BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI- HYDERABAD CAMPUS

LAB REPORT ON

"Advanced Digital Communication - Lab Project"

Submitted By

AYUSH JAIN (2020H1240079H)

Under the guidance of

Mr. Prashant Wali

Assistant Professor

EEE Department

BITS Pilani - Hyderabad Campus

Submission Date

10th June, 2021

INDEX

Advanced Digital Communication - Lab Project

I. AIM:

1. Submit a working code of the digital communication system that consists of an English source (attached in this mail as a text file), source encoder (of your choice), channel encoder, line coder (of your choice), modulator of your choice (BPSK, or QPSK or QAM), channel (AWGN), receiver (correlator receiver or matched filter) followed by a detector, channel decoder, source decoder.
2. Remove any other character (comma, full stop, a hyphen, number, and so on) other than English alphabets in the attached file. Note that the source contains both capital and small letters.
3. The output of the source decoder should be matched with the source text, and the symbol error rate should be tabulated against SNR values of -5dB, 0dB, 5dB, 10dB and 15dB (preferably averaged over multiple simulations).
4. A report should be prepared that contains all the details of the project like what method/mechanism/algorithm is used in each block. The report should also indicate the compression achieved by your source encoder and the table for symbol error rate for the SNR values indicated. The report should also indicate how to run the code.

II. Important Parameters and Assumptions:

- *No. of bits transmitted:* length(compressed source)*6
- *No. of Samples:* 200
- *Time Duration (Tb):* 500 units
- *Enerygy per bit (Eb):* 0.01
- *Amplitude of NRZ Bipolar Pulse is:* –root(Eb) if ‘0’ is transmitted.
- *Amplitude of NRZ Bipolar Pulse is:* root(Eb) if ‘1’ is transmitted.
- *Basis Function is:* root(2/Tb)*cos(2*pi.Fc.t).

- *Carrier Frequency(Fc):* 400 units
- *Language used to code:* Python 3
- *IDE:* Google Colab (https://colab.research.google.com/)
- *Source Encoder:* Fixed Length Encoding
- *Channel Encoder:* (k,1) Repetition Code (k = 3)
- *Line Coder:* Polar NRZ
- *Modulator:* BPSK
- *Channel:* AWGN
- *Demodulator:* Matched Filter
- *Channel Decoder:* Maximum Likelihood Rule

III. Steps involved to create Digital Communication System:

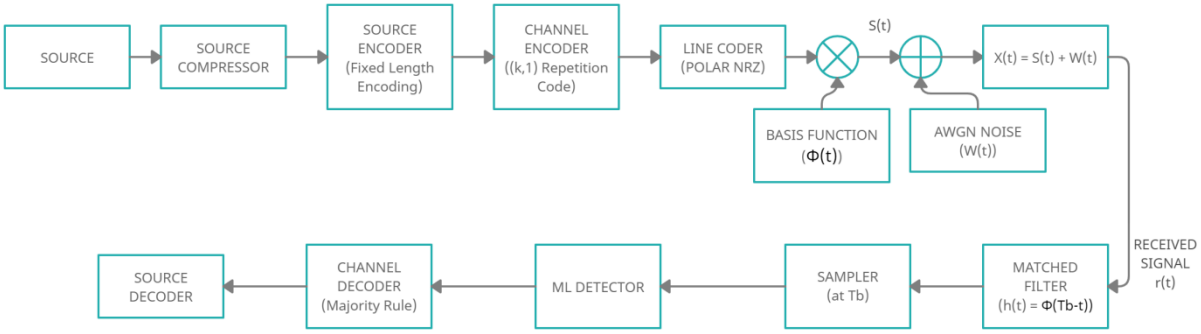

Fig. 1 DIGITAL COMMUNICATION SYSTEM

1. Firstly, the given text file is taken as a source.

A.) COMPRESS THE SOURCE:

2. Now, the given source file is compressed by removing other characters such as (‘,’, ‘.’, ‘-‘, numbers’0-9’, and so on) other than English alphabets in the attached file. The compressed source contains capital, small letters and space bars.

Algorithm:

STEP 1: START

STEP 2: Define compress file (file name):

- Open the given text file and read
- Declare a string to store the compressed string
- Set a loop to read every string in given file
- Set an inner loop to read every character of the string
- Apply if-else to replace the unwanted characters
- Add all the strings to form a single string
- Remove all the numbers from the obtained string
- Close the file
- Return compressed data

STEP 3: Calculate the compressed string length

STEP 4: STOP

B.) CALCULATE COMPRESSION RATIO:

3. Compression ratio is defined as the ratio between the uncompressed size and compressed size:

   Compression Ratio = Uncompressed Size/Compressed Size

Algorithm:

STEP 1: START

STEP 2: Define compression ratio (file name, compressed length):

- Open the given text file and read.
- Calculate the length of the uncompressed string.
- Now use:

  Compression Ratio = Uncompressed Size/Compressed Size
- Return Compression Ratio.

STEP 3: Print the compression ratio upto 3 decimal places.

STEP 4: STOP

Obtained **Compression Ratio** is **1.022**.

C.) SOURCE ENCODER:

4. Now, a dictionary is created for encoding the source. Fixed length encoding is used for encoding the source. Each symbol is represented by a 6 bit length code.

Algorithm:

STEP 1: START

STEP 2: Define source encoder (Compressed Data):

- Initialize a list to store the encoded sequence.
- Now, for each symbol append a 6 bit length code in encoded sequence list.
  Note: Small and Capital alphabets and space are encoded.
- Now, convert the encoded sequence into 1-D numpy array.
- Return encoded sequence.

STEP 3: STOP

D.) CHANNEL ENCODER:

5. Now, channel is encoded using the (k,1) repetitive code in which each bit is repeated k number of times.

Algorithm:

STEP 1: START

STEP 2: Define channel encoder (encoded sequence, k):

- Initialize a list to store the channel encoded sequence.
- Now, for each bit in the encoded sequence repeat k number of times and append in channel encoded sequence list.
- Now, convert the channel encoded sequence into 1-D numpy array.
- Return channel encoded sequence

STEP 3: STOP

E.) LINE CODING (POLAR NRZ):

6. In next step line is coded in which bit ‘0’ is represented by –root(Eb) and bit ‘1’ is represented by root(Eb). Take Eb = 0.01, Samples = 200.

Algorithm:

STEP 1: START

STEP 2: Define line coder (channel encoded sequence, Eb, samples):

- Initialize a list to store line coded sequence.
- Set a loop to read all channel encoded sequence values.
- If the bit is '0', then append (-root(Eb)) sample number of times.
- Else, append (root(Eb)) sample number of times.
- Now, convert the line coded sequence into numpy array
- Return line coded sequence

STEP 3: STOP

F.) PLOTTING FUNCTION:

7. For plotting the line coded sequence and basis function a common plotting function is created.

Algorithm:

STEP 1: START

STEP 2: Define plot sequence (line coded sequence, Tb):

- Set the time axis linspace.
- Give xlabel, ylabel.
- Then, plot the sequence. Library file used for plotting is Matplotlib
- Show the plot.
- Return time axis.

STEP 3: Set bit duration Tb = 500.

STEP 4: Call the function to plot the line coded sequence.

STEP 5: STOP

Obtained line coded sequence plot is:

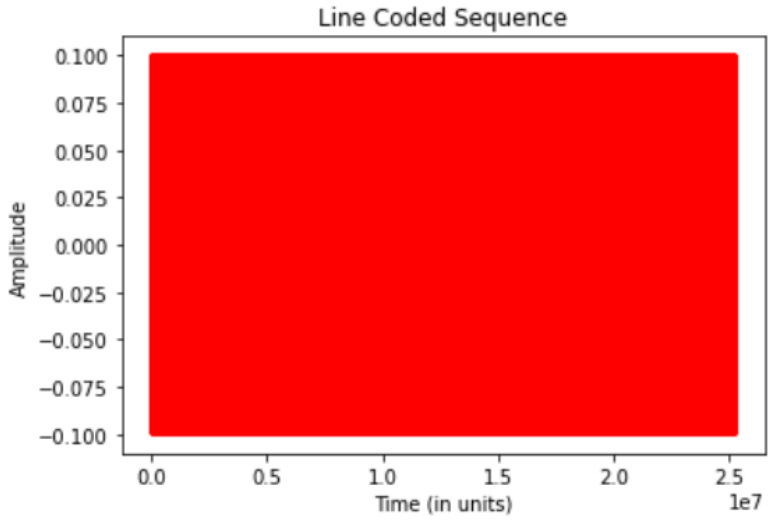

Fig. 2 Line Coded Sequence

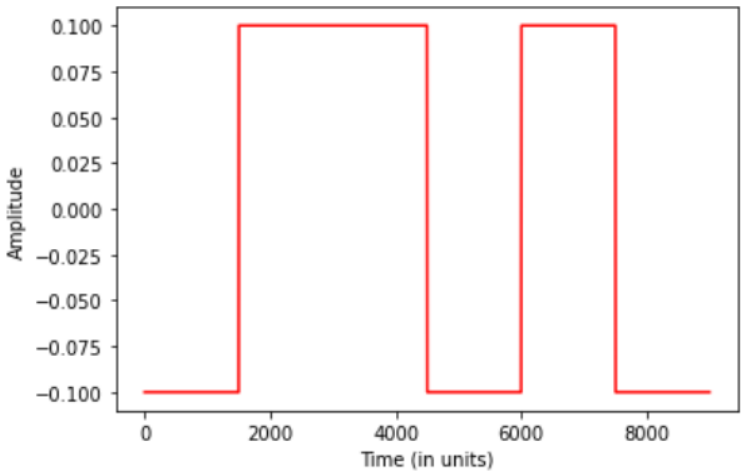

Fig. 3 POLAR NRZ Pulse for a symbol 'A'

G.) BASIS FUNCTION:

8. Now, a basis function is created which is defined as : (2/Tb)*cos(2*pi*fc*t).

Algorithm:

STEP 1: START

STEP 2: Define basis function (time axis):

- Calculate samples per second.
- Then, calculate seconds per sample.
- Set carrier frequence to 400 Hz.

- Apply basis function formula: (2/Tb)*cos(2*pi*fc*t).
- Return basis function sequence.

STEP 3: Call the function to store basis function sequence.

STEP 4: Plot the basis function using the plot sequence function defined in previous step.

STEP 5: STOP

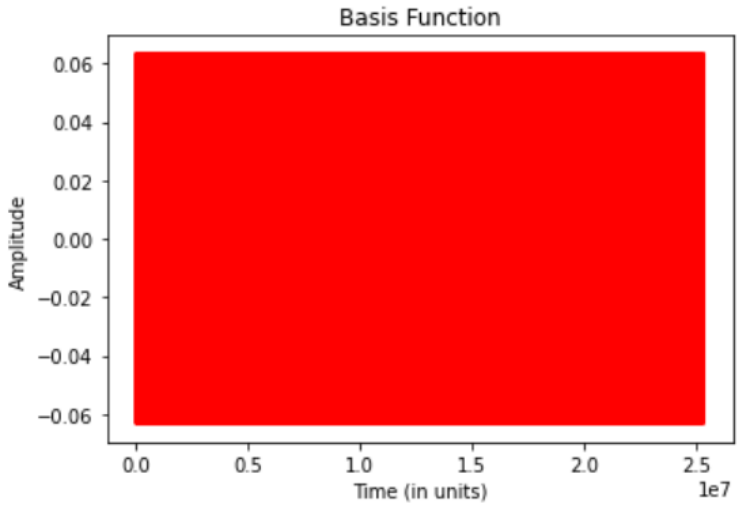

Fig. 4 Basis Function

H.) MULTIPLIER:

9. Now, basis function is multiplied with line coded sequence.

Algorithm:

STEP 1: START

STEP 2: Define multiplier (line coded sequence, basis function sequence):

- Store the multiplication of line coded sequence and basis function sequence in transmitter output.
- Return transmitter output.

STEP 3: STOP

I.) AWGN CHANNEL:

Additive White Gaussian Noise is a channel model which is a linear addition of white noise with a constant spectral density and a Gaussian distribution function of amplitude. This model does not account fading, non linearity, dispersion or interference but it

produces simple mathematical models which are useful to gain the insight into the underlying behaviour of a system. The channel capacity 'C' for AWGN channel is given by:

$$\mathbf{C = B \log_2(1 + S/N) \text{ bits/sec}}$$

where, S/N is Signal-toNoise ratio, B is bandwidth in Hz and is measured in bits per second.

10. Now, the given transmitted signal passed through AWGN channel. Function for AWGN channel is given below:

Algorithm:

STEP 1: START

STEP 2: Define awgn (transmitter output, snr_dB):

- Convert snr_dB to SNR_linear.
- Initialize a list to store awgn output
- Now, for each bit calculate its power.
- Then, calculate noise spectral density.
- Now, add noise to each bit of the transmitted output.
- Append the calculated noisy signal to the list.
- Return channel output

STEP 3: STOP

J.) MATCHED FILTER AND SAMPLER:

11. Now, the channel output passed through the matched filter and sampled at every time duration Tb = 500.

Algorithm:

STEP 1: START

STEP 2: Define matched filter (channel output):

- Initialize a list to store matched filter output.

- Now, convolve every received bit with the basis function of duration Tb.
- After that sample the matched filter output at every time duration Tb.
- Return matched filter sampler output.

STEP 3: STOP

K.) ML DETECTOR:

12. After sampling the matched filter sampler output is given to the ML detector, in which it detects the binary digit is '0' or '1' by seting boundary to '0'.

Algorithm:

STEP 1: START

STEP 2: Define ml detector (matched filter sampler output):

- Initialize a list to store ml detector output.
- For each bit in matched filter sampler output:
  - Check if the value is less than zero.
  - Append 0 to ml detector output list.
  - Else append 1 to ml detector output list.
- Now, convert the ml detector output list to numpy array.
- Return ml detector output

STEP 3: STOP

L.) CHANNEL DECODER:

13. Now, the ml detector output is passed through the channel decoder where maximum likelihood rule is applied to decode.

Algorithm:

STEP 1: START

STEP 2: Define channel decoder (ml detector output):

- Initialize a list to store channel decoded output.
- For every 'k' bits in ml detector output apply maximum likelihood rule.
- Now, convert the channel decoded output list to numpy array.

- Return channel decoded output

STEP 3: STOP

M.) SOURCE DECODER:

14. Now, the channel decoded output is passed through source decoder where every 6 bits are compared with dictionary.

Algorithm:

STEP 1: START

STEP 2: Define source decoder (channel decoded output):

- Initialize numpy character array to store the decoded output.
- For every 6 bits check which character is corresponding to them.
- Store that character in numpy character array.
- Now, convert the numpy char array to list.
- Then convert list element to string.
- Return decoded output

STEP 3: STOP

N.) SYMBOL ERROR PROBABILITY:

15. Now, calculate the symbol error probability by comparing the strings.

Algorithm:

STEP 1: START

STEP 2: Define symbol error prob (str1, str2, file name):

- Open the text file.
- Read the text file.
- Initialize the counter.
- Compare both the strings and store number of unmatched characters in counter.
- Calculate symbol error probability by taking the ratio of number of error bits and length of the uncompressed string.
- Return symbol error probability.

STEP 3: STOP

16. Now, all the functions are called before AWGN channel once.
17. After that for different SNR values ranging between -5dB to 18dB in the steps of 0.1dB all remaining functions are called.
18. Symbol error probability is calculated for each SNR value.
19. After that SNR vs Symbol Error Rate Plot is plotted.

    Obtained Plot is:

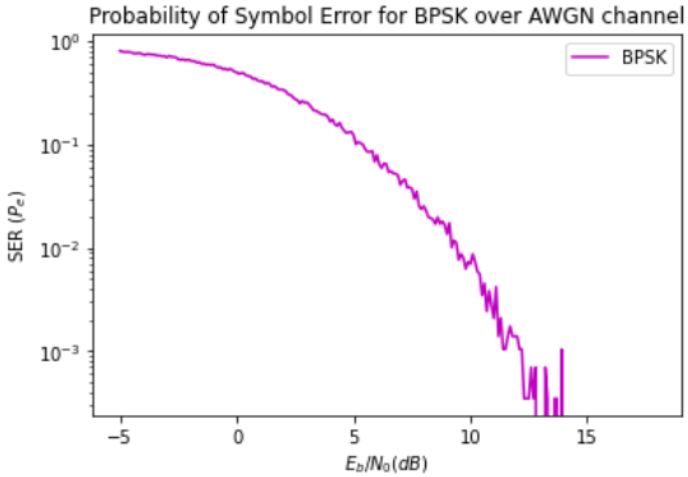

Fig. 5 SNR (in dB) vs Symbol Error Probability Plot

20. Now, tabulate Symbol Error Rate values for 5 different SNR values.

SNR (in dB)	SER
-5	0.7768
0	0.5021
5	0.1227
10	0.0076
15	0

Fig. 6 Symbol Error Rate for 5 different SNR values

IV. Steps to run the code:

1. Open google colab using this link: https://colab.research.google.com/
2. Use that mail ID to which the access is given to run the code. Direct link to the code is given below:

3. https://colab.research.google.com/drive/1GBQM0LF4ZjswEPjUcOVY_hWxlf_pkbvj?usp=sharing
4. Now, upload the text file as shown in the steps below:

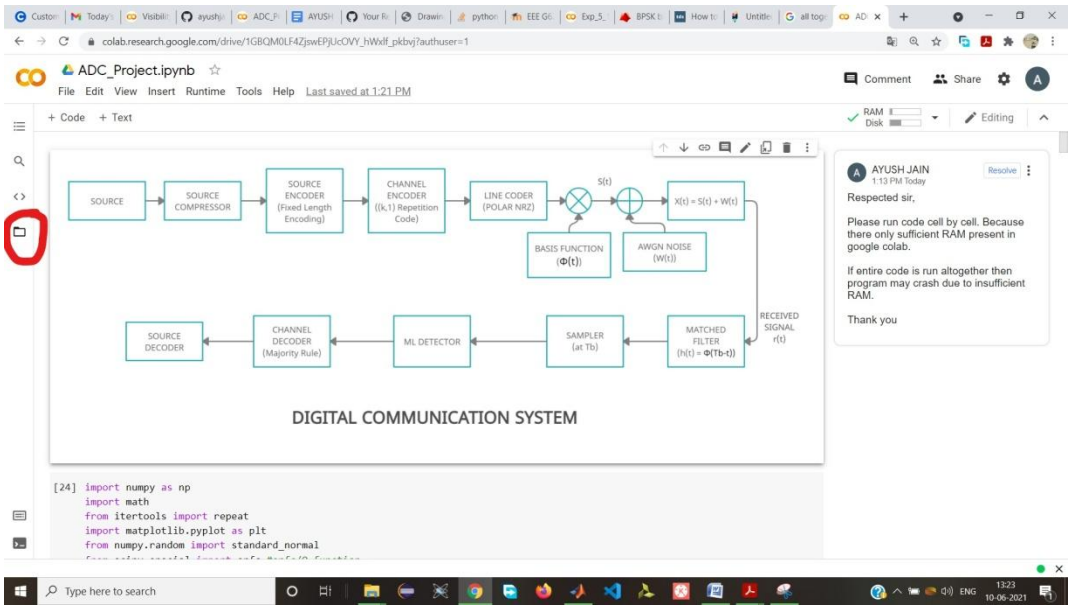

Fig. 7 Step 1

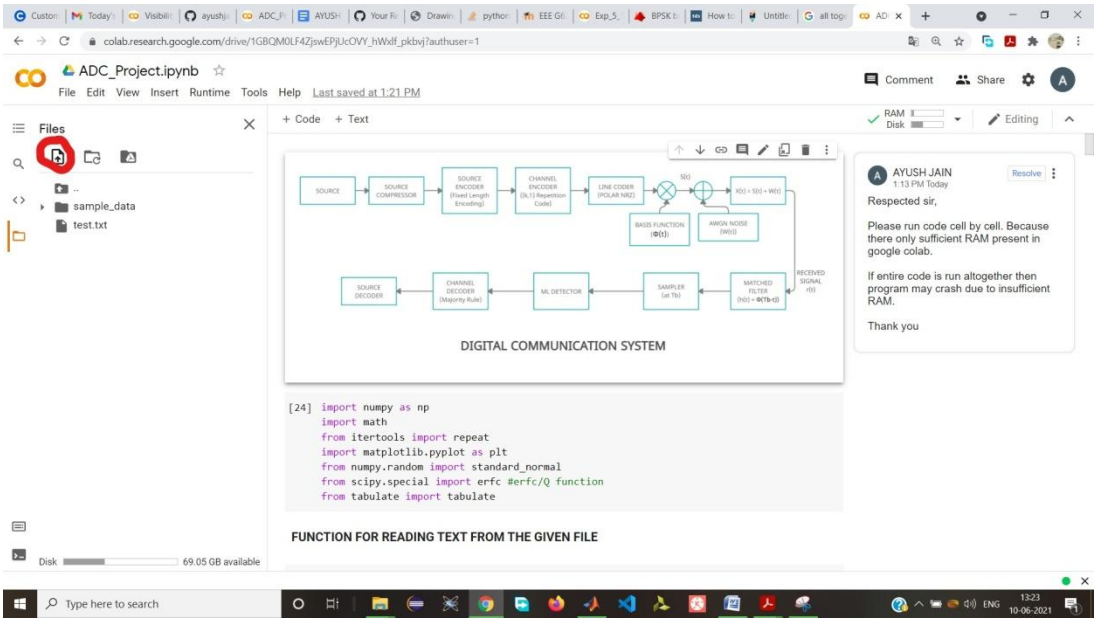

Fig. 8 Step 2

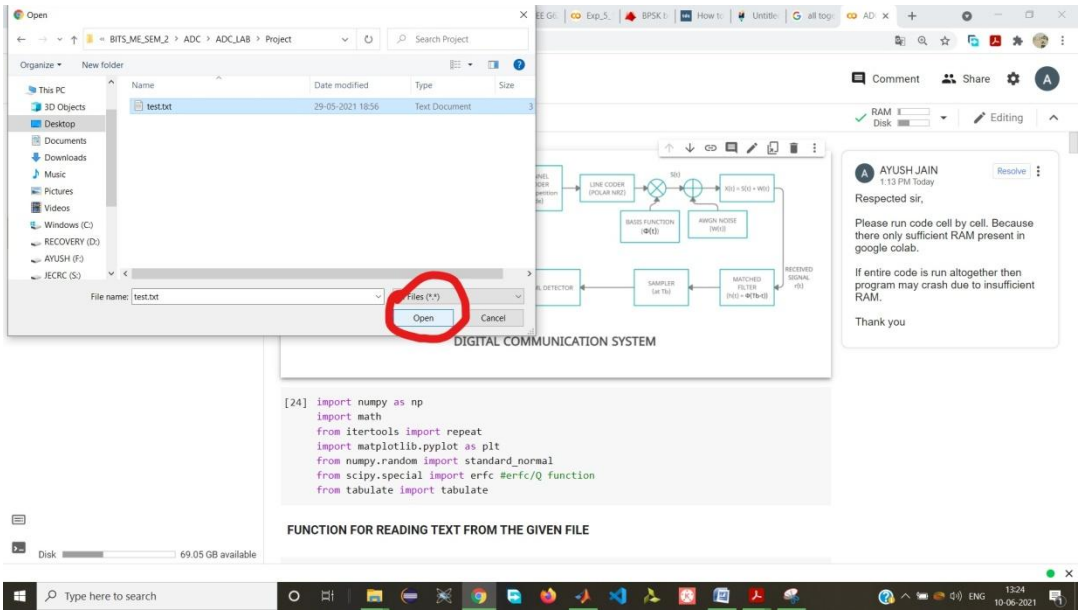

Fig. 9 Step 3

5. Run each cell as shown in figure below:

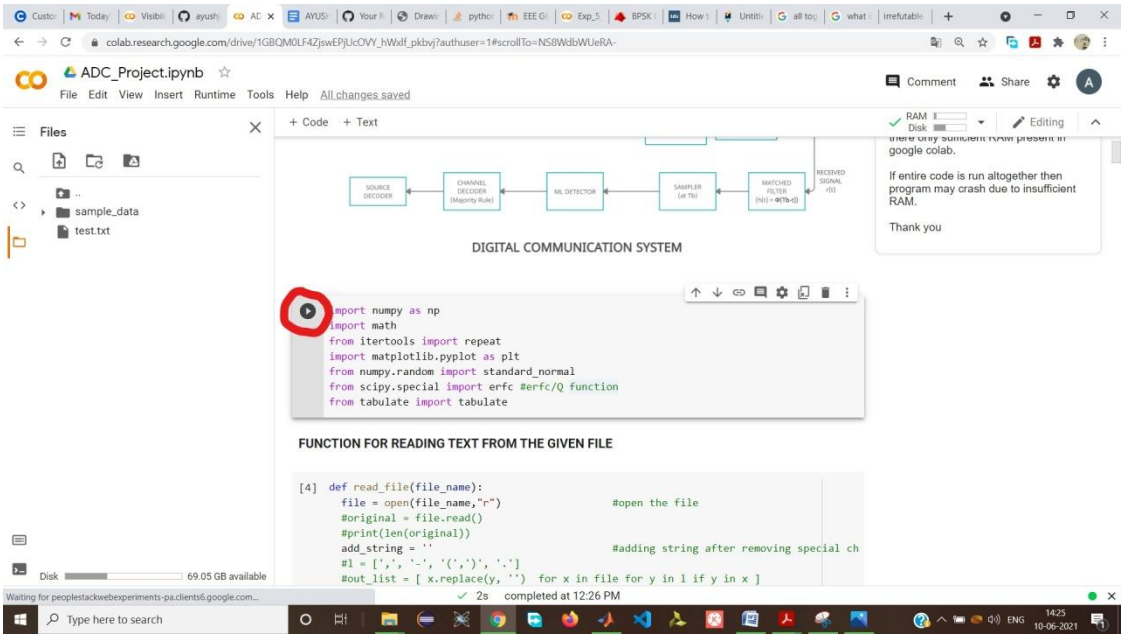

Fig. 10 Run Program

V. CONCLUSION:

- We have successfully created a digital communication system using BPSK modulator and Matched Filter detector.
- Observed that symbol error rate is decreasing as the SNR value is increasing.

- It is also observed that the symbols having high energy are less prone to noise.