculating the SNR theortical
for i = 1:length(n_bits)
    % quantizing and deqantizing
    quantizedVals = UniformQuantizer(x, n_bits(i), 5, 0);
    dequantizedVals = UniformDequantizer(quantizedVals, n_bits(i), 5, 0);
    % calculate actual SNR
    differenceInSigs = x - dequantizedVals;
    SNR_actual(i) = mean(x.^2) / mean(differenceInSigs.^2);
    % calculate theoritcal SNR
    LevelsNum = 2^n_bits(i);
    SNR_theortical(i) = 3 * (LevelsNum^2) * mean(x.^2) / (25);
end

% plot the graph , mag2db -> 20log(SNR)
figure
title('SNR comparison(non-uniform random input)');
hold on
xlabel('n-bits');
ylabel('SNR (in db)');
plot(n_bits, mag2db(SNR_theortical), '--','DisplayName','theoritical SNR');
plot(n_bits, mag2db(SNR_actual), 'DisplayName','actual SNR');
legend;
hold off
```

In the above lines we repeated part4 of the assignment but the magnitude of of the signal follows the exponential distribution with PDF f(x)= e^(-x) and the sign of the value is either positive or negative with probability 0.5.

6. Codes for Part 6 of the assignment:

```
%---------------------------------------%
% Requirement 6
%---------------------------------------%

% creating array of possible u
u = [0, 5, 100, 200];

% quantizing/dequantizing over different number of bits but same signal,
% also calculating the SNR theortical
figure
for i = 1:length(u)
    for j = 1:length(n_bits)

        % get the dequantized signal
        dequantizedVals = compressExpand(x, u(i), n_bits(j), 0);

        % calculate actual SNR
        differenceInSigs = x - dequantizedVals;
        SNR_actual(j) = mean(x.^2) / mean(differenceInSigs.^2);

        % calculate theoritcal SNR
        LevelsNum = 2^n_bits(j);

        if u(i) == 0
            SNR_theortical(j) = 3 * (LevelsNum^2) * mean(x.^2) / (25);
        else
            SNR_theortical(j) = 3 * (LevelsNum^2) / (log(1+u(i)) ^ 2);
        end

    end

    % plot the graph , mag2db -> 20log(SNR)
    subplot(2, 2, i);
    title(['u = ' num2str(u(i))]);
    hold on
    xlabel('n-bits');
    ylabel('SNR (in db)');
    plot(n_bits, mag2db(SNR_theortical), '--','DisplayName','theoritical SNR');
    plot(n_bits, mag2db(SNR_actual), 'DisplayName','actual SNR');
    legend;
    hold off

end


% this fucntion is to compress the signal and return the dequantized value
% after expnading the value again
function [deNormalized] = compressExpand(signal, u, n_bits, m)

    % normalize the signal
    normalizedSignal = signal / max(abs(signal));

    % calculate second part of the equation
    compressedVal = (log(1 + u .* abs(normalizedSignal)) / log(1 + u));
    NanIndex = isnan(compressedVal);
    compressedVal(NanIndex) = abs(normalizedSignal(NanIndex));

```

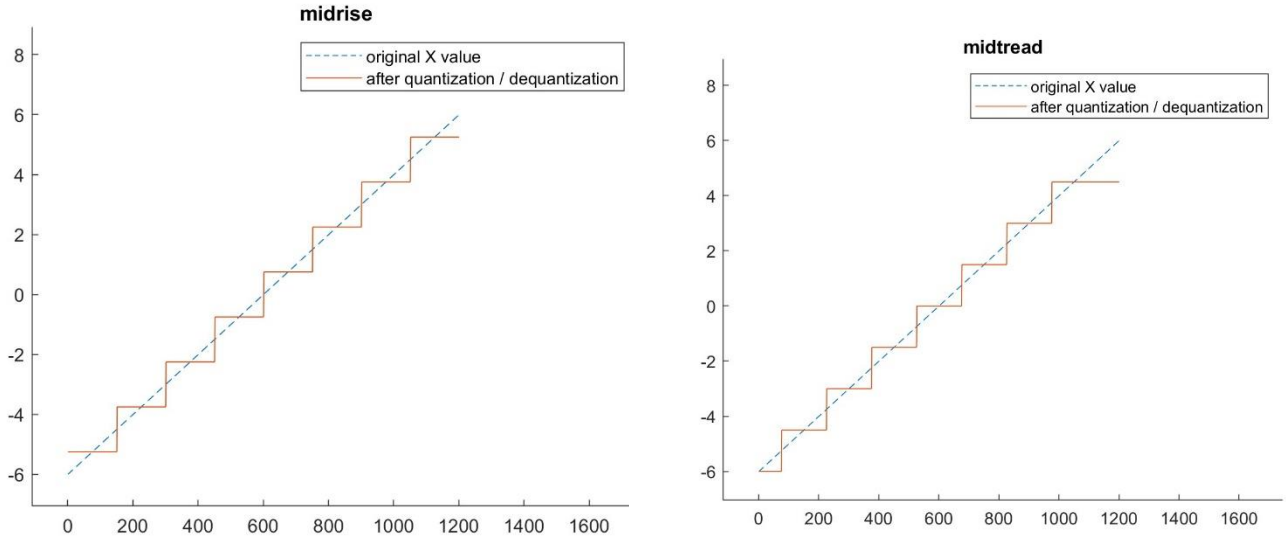
```
    % compress the signal
    compressedSignal = (sign(signal) .* compressedVal);


    % quantize the signal
    quantizedVals = UniformQuantizer(compressedSignal, n_bits, 1, m);

    % dequantize the signal
    dequantizedVals = UniformDequantizer(quantizedVals, n_bits, 1, m);

    % calculate second part of the equation
    expandedVal = ((((1 + u) .^ abs(dequantizedVals)) - 1) / u);
    NanIndex = isnan(expandedVal);
    expandedVal(NanIndex) = abs(dequantizedVals(NanIndex));

    % expand the signal
    expandedSignal = sign(dequantizedVals) .* expandedVal;

    % deNormalize the signal
    deNormalized = expandedSignal .* max(abs(signal));

end

```

In part6, we repeated part5 but instead of using a uniform quantizer/dequantizer, we used a non-uniform one that follow μ-law and then we tried the non-uniform quantizer for different values of μ and compared the theoretical and actual SNR for different number of bits, and we are using midrise representation. so instead of modifying our uniform quantizer/dequantizer,

We created a function called "**compressExpand**" that first takes a signal, normalized it then compress it using μ-law then pass the compressed signal to the uniform quantizer with xmax = 1 then get the quantized values and pass it to the dequantizer, the take the dequantized values and expand it using inverse of μ-law, then it takes the expanded signal and denormalize it and return it.

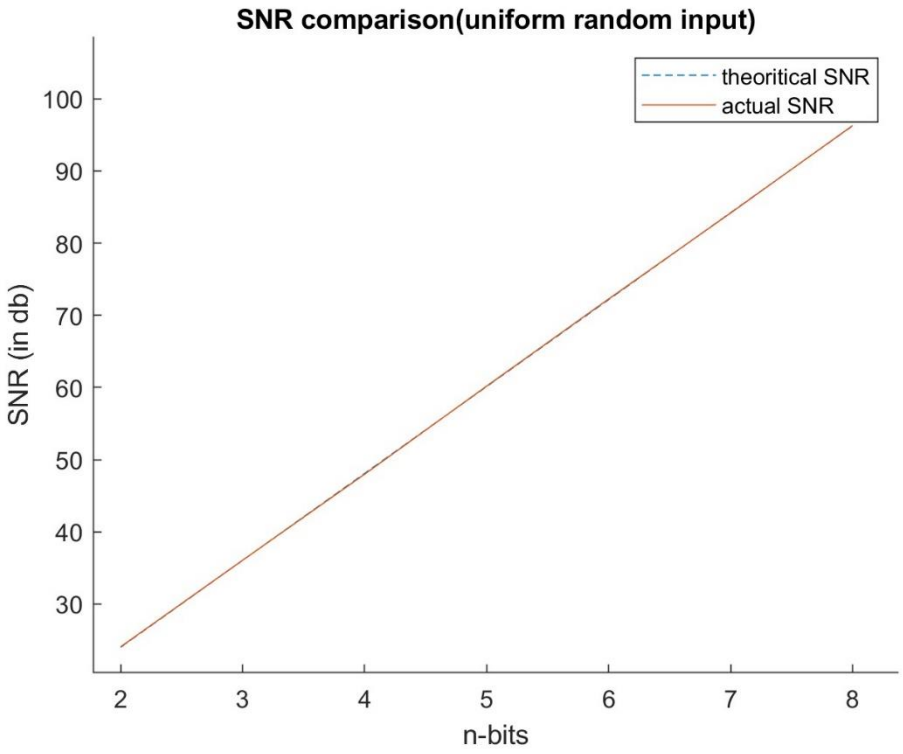
Graphs of the requirements:

Requirement 3



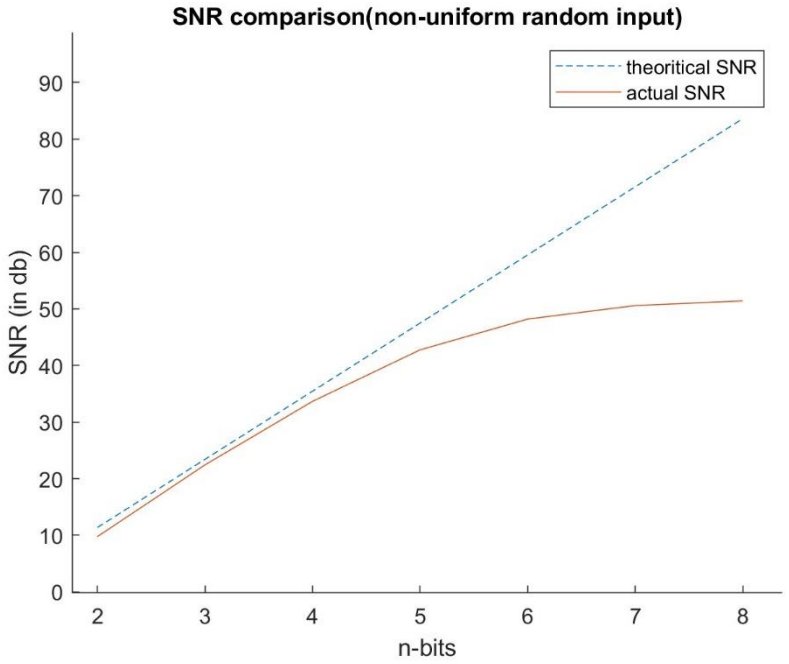
you can notice that the midtread representation will have lower SNR due to shifting the graph down by delta/2, while midrise signal has no defined value at 0 but it has higher SNR as it's symmetric around the x axis.

Requirement 4



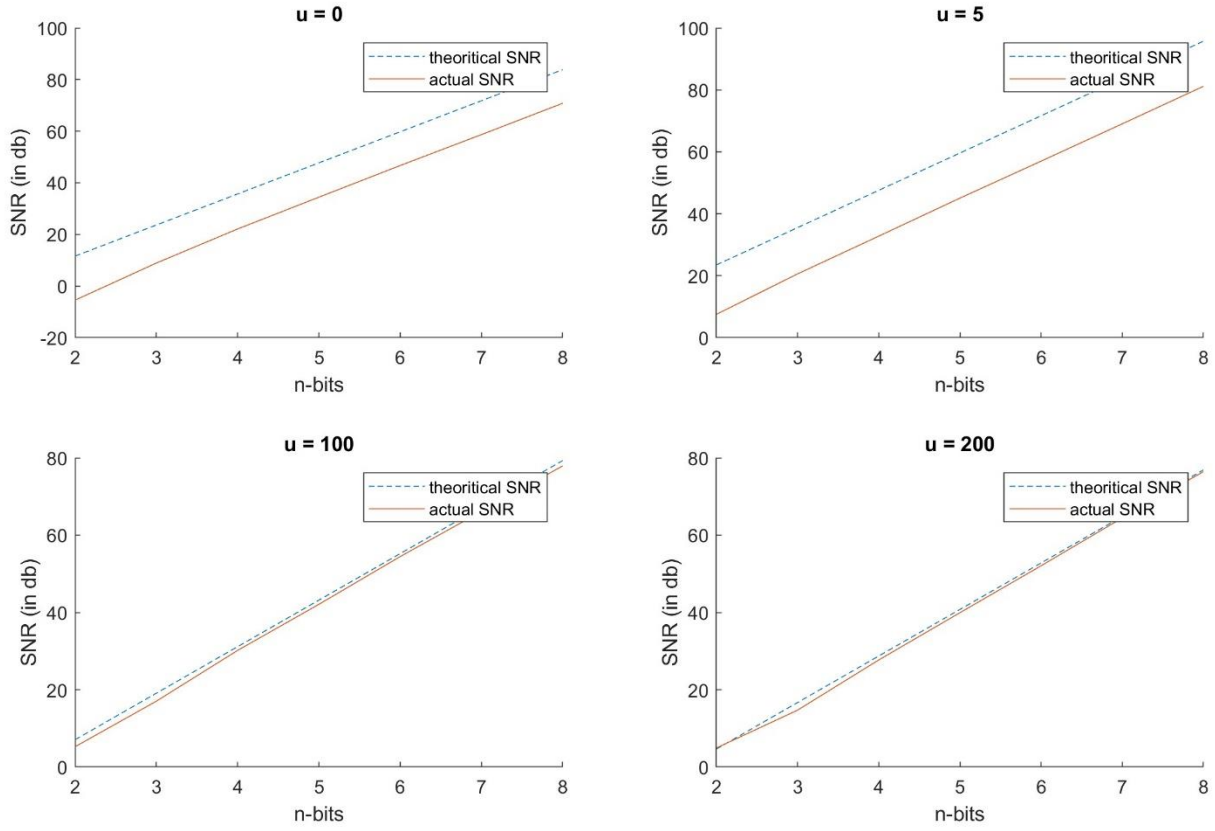
For a big amount of signal data, the theoretical SNR is nearly same as actual SNR, but if we made the signal has small amount of values, you will notice that actual SNR isn't same as theoretical SNR for many values, the actual SNR is nearly same as theoretical SNR.

Requirement 5



From the above graph, Using uniform quantizer with non-uniform input will result in actual bad SNR, so that's why we added a block in requirement 6 to convert the uniform quantizer/dequantizer to a non-uniform one.

Requirement 6



From the above 4 graphs, increasing the value of μ will result in actual SNR nearly equal to the theoretical SNR for a non-uniform input but it may affect negatively the uniform input , while low values of μ will decrease the amount of compression and expanding so it will result in a worst actual SNR for non-uniform input.

Total Code:

```
%---------------------------------------%
% Requirement 3
%---------------------------------------%

% creating signal
x = -6:0.01:6;

% quantizing and dequantizing the signal using m = 0
quantizedVals = UniformQuantizer(x, 3, 6, 0);
disp(quantizedVals);
dequantizedVals = UniformDequantizer(quantizedVals, 3, 6, 0);
disp(dequantizedVals);

% plot the graph when m = 0
figure
title('midrise');
hold on
plot(x, '--','DisplayName','original X value');
plot(dequantizedVals, 'DisplayName','after quantization / dequantization');
legend;
hold off

% quantizing and dequantizing the signal using m = 1
quantizedVals = UniformQuantizer(x, 3, 6, 1);
disp(quantizedVals);
dequantizedVals = UniformDequantizer(quantizedVals, 3, 6, 1);
disp(dequantizedVals);

% plot the graph when m = 1
figure
title('midtread');
hold on
plot(x, '--','DisplayName','original X value');
plot(dequantizedVals, 'DisplayName','after quantization / dequantization');
legend;
hold off
```

```
%---------------------------------------%
% Requirement 4
%---------------------------------------%

% generating random signal
x = -5 + 10 * rand(1, 10000);

% number of bits to be tested on
n_bits = 2:1:8;

% create 2 vectors for storing both theoritcal and actual SNR
SNR_theortical = zeros(1, length(n_bits));
SNR_actual = zeros(1, length(n_bits));

% quantizing/dequantizing over different number of bits but same signal,
% also calculating the theortical
for i = 1:length(n_bits)
    % quantizing and deqantizing
    quantizedVals = UniformQuantizer(x, n_bits(i), 5, 0);
    dequantizedVals = UniformDequantizer(quantizedVals, n_bits(i), 5, 0);
    % calculate actual SNR
    differenceInSigs = x - dequantizedVals;
    SNR_actual(i) = mean(x.^2) / mean(differenceInSigs.^2);
    % calculate theoritcal SNR
    LevelsNum = 2^n_bits(i);
    SNR_theortical(i) = 3 * (LevelsNum^2) * mean(x.^2) / (25);
end

% plot the graph , mag2db -> 20log(SNR)
figure
title('SNR comparison(uniform random input)');
hold on
xlabel('n_bits');
ylabel('SNR (in db)');
plot(n_bits, mag2db(SNR_theortical), '--','DisplayName','theoritical SNR');
plot(n_bits, mag2db(SNR_actual), 'DisplayName','actual SNR');
legend;
hold off
```

```
%---------------------------------------%
% Requirement 5
%---------------------------------------%

% array to choose random phase values from
arr = [-1 1];

% generate the magnitude of the signal
x_mag = exprnd(1, 1, 10000);

% generate phase of the signal
index = randi(2, 1, 10000);
x_phase = arr(index);

% calculate the actual random signal
x = x_mag .* x_phase;

% quantizing/dequantizing over different number of bits but same signal,
% also calculating the SNR theortical
for i = 1:length(n_bits)
    % quantizing and deqantizing
    quantizedVals = UniformQuantizer(x, n_bits(i), 5, 0);
    dequantizedVals = UniformDequantizer(quantizedVals, n_bits(i), 5, 0);
    % calculate actual SNR
    differenceInSigs = x - dequantizedVals;
    SNR_actual(i) = mean(x.^2) / mean(differenceInSigs.^2);
    % calculate theoritcal SNR
    LevelsNum = 2^n_bits(i);
    SNR_theortical(i) = 3 * (LevelsNum^2) * mean(x.^2) / (25);
end

% plot the graph , mag2db -> 20log(SNR)
figure
title('SNR comparison(non-uniform random input)');
hold on
xlabel('n-bits');
ylabel('SNR (in db)');
plot(n_bits, mag2db(SNR_theortical), '--','DisplayName','theoritical SNR');
plot(n_bits, mag2db(SNR_actual), 'DisplayName','actual SNR');
legend;
hold off
```

```
%---------------------------------------%
% Requirement 6
%---------------------------------------%

% creating array of possible u
u = [0, 5, 100, 200];

% quantizing/dequantizing over different number of bits but same signal,
% also calculating the SNR theortical
figure
for i = 1:length(u)
    for j = 1:length(n_bits)

        % get the dequantized signal
        dequantizedVals = compressExpand(x, u(i), n_bits(j), 0);

        % calculate actual SNR
        differenceInSigs = x - dequantizedVals;
        SNR_actual(j) = mean(x.^2) / mean(differenceInSigs.^2);

        % calculate theoritcal SNR
        LevelsNum = 2^n_bits(j);

        if u(i) == 0
            SNR_theortical(j) = 3 * (LevelsNum^2) * mean(x.^2) / (25);
        else
            SNR_theortical(j) = 3 * (LevelsNum^2) / (log(1+u(i)) ^ 2);
        end

    end

    % plot the graph , mag2db -> 20log(SNR)
    subplot(2, 2, i);
    title(['u = ' num2str(u(i))]);
    hold on
    xlabel('n-bits');
    ylabel('SNR (in db)');
    plot(n_bits, mag2db(SNR_theortical), '--','DisplayName','theoritical SNR');
    plot(n_bits, mag2db(SNR_actual), 'DisplayName','actual SNR');
    legend;
    hold off

end
```

```
% this fucntion is to compress the signal and return the dequantized value
% after expnading the value again
function [deNormalized] = compressExpand(signal, u, n_bits, m)

    % normalize the signal
    normalizedSignal = signal / max(abs(signal));

    % calculate second part of the equation
    compressedVal = (log(1 + u .* abs(normalizedSignal)) / log(1 + u));
    NanIndex = isnan(compressedVal);
    compressedVal(NanIndex) = abs(normalizedSignal(NanIndex));

    % compress the signal
    compressedSignal = (sign(signal) .* compressedVal);

    % quantize the signal
    quantizedVals = UniformQuantizer(compressedSignal, n_bits, 1, m);

    % dequantize the signal
    dequantizedVals = UniformDequantizer(quantizedVals, n_bits, 1, m);

    % calculate second part of the equation
    expandedVal = ((((1 + u) .^ abs(dequantizedVals)) - 1) / u);
    NanIndex = isnan(expandedVal);
    expandedVal(NanIndex) = abs(dequantizedVals(NanIndex));

    % expand the signal
    expandedSignal = sign(dequantizedVals) .* expandedVal;

    % deNormalize the signal
    deNormalized = expandedSignal .* max(abs(signal));

end
```

```
%---------------------------------------%
% Requirement 1
%---------------------------------------%
% this is our quantizer that converts signals to levels (0 -> 2^n-1)
function [q_out] = UniformQuantizer(in_val, n_bits, xmax, m)

    % get the number of the levels
    numOfLevels = 2^n_bits;

    % calculate the step size
    stepSize = (2*xmax) / numOfLevels;

    % get the min and max values of the signals
    minVal = (-m * stepSize /2) - xmax + stepSize /2;
    maxVal = (-m * stepSize /2) + xmax - stepSize /2;

    % get the possible the values
    values = minVal:stepSize:maxVal;

    % repeat the vectors so that we can subtract them
    repeatedValues = repmat(values, [length(in_val) 1]);
    repeatedIn_vals = repmat(in_val', [1 length(values)]);

    % subtract the each value in the repeatedIn_val from values to get the
    % least non-negative number
    subtractedVal = abs(repeatedValues' - repeatedIn_vals');

    % get the index of the least non-negative number
    [~, closestIndex] = min(subtractedVal, [], 'omitnan');

    % get the closests values to the readings
    q_out = closestIndex - 1;

end



%---------------------------------------%
% Requirement 2
%---------------------------------------%
% this is the dequantizer that converts levels back to amplitude
function [deq_val] = UniformDequantizer(q_ind, n_bits, xmax, m)

    % get the number of the levels
    numOfLevels = 2^n_bits;

    % calculate the step size
    stepSize = (2*xmax) / numOfLevels;

    % get the min and max values of the signals
    minVal = (-m * stepSize /2) - xmax + stepSize /2;

    % calculate the values
    deq_val = q_ind * stepSize + minVal;

end
```