Assignment – 1

Digital Communication

Aryendra Singh 18EC10077

To calculate the value of "a" we simply equate the area under the given pdf to 1. We get the value of $a = \frac{2}{15}$

We then have to generate the custom pdf. In MATLAB we can use a cumulative distribution to create a custom pdf. I used the Heaviside function to define the cdf and pdf in ranges and then plotted the result as shown below. Note that the calculation of the cdf was done manually and the constants to be added after each range was also calculated manually.

In figure 1 the red line is the pdf equation and the histogram is made by binning the samples generated for the random variable X.

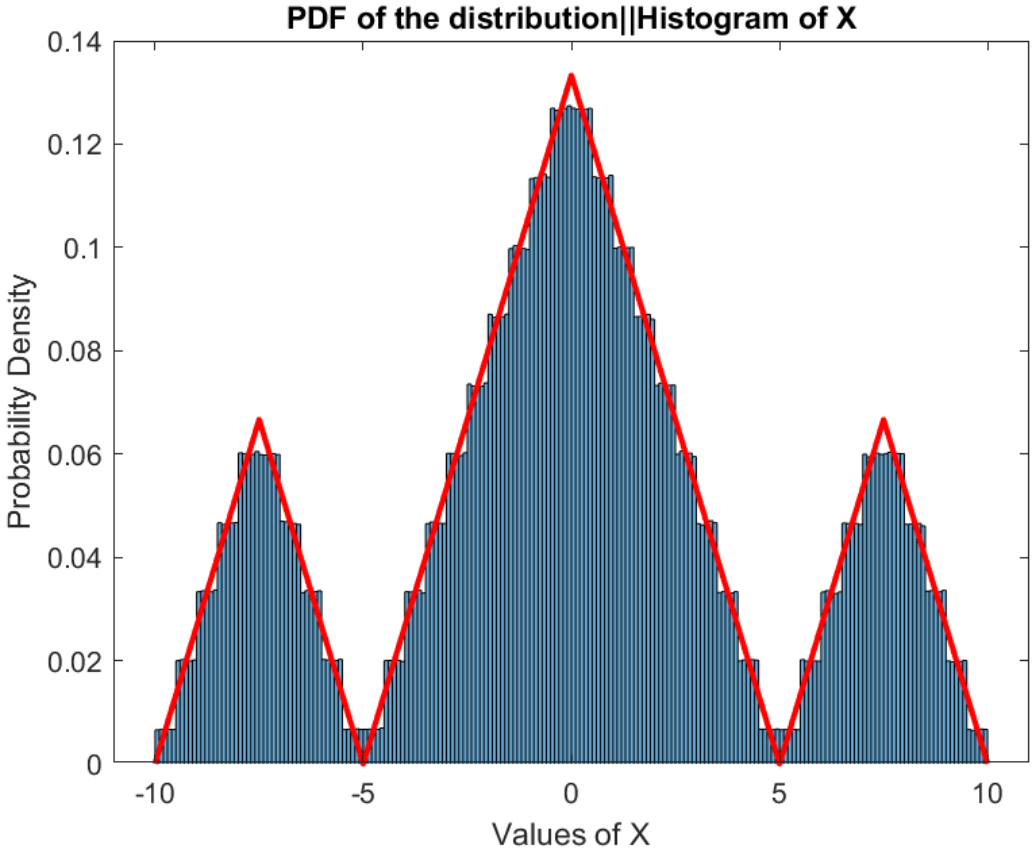
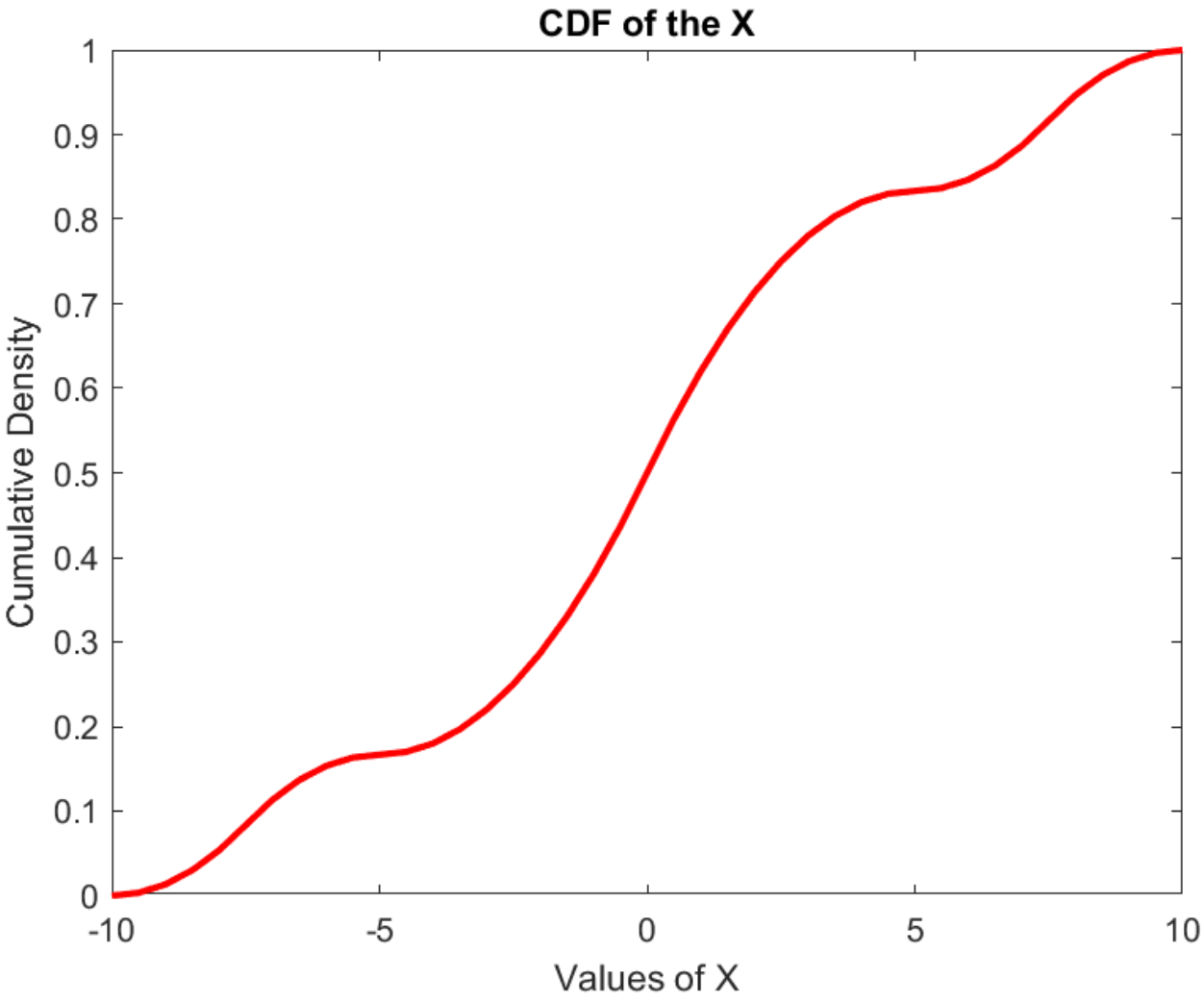



Figure 1

The plot of the cdf of X is as below:



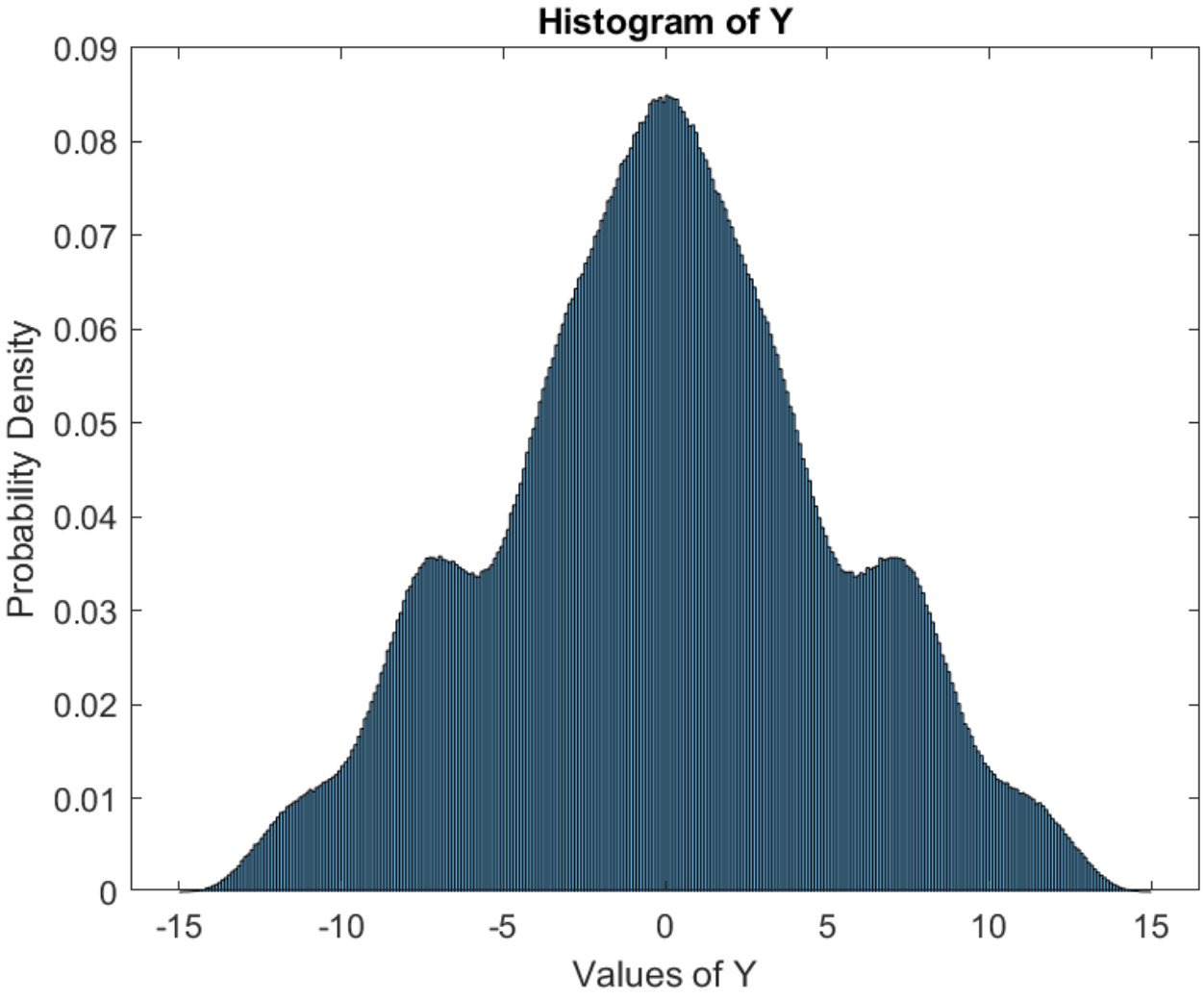

The equation of the pdf is as below:

Note that the symbol $[p, r]$ is for the Heaviside function between $p$ and $r$

$$f(x) = m(x+10)[-10,-7.5] - m(x+5)[-7.5,-5] + m(x+5)[-5,0] - m(x-5)[0,5] + m(x-5)[5,7.5] - m(x-10)[7.5,10]$$

To get the cdf equation we simply integrate the above equation and add the necessary constants.

Below is the pdf plot of the random variable Y. It is produced by the binning the random variables obtained by Y. We can note that the range is now $[-15,15]$ instead of $[-10,10]$.




Next, we tackle the problem of quantizing this random variable. We have a constraint on the mean squared error as $10^{-4}$. We will use uniform quantization.

We use the formula $\frac{\Delta^2}{12} = MSE$, which gives us $\Delta \approx 0.035$. Thus the number of bins is calculated as $M = {range(Y)}/{\Delta} = {30}/{\Delta} \approx 867$

We use the Lloyd Algorithm using the *llyods* function of MATLAB, to obtain the codebook and the partitions(intervals) for the quantization. The

means squared error is found to be $9.98 \times 10^{-5}$, which is very close to our requirement.

Once we have the partitions and the codebook, we will move on to actually quantizing the series of analog values of the random variable Y, into a discrete set of symbols as given in the code book. The symbols will be numbered 0 to 866.

Now that we have a $10^6$ long array of symbols we can move to create the transition probability matrix and the stationary probabilities to find the entropy of Y. Our methodology is as below:

To find the stationary probability we can either find the vector $p$ such that $p \times TPM = p$ or we can find it statistically by simply counting the number of occurrences of each symbol in the $10^6$ long array. We use the latter methodology as finding $p$ for a 867 × 867 matrix is a time intensive job.

To find the TPM we make a single pass of the symbols array, counting occurrences of the doublets given by 2 consecutive symbols. We keep on updating an empty 867 × 867 matrix with the corresponding occurrences.

Finally, we divide each row in the TPM by the total number of occurrences of that symbol (given by the row number), to find the probabilities.

For example, if a part of the sequence is ….4,255,654,3…. Then the doublets formed are (4,255), (255,654), (654,3) and the corresponding cells in TPM updated are [5,256], [256,655], [655,4]. Note the add 1 because the symbols are numbered from 0 but MATLAB indexing starts at 1.

Once we have the TPM entries and the stationary probabilities we can find the entropy by the below formula:

$$H(x) = -\sum_{i,j} s_i p_{ij} \log_2 p_{ij}$$

Here $s_i$ is the stationary probability of symbol $i$ and $p_{ij}$ is the entry in the TPM corresponding to symbol $i$ and $j$.

The value of entropy thus obtained is 8.643. That is the average number of bits required per symbol should be $\mathbf{8.643}$.

If we use fixed length coding then we will require **10** bits for each symbol, given by $\text{ceil}(\log_2 867) = 10$

We now use Huffman encoding routine already implemented in MATLAB. It takes the list of symbols and the probabilities (stationary probabilities already calculated) to give us the complete encoding of the file. Then we compute the average bits per symbol using:

$$\sum p_i l_i$$

Where $p_i$ is the probability of a symbol and $l_i$ is the length of that symbol. It comes out to be **9.3**

The minimum length of a symbol is 8 and the maximum is 27.

In the final step we had to implement LZ77 encoding.

A fixed length search window is defined and the first search window is encoded using fixed length binary encoding. Then, a maximum size of the rolling window is defined. From the next symbol from the end of the search window, starting from an array of size 1 to the length of the rolling window, an exact match to string was searched in the search window. The exact string, closest (with respect to distance) was found, and the distance was set as “u”, the length of the string was made “n”. “n” was encoded into unary binary and “u” into binary of fixed length such that the complete window can be encoded.

However, for a length $10^6$ length window running such an algorithm is difficult.

When searching for a rolling window, it is possible that a symbol occurs which was not present in the original search window. In this case, the rolling window would move forward until the new symbol is the first element in the rolling window. The distance “u” would be zero and it would not be encoded in any case. This leads to an error.

Given this difficulty the algorithm ran into an error roughly an hour in to execution. The problem is that LZ77 is designed mostly for text files, it needs symbol sequences which repeat in patterns often enough, as most text files do. The open-source algorithmic implementations of the LZ77 are

all for the alphabets and text files. I modified one of these by converting our numeric symbols to string and then try to run the algorithm. It failed only after some $10^4$ symbols, and till that point used about $12 \times 10^4$ bits, thus averaging at about **12** bits per symbol, **even worse than fixed length encoding**.

To fix the error as explained above, "u" or location in the search window, which is of fixed length was used to encode the new symbol obtained in the look-ahead window.

"n" or the unary binary encoded length of the found subarray of look-ahead window was changed to max length of look-ahead window plus 1.

This was done because unary binary cannot encode a 0. But since the length is more than the length of look-ahead window possible, an error can be detected by the decoder which then knows that the next symbol is a new symbol of the length of fixed length encoding.

After fixing the error as explained above, 16.67 million bits were used for 10^6 bits. This means **16.67 bits** were required per symbol.

It is all too clear that LZ77 is not a good match for our sequence as such even though it works wonders for text compression.

The github repository with the MATLAB code is at
https://github.com/Memester2112/Digital_Communication