**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

**abstract3**

**Codes with small explanation4**

Requirement1 codes4

Requirement2 codes6

Requirement3 codes7

Requirement4 codes9

Requirement5 codes10

Requirement6 codes11

**graphs13**

Requirement3 graphs13

Requirement4 graphs 14

Requirement5 graphs 15

Requirement6 graphs 16

**total code17**

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

1. **Codes for Part 1 of the assignment:**

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

|  |
| --- |
| 1. 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 |

|  |
| --- |
| 1. % get the number of the levels 2. numOfLevels = 2^n\_bits; 3. % calculate the step size 4. stepSize = (2\*xmax) / numOfLevels; 6. % get the min and max values of the signals 7. minVal = (-m \* stepSize /2) - xmax + stepSize /2; 8. maxVal = (-m \* stepSize /2) + xmax - stepSize /2; 9. % get the possible the values 10. 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.

|  |
| --- |
| 1. % repeat the vectors so that we can subtract them 2. repeatedValues = repmat(values, [length(in\_val) 1]); 3. repeatedIn\_vals = repmat(in\_val', [1 length(values)]); 4. % subtract the each value in the repeatedIn\_val from values to get the 5. % least non-negative number 6. subtractedVal = abs(repeatedValues' - repeatedIn\_vals'); 8. % get the index of the least non-negative number 9. [~, closestIndex] = min(subtractedVal, [], 'omitnan'); 11. % get the closests values to the readings 12. 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.

1. **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.

1. **Codes for Part 3 of the assignment:**

|  |
| --- |
| 1. %----------------------------------------% 2. % Requirement 3 3. %----------------------------------------% 4. % creating signal 5. x = -6:0.01:6; 6. % quantizing and dequantizing the signal using m = 0 7. quantizedVals = UniformQuantizer(x, 3, 6, 0); 8. disp(quantizedVals); 9. dequantizedVals = UniformDequantizer(quantizedVals, 3, 6, 0); 10. disp(dequantizedVals); 11. % plot the graph when m = 0 12. figure 13. title('midrise'); 14. hold on 15. plot(x, '--','DisplayName','original X value'); 16. plot(dequantizedVals, 'DisplayName','after quantization / dequantization'); 17. legend; 18. hold off 19. % quantizing and dequantizing the signal using m = 1 20. quantizedVals = UniformQuantizer(x, 3, 6, 1); 21. disp(quantizedVals); 22. dequantizedVals = UniformDequantizer(quantizedVals, 3, 6, 1); 23. disp(dequantizedVals); 24. % plot the graph when m = 1 25. figure 26. title('midtread'); 27. hold on 28. plot(x, '--','DisplayName','original X value'); 29. plot(dequantizedVals, 'DisplayName','after quantization / dequantization'); 30. legend; 31. 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.

|  |
| --- |
| 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. % plot the graph when m = 0 8. figure 9. title('midrise'); 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 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. % plot the graph when m = 1 7. figure 8. title('midtread'); 9. hold on 10. plot(x, '--','DisplayName','original X value'); 11. plot(dequantizedVals, 'DisplayName','after quantization / dequantization'); 12. legend; 13. 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.

1. **Codes for Part 4 of the assignment:**

|  |
| --- |
| 1. %----------------------------------------% 2. % Requirement 4 3. %----------------------------------------% 4. % generating random signal 5. x = -5 + 10 \* rand(1, 10000); 6. % number of bits to be tested on 7. n\_bits = 2:1:8; 8. % create 2 vectors for storing both theoritcal and actual SNR 9. SNR\_theortical = zeros(1, length(n\_bits)); 10. SNR\_actual = zeros(1, length(n\_bits)); 11. % quantizing/dequantizing over different number of bits but same signal, 12. % also calculating the theortical 13. for i = 1:length(n\_bits) 14. % quantizing and deqantizing 15. quantizedVals = UniformQuantizer(x, n\_bits(i), 5, 0); 16. dequantizedVals = UniformDequantizer(quantizedVals, n\_bits(i), 5, 0); 17. % calculate actual SNR 18. differenceInSigs = x - dequantizedVals; 19. SNR\_actual(i) = mean(x.^2) / mean(differenceInSigs.^2); 20. % calculate theoritcal SNR 21. LevelsNum = 2^n\_bits(i); 22. SNR\_theortical(i) = 3 \* (LevelsNum^2) \* mean(x.^2) / (25); 23. end 24. % plot the graph , mag2db -> 20log(SNR) 25. figure 26. title('SNR comparison(uniform random input)'); 27. hold on 28. xlabel('n-bits'); 29. ylabel('SNR (in db)'); 30. plot(n\_bits, mag2db(SNR\_theortical), '--','DisplayName','theoritical SNR'); 31. plot(n\_bits, mag2db(SNR\_actual), 'DisplayName','actual SNR'); 32. legend; 33. 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 and compare it with the actual SNR that can be calculate from : 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.

1. **Codes for Part 5 of the assignment:**

|  |
| --- |
| 1. %----------------------------------------% 2. % Requirement 5 3. %----------------------------------------% 4. % array to choose random phase values from 5. arr = [-1 1]; 6. % generate the magnitude of the signal 7. x\_mag = exprnd(1, 1, 10000); 8. % generate phase of the signal 9. index = randi(2, 1, 10000); 10. x\_phase = arr(index); 11. % calculate the actual random signal 12. x = x\_mag .\* x\_phase; 13. % quantizing/dequantizing over different number of bits but same signal, 14. % also calculating the SNR theortical 15. for i = 1:length(n\_bits) 16. % quantizing and deqantizing 17. quantizedVals = UniformQuantizer(x, n\_bits(i), 5, 0); 18. dequantizedVals = UniformDequantizer(quantizedVals, n\_bits(i), 5, 0); 19. % calculate actual SNR 20. differenceInSigs = x - dequantizedVals; 21. SNR\_actual(i) = mean(x.^2) / mean(differenceInSigs.^2); 22. % calculate theoritcal SNR 23. LevelsNum = 2^n\_bits(i); 24. SNR\_theortical(i) = 3 \* (LevelsNum^2) \* mean(x.^2) / (25); 25. end 26. % plot the graph , mag2db -> 20log(SNR) 27. figure 28. title('SNR comparison(non-uniform random input)'); 29. hold on 30. xlabel('n-bits'); 31. ylabel('SNR (in db)'); 32. plot(n\_bits, mag2db(SNR\_theortical), '--','DisplayName','theoritical SNR'); 33. plot(n\_bits, mag2db(SNR\_actual), 'DisplayName','actual SNR'); 34. legend; 35. 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.

1. **Codes for Part 6 of the assignment:**

|  |
| --- |
| 1. %----------------------------------------% 2. % Requirement 6 3. %----------------------------------------% 4. % creating array of possible u 5. u = [0, 5, 100, 200]; 6. % quantizing/dequantizing over different number of bits but same signal, 7. % also calculating the SNR theortical 8. figure 9. for i = 1:length(u) 10. for j = 1:length(n\_bits) 11. % get the dequantized signal 12. dequantizedVals = compressExpand(x, u(i), n\_bits(j), 0); 13. % calculate actual SNR 14. differenceInSigs = x - dequantizedVals; 15. SNR\_actual(j) = mean(x.^2) / mean(differenceInSigs.^2); 16. % calculate theoritcal SNR 17. LevelsNum = 2^n\_bits(j); 19. if u(i) == 0 20. SNR\_theortical(j) = 3 \* (LevelsNum^2) \* mean(x.^2) / (25); 21. else 22. SNR\_theortical(j) = 3 \* (LevelsNum^2) / (log(1+u(i)) ^ 2); 23. end 25. end 26. % plot the graph , mag2db -> 20log(SNR) 27. subplot(2, 2, i); 28. title(['u = ' num2str(u(i))]); 29. hold on 30. xlabel('n-bits'); 31. ylabel('SNR (in db)'); 32. plot(n\_bits, mag2db(SNR\_theortical), '--','DisplayName','theoritical SNR'); 33. plot(n\_bits, mag2db(SNR\_actual), 'DisplayName','actual SNR'); 34. legend; 35. hold off 36. end 37. % this fucntion is to compress the signal and return the dequantized value 38. % after expnading the value again 39. function [deNormalized] = compressExpand(signal, u, n\_bits, m) 41. % normalize the signal 42. normalizedSignal = signal / max(abs(signal)); 43. % calculate second part of the equation 44. compressedVal = (log(1 + u .\* abs(normalizedSignal)) / log(1 + u)); 45. NanIndex = isnan(compressedVal); 46. compressedVal(NanIndex) = abs(normalizedSignal(NanIndex)); 47. % compress the signal 48. compressedSignal = (sign(signal) .\* compressedVal); 50. % quantize the signal 51. quantizedVals = UniformQuantizer(compressedSignal, n\_bits, 1, m); 52. % dequantize the signal 53. dequantizedVals = UniformDequantizer(quantizedVals, n\_bits, 1, m); 54. % calculate second part of the equation 55. expandedVal = ((((1 + u) .^ abs(dequantizedVals)) - 1) / u); 56. NanIndex = isnan(expandedVal); 57. expandedVal(NanIndex) = abs(dequantizedVals(NanIndex)); 58. % expand the signal 59. expandedSignal = sign(dequantizedVals) .\* expandedVal; 61. % deNormalize the signal 62. deNormalized = expandedSignal .\* max(abs(signal)); 63. 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**

Chart

Description automatically generatedChart, histogram

Description automatically generated

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**

Chart, line chart

Description automatically generated

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**

Chart, line chart

Description automatically generated

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**

Chart

Description automatically generated

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 |