EEE 431 TELECOMMUNICATIONS I
MATLAB ASSIGNMENT I

Efe Eren CEYANİ

21903359

EEE 431-01

INTRODUCTION

In this MATLAB assignment, our aim was to experiment with uniform/non-uniform quantization and encoding techniques. Quantization is necessary to represent analog signals in a digital environment, and because of this, often questions such as “How to quantize?” arise. While the uniform quantization, i.e., equally spaced reconstruction boundaries and levels, is optimal for uniform distributions, for non-uniform distributions a non-uniform quantizer is required for a lower quantization error. To examine this behavior, we first quantize a non-uniform distribution with a uniform quantizer, then compare its performance with a non-uniform quantizer obtained by companding. Then, the Lloyd-Max algorithm, which find locally minimum values for the quantization error, is applied.

After experimenting with quantization, two encoding techniques were used: Huffman encoding and Lempel-Ziv-Welch (LZW) encoding. Huffman algorithm tries to assign longer codewords to less frequent words and shorter codewords to more frequent words, and LZW algorithm tries to exploit the recurring patterns to reduce the memory allocated. These two algorithms are implemented and compared with the raw message in terms of length and memory usage.

Part 1

1)

$a = 5, b = 9, X_1 \sim Uniform[0, 5], X_2 \sim Uniform[-9, 0]$

$Y = X_1 + X_2$

$$E[Y^2] = E[(X_1 + X_2)^2] = E[X_1^2 + 2X_1X_2 + X_2^2]$$
$$= E[X_1^2] + 2E[X_1X_2] + E[X_2^2]$$
$$\xrightarrow[X_1 \perp X_2]{} E[X_1^2] + 2E[X_1]E[X_2] + E[X_2^2]$$
$$= \frac{1}{5}\int_0^5 x^2 dx + 2\frac{1}{5}\int_0^5 x dx \frac{1}{9}\int_{-9}^0 x dx + \frac{1}{9}\int_{-9}^0 x^2\, dx$$
$$= \frac{25}{3} - \frac{45}{2} + 27 = \frac{77}{6} \approx 12.8333$$

3)

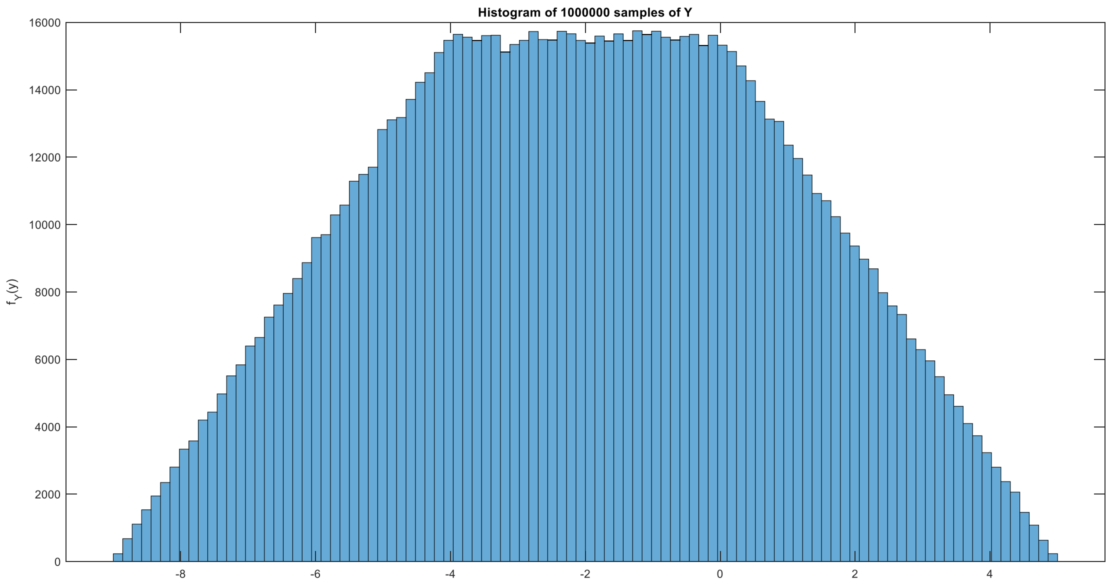


Figure 1.1 Histogram of samples of Y, with a bin size of 100.

In MATLAB, the average value of the squares of these samples was calculated as 12.8301. It is close to the theoretical value, which was 12.8333. We expected this because of the law of large numbers. As we repeat the experiment over and over, the empirical mean approximates to the theoretical mean.

4)

$$SQNR_{in\ dB} = 10\log_{10}\left(\frac{E[Y^2]}{E[(Y^2 - \tilde{Y}^2)]}\right)$$

The average power of the quantization error is 0.2542, and the resulting signal-to-quantization-noise ratio (SQNR) in dB is 17.0305.

5)

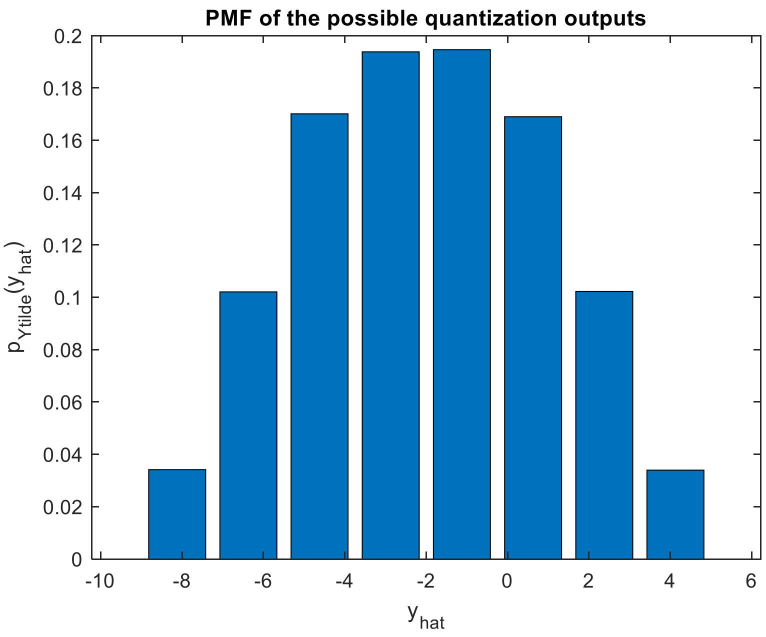
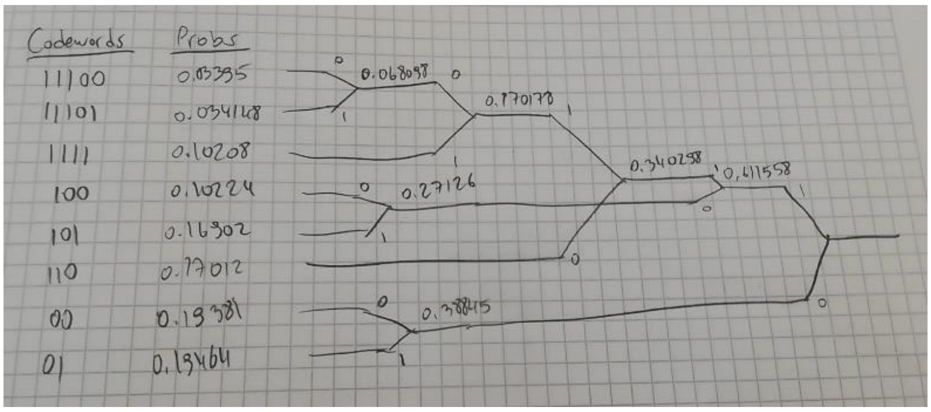


Figure 1.2 PMF of the possible quantization outputs.



Codewords	Probs
11100	0.03395
11101	0.034148
1111	0.10208
100	0.10224
101	0.16902
110	0.17012
00	0.19381
01	0.19464

Figure 1.3 Huffman encoding of the PMF, theoretical.

```
0.0340    0.0681    0.1022    0.1701    0.1938    0.2713    0.3884
0.0341    0.1021    0.1690    0.1702    0.1946    0.3403    0.6116
0.1021    0.1022    0.1701    0.1938    0.2713    0.3884    1.0000
0.1022    0.1690    0.1702    0.1946    0.3403    1.0000    1.0000
0.1690    0.1701    0.1938    0.2713    1.0000    1.0000    1.0000
0.1701    0.1938    0.1946    1.0000    1.0000    1.0000    1.0000
0.1938    0.1946    1.0000    1.0000    1.0000    1.0000    1.0000
0.1946    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
```

Figure 1.4 Huffman encoding of the PMF, MATLAB result, probabilities.

```
"11100"    "1110"    "100"    "110"    "00"    "10"    "0"
"11101"    "1111"    "101"    "111"    "01"    "11"    "1"
 "1111"     "100"    "110"     "00"    "10"     "0"    ""
  "100"     "101"    "111"     "01"    "11"     ""     ""
  "101"     "110"     "00"     "10"    ""       ""     ""
  "110"      "00"     "01"     ""      ""       ""     ""
   "00"      "01"     ""       ""      ""       ""     ""
   "01"      ""       ""       ""      ""       ""     ""
```

Figure 1.5 Huffman encoding of the PMF, MATLAB result, codewords.

Average codeword length with the Huffman encoding is 2.8498bits. Average codeword length without the Huffman encoding is 3 bits.

Algorithm explanation:

I created an 8x7 matrix and arranged it in a such way that the columns of the matrix represent the probabilities at each step. So, in the last column there are only two probabilities whose sum is equal to 1. While moving to the next column, the lowest two probabilities are combined, and the probabilities are sorted before the creation of the new column. After having two valid probabilities in the final column, I started creating the codewords. I always assigned the lower probabilities “0” and higher ones “1”. So, after assigning “0” to 0.38788 and “1” to 0.6121, I looked for two probabilities in the previous column whose sum was equal to one of the probabilities in the last one. Iteratively, I found that 0.2711 + 0.3411 = 0.6121, so their codewords start with 0.6121’s codeword. I also assigned “0” to 0.2711 because it is the lower one. This step was repeated until the initial probabilities were recovered. The function for this is called “HuffmanDeneme”.

6) MATLAB verifies that both the original input sequence and the output of the decoder are identical.

Algorithm Explanation:

First, I created an empty string to store the decoded information. To do this, I searched for substrings in the encoded string to find the Huffman codewords. Whenever I found a matching codeword, I appended the corresponding value back to the output of the decoder.

7) First, we should find the $f_Y(y)$ by convolving X_1 and X_2. We get the following result:

$$f_Y(y) = \begin{cases} \frac{y+9}{45} & ,-9 \le y < -4 \\ \frac{1}{9} & ,4 \le y < 0 \\ \frac{5-y}{45} & ,0 \le y \le 5 \end{cases}$$

Then, $F_Y(y)$ can be found by integrating the PDF of Y.

$$F_Y(y) = \begin{cases} \frac{(y+9)^2}{90} & ,-9 \le y < -4 \\ \frac{5}{18} + \frac{y+4}{9} & ,-4 \le y < 0 \\ \frac{13}{18} + \frac{y}{18} + \frac{(5-y)y}{90} & ,0 \le y \le 5 \end{cases}$$

We can find the inverse of PDF of Y, $F_Y^{-1}(y)$, as the following:

$$F_Y^{-1}(y) = \begin{cases} \sqrt{90y} - 9 & ,0 \le y < \frac{25}{90} \\ 9y - \frac{13}{2} & ,\frac{25}{90} \le y < \frac{13}{18} \\ 5 - \sqrt{90(1-y)} & ,\frac{13}{18} \le y \le 1 \end{cases}$$

8) Check the MATLAB file.

9) Check the MATLAB file.

10)

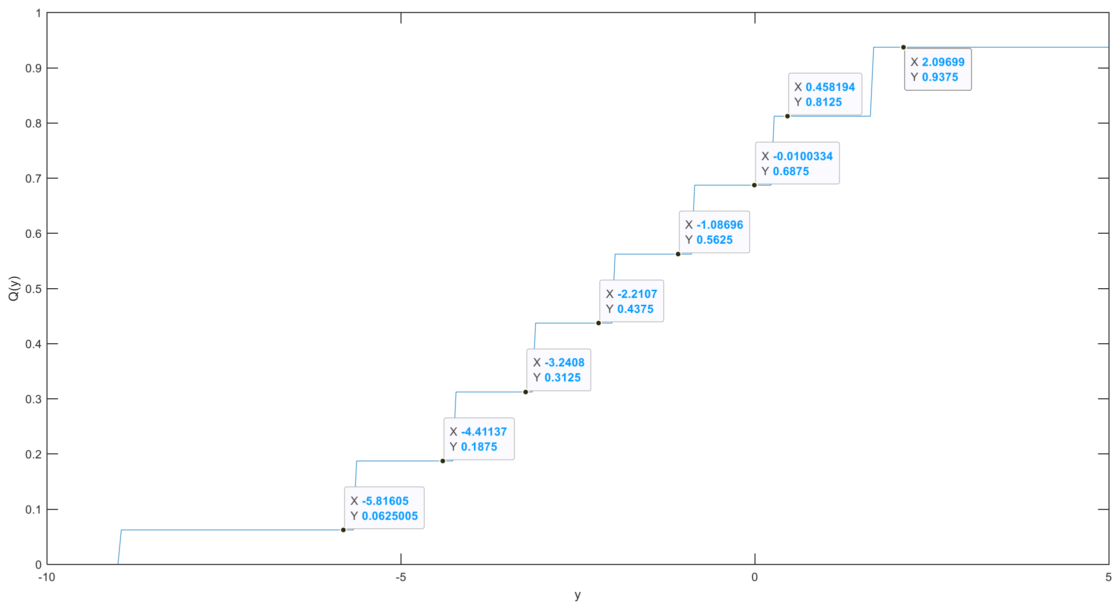


Figure 1.6 Quantization regions vs. corresponding reconstruction points.

11) The average power of the quantization error is 0.2535, and the resulting SQNR in dB is 17.0424.

This result is an improvement compared to the initial SQNR value because our sample distribution is non-uniform. Compander assigns more reconstruction levels to regions which are more frequent.

12)

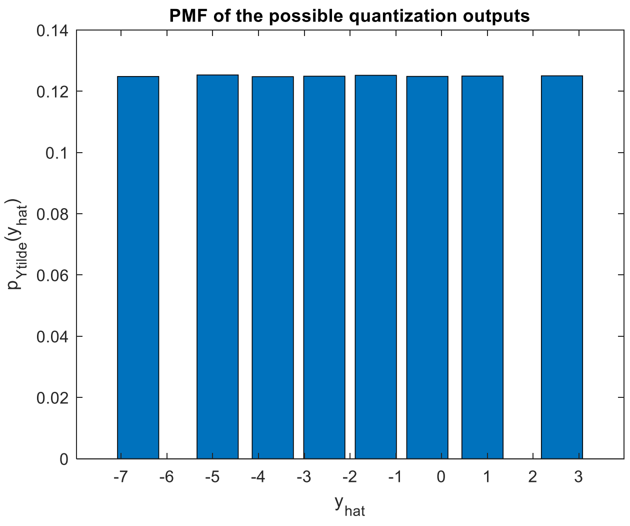


Figure 1.7 PMF of the possible quantization outputs with compressor.

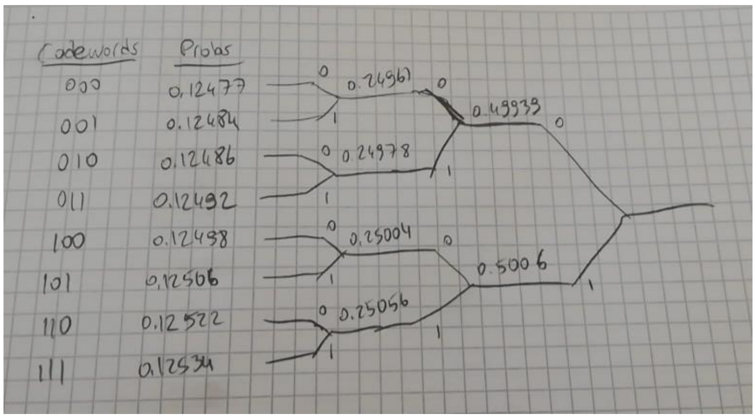


Figure 1.8 Huffman encoding of the PMF, theoretical.

```
0.1248    0.1249    0.1250    0.1252    0.2496    0.2500    0.4994
0.1248    0.1249    0.1251    0.1253    0.2498    0.2506    0.5006
0.1249    0.1250    0.1252    0.2496    0.2500    0.4994    1.0000
0.1249    0.1251    0.1253    0.2498    0.2506    1.0000    1.0000
0.1250    0.1252    0.2496    0.2500    1.0000    1.0000    1.0000
0.1251    0.1253    0.2498    1.0000    1.0000    1.0000    1.0000
0.1252    0.2496    1.0000    1.0000    1.0000    1.0000    1.0000
0.1253    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
```

Figure 1.9 Huffman encoding of the PMF, MATLAB result, probabilities.

```
"000"    "010"    "100"    "110"    "00"    "10"    "0"
"001"    "011"    "101"    "111"    "01"    "11"    "1"
"010"    "100"    "110"    "00"     "10"    "0"     ""
"011"    "101"    "111"    "01"     "11"    ""      ""
"100"    "110"    "00"     "10"     ""      ""      ""
"101"    "111"    "01"     ""       ""      ""      ""
"110"    "00"     ""       ""       ""      ""      ""
"111"    ""       ""       ""       ""      ""      ""
```

Figure 1.10 Huffman encoding of the PMF, MATLAB result, codewords.

Average codeword length with the Huffman encoding is 3 bits. Average codeword length without the Huffman encoding is 3 bits. Huffman algorithm used for this step was identical to the one used in step 5.

13) MATLAB verifies that both the original input sequence and the output of the decoder are identical. The same decoding algorithm in step 6 was also used for this step.

14)

```
-8.99325715377613   -7.05206522509517   -5.41957505992980   -3.71842958125393
  -1.99619137850443   -0.275729300876598 1.42444818899048    3.05581915462158
                              4.99877454247269
```

Figure 1.11 Boundaries for the Lloyd-Max algorithm.

```
-7.83131335880578   -6.27281709138455   -4.56633302847504   -2.87052613403282
  -1.12185662297604   0.570398021222846  2.27849835675812    3.83313995248505
```

Figure 1.12 Reconstruction levels for the Lloyd-Max algorithm.

Algorithm Explanation:

I used the original samples and the initial boundaries to initiate the Lloyd-Max algorithm. I started with a uniform guess for the reconstruction levels. I chose the stopping criterion to be 0.05, meaning that if two consecutive quantization errors are different from each other with a value less than 0.05, the algorithm would stop. To calculate the locally optimal reconstruction values, ideally, we would have to calculate two integrals, however, in the case of this application, we only have the samples of the distribution, assuming we cannot use the PDF. So, approximate the integrals, divided the sum of the samples in a region by the region's width. This gave me the locally optimal reconstruction levels. After finding the locally optimal reconstruction levels, all I had to was taking the averages of reconstruction levels to find the locally optimal boundaries. After calculating the new quantization error, I continued this algorithm until the 0.05 criterion was not met, meaning that consecutive errors are too close to each other so that error is near the locally minimum point. The function for this step is called "LloydMaxFunc".

15)

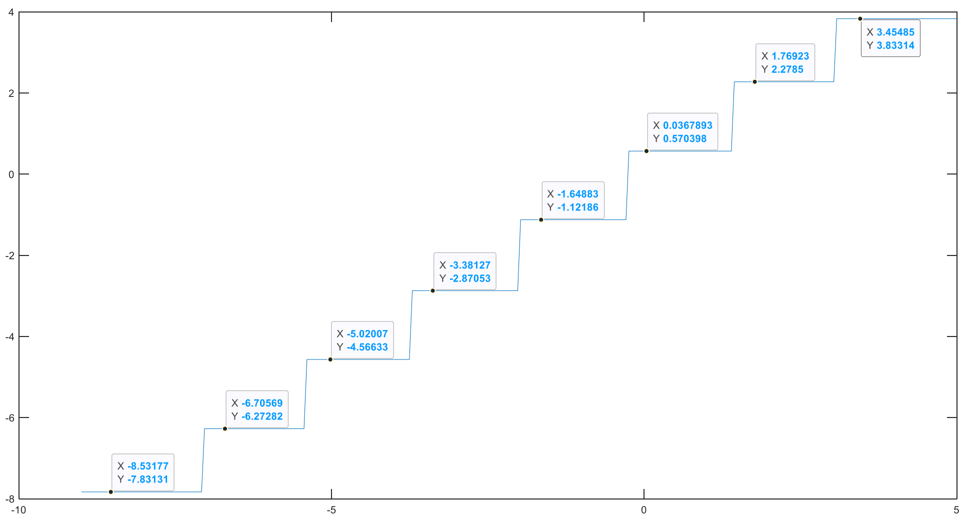


Figure 1.13 Quantization regions vs. corresponding reconstruction points, for Lloyd-Max.

Lloyd-Max’s quantizer is a uniform quantizer, however, compander quantizer was non-uniform.

16) The average power of the quantization error is 0.2390, and the resulting SQNR in dB is 17.2984. This result outperforms the result in the step 13.

Part 2

1) I picked my word corpus from the Wikipedia page of Richard Feynman[1].

Algorithm Explanation:

I parsed the first character of the input sequence and tried continuing the parsing. Whenever the program encountered an unknown character combination, I added the unknown character to the dictionary and encoded the previous known character. Carrying this procedure throughout the input sequence results in the encoded sequence and the finalized dictionary. Whenever the dictionary size surpassed a power of 2, I also increased the length of each codeword.

2) Average codeword length without coding is 5 bits, because there are 28 possible characters. Average codeword length with LZW coding is 9.5413 bits. This result natural because LZW creates new characters with longer codewords, so it is natural for average to increase. LZW decreases the total number of bits used.

Without coding, we would have used 14115 bits, however, with LZW this number decreases to 12127 bits.

3) Check the MATLAB code. It prints out "Input sequence and the decoder output are the same.".

Algorithm Explanation:

I parsed the encoded sequence with a varying length depending on the current dictionary size. Whenever the dictionary size surpassed a power of 2, I also increased the parsing window length. Also, if there was an unknown encoded codeword, I would append the previous translation's first character to the end of the new translation.

4)

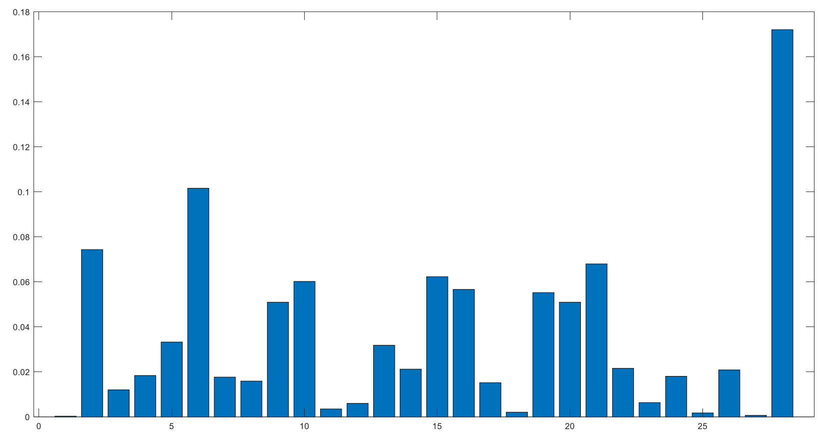


Figure 2.1 PMF of the characters appearing in the word corpus ('#' => 0, ' ' => 27).

The entropy of the corpus is equal to 4.1324 bits. If we take a longer sequence, the chance of having repetition increases. So, for shorter sequence LZW encoding has little to zero impact. The entropy calculated here is smaller than both encoded and raw average codeword lengths, which is the expected result.

[1] https://en.wikipedia.org/wiki/Richard_Feynman.

REFERENCES

https://en.wikipedia.org/wiki/Richard_Feynman