A Complete Digital Communication System

MD MAHMUDUL ALAM

STUDENT ID: 1406131

Bangladesh University of Engineering and Technology

Bangladesh University of Engineering and Technology

EEE 438

Digital Communication Laboratory

A report on

A Complete Digital Communication System

PREPARED FOR

Dr. Md. Forkan Uddin
Associate Professor
Dept. of EEE, BUET

Rajat Chakraborty
Lecturer
Dept. of EEE, BUET

PREPARED BY

Md. Mahmudul Alam
ID: 1406131

Level- 4 / Term- 2

Department of Electrical & Electronic Engineering

Date of submission: 11/02/2019

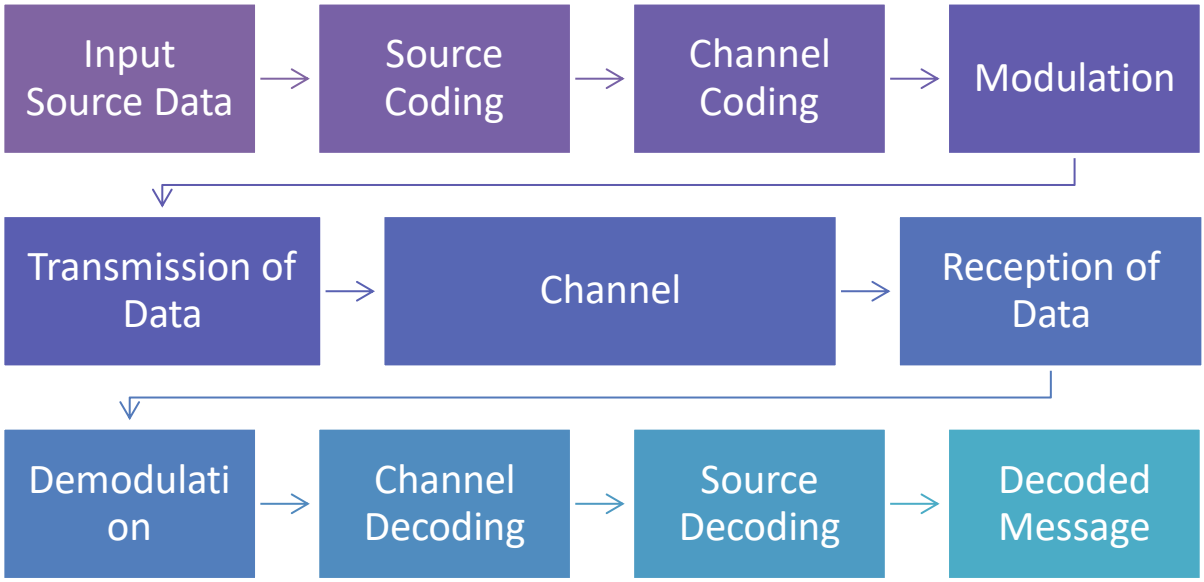
Contents

Introduction

- Q1: A succinct theoretical description of the blocks discussed above and how it is implemented in the MATLAB code.

Digital communication is a mode of a communication system where the information or the thought is encoded digitally as discrete signals and electronically transferred to the recipients. Digital communication is one of the most commonly used modes of the communication system of the modern age. A complete digital communication system consists of multiple blocks. A step by step block diagram representation of the digital communication system is shown below and the aim is to build each of the building blocks of the digital communication system.

Block Diagram Representation of a Complete Digital Communication System:



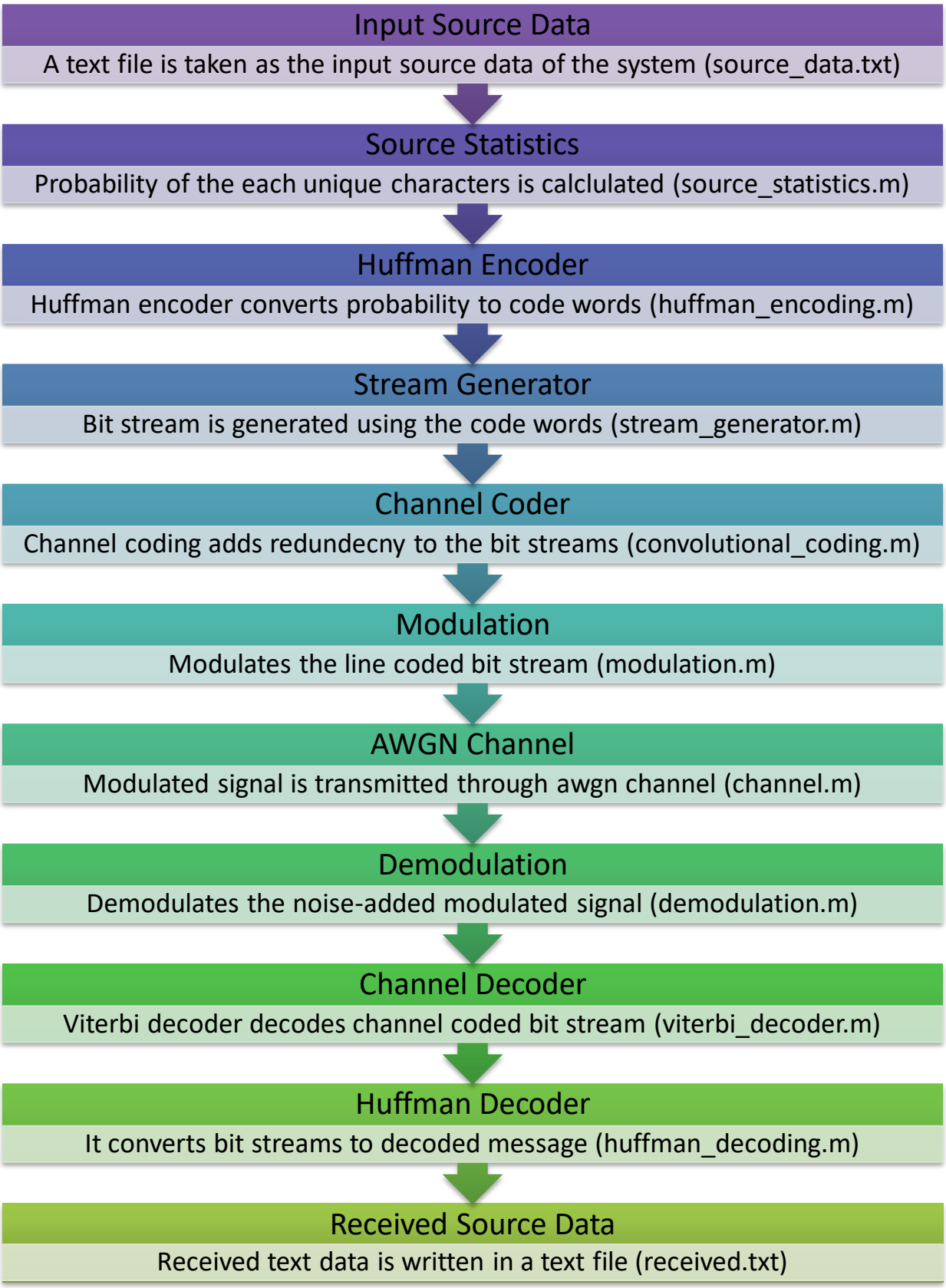
The block diagram articulately shows all the building blocks of a complete digital communication system. The build the system the following blocks need to be built using the mentioned technique:

- Source Coding: Huffman encoding
- Channel Coding: Convolutional encoding
- Modulation: Amplitude Shift Keying (ASK), and Phase Shift Keying (PSK) modulation
- Channel: Additive White Gaussian Noise Channel
- Demodulation: ASK and PSK demodulation
- Channel Decoding: Viterbi decoding
- Source Decoding: Huffman decoding

Before going through all of the details of each block, first, a step by step data flow diagram of each block is shown below additionally mentioning the gist of each block along with their

code file name and followed by a succinct description of each of the blocks including their working principle and MATLAB code snippet to show how they are implemented.

Data Flow Diagram:



➢ Input Source Data:

A text file having 247969 characters named 'source_data.txt' is taken as the input of the system. All the block operation will be performed on this text data. As text data cannot be transmitted directly, each character needs to be represented by binary bit stream which is performed by huffman encoder. But at first, the statistics of the text data such as the number of total unique characters and their corresponding probability need to be calculated.

Code Snippet:

```
file = fopen('Data/source_data.txt');
text = fread(file, '*char')';
fclose(file);
```

➢ Source Statistics:

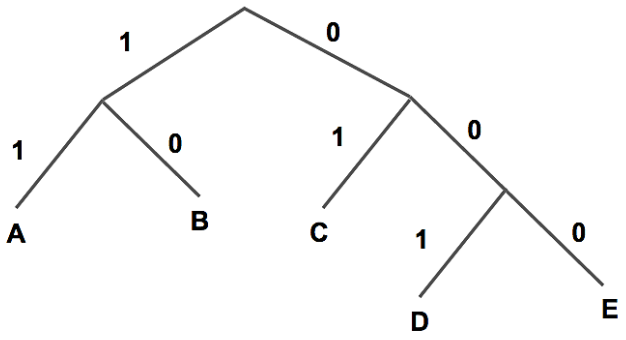
Huffman encoder converts characters to set of bits based on their occurrence probability of in the text file. Thus, the most probable characters can be represented via the lowest numbers of the bit and vice versa. First, total unique characters of the text data and their occurrence time is calculated using MATLAB built-in 'unique' and 'histc' function and later dividing the total occurrence time by total text file length the probability of each character is calculated.

Code Snippet:

```
unique_symbol = unique(text);
count_symbol = histc(text, unique_symbol);
probability = count_symbol / length(text);
```

➢ Huffman Encoder:

Huffman encoder represents each unique character of the text file by a set of bits using their occurrence probability. Huffman coding uses a specific method for choosing the representation for each symbol, resulting in a prefix code. The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source character. The algorithm derives the variable length table from the estimated probability of occurrence for each unique character, adds the probability of lowest characters and creates two branches and one node. This process goes on until all probability adds up to one. Then tracebacking bit representation of each branch code word for each character can be found. An example is given below:



Code words: A = 11, B = 10, C = 01, D = 001, E = 000

Code Snippet:

```
for j = 1:n
    if x(j, i+1) == min1
        code_word(j) = strcat('0', code_word(j));
    elseif x(j, i+1) == min2
        x(j, i+1) = min1;
        code_word(j) = strcat('1', code_word(j));
    end
end
```

➢ Stream Generator:

The Huffman encoder represents each unique character by a minimum number of bits and the next step is to convert the whole text file to bitstream by concatenating each character bit sequence. First, the bit representation of a character is searched and then it is concatenated to the output bitstream and thus the bit stream of the overall text file is generated.

Code Snippet:

```
for i = 1:increment:length(msg)
    index = strfind(unique_symbol, msg(i:i+increment-1));
    bit_stream = [bit_stream char(code_word(index))];
end
```

➢ Channel Coding:

Channel coding adds redundancy to the existing bit stream so that at the receiver end of the communication system noise added transmitted bit stream from the channel can be recovered at the most accurate possible way. The convolutional channel coding is implemented in this project. It is a type of error-correcting code that generates parity symbols via the generator matrix to a data stream. The sliding application represents the 'convolution' of the encoder over the data, which gives rise to the term 'convolutional coding'. In convolutional code, each block of k bits is mapped into a block of n bits to be transmitted over the channel, but these n bits are not only determined by the present k information bits but also by the previous information bits. It is implemented using the MATLAB built-in ‘conv2’ function by directly convolving bit stream and generator matrix and further data processing according to criteria.

Code Snippet:

```
y = conv2(bit_stream, G);
y = y(:, 1:end-2);
y = rem(y, 2);
[row, col] = size(y);
y = reshape (y, 1, row * col);
```

➢ Modulation:

A signal is modulated by multiplying the signal with a high-frequency carrier signal. There are 3 basic types of modulation: Amplitude modulation, Frequency modulation, and Phase modulation. Among them, their digital counterfeit BASK (Binary Amplitude Shift Keying), BPSK (Binary Phase shift Keying), and QPSK (Quaternary Phase Shift Keying) are implemented in the project. Modulation is accomplished because of:

- To reduce the antenna length
- To separate signal from different transmitters
- To avoid mixing of signals
- To increase the transmission range
- To multiplex and efficient use of the channel
- To improve noise immunity
- To improve the quality of reception

In the modulation process, first, the bit stream is line coded and then they are modulated according to the name of the modulation that is asked for. Three types of modulation are performed in the project. A short description of all of them is given below:

BASK:

The digital data to be transmitted is binary number 0 and 1. Two amplitudes are used to directly represent the data, either 0 or 1. In this case, the modulation is called binary amplitude shift keying or BASK.

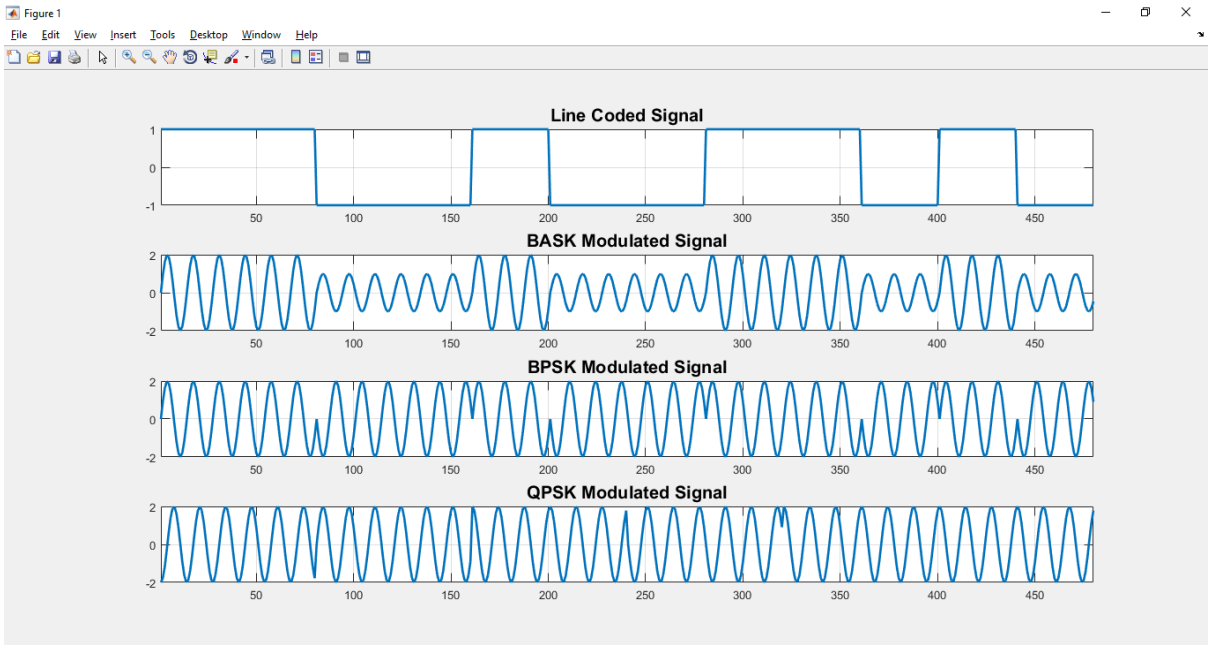
BPSK:

The most straightforward type of PSK is called binary phase shift keying (BPSK), where binary 0 and 1 represented with the use of two phases 0 and 180 degrees. As phase is much less susceptible to noise BPSK performs far better than BASK.

QPSK:

QPSK is a modulation scheme that allows one symbol to transfer two bits of data. There are four possible two-bit numbers (00, 01, 11, 10), and consequently, four phases are needed to be separated by 90 degrees to represent each symbol.

The following figure shows a visual example of those modulation techniques.



Code Snippet:

```
% BASK
carrier = sin(2*pi*freq*time);
line_code = a1 * line_code + a0 .* (line_code==0);
modulated = line_code .* carrier;
% BPSK
carrier = sin(2*pi*freq*time);
line_code = a .* (line_code == 1) + (-a) .* (line_code == 0);
modulated = line_code .* carrier;
% QPSK
for i = 1:bn:length(bit_stream)
    sym = bit_stream(i:i+bn-1);
    index = find(ismember(gray_code, sym, 'rows'), 1);
    val = bi2de(gray_code(index, :));
    phi = 2 * pi * val / M;
    modulated = [modulated amp(1)*sin(2*pi*freq*t + phi)];
end
```

➢ Channel:

After modulation, the signal is transmitted through a communication channel. A channel can be coaxial cable, optical fiber or a radio link. An Additive White Gaussian Noise (AWGN) channel is modeled in the simulation process. The channel is additive because it is added to any noise that might be intrinsic to the information system. The white name refers to the idea that it has uniform power across the frequency band for the information system and finally, Gaussian because it has a normal distribution in the time domain with an average time domain value of zero.

Code Snippet:

```
received = awgn(modulated, snr, 'measured');
```

➢ Demodulation:

At the receiver end, the first thing after receiving a signal is to demodulate the received signal. Demodulation is extracting the original information-bearing signal from a carrier wave. The process is accomplished by multiplying the received signal by the same carrier it original signal is modulated and then taking averages and comparing to it to a threshold value. Corresponds to modulation, BASK, BPSK, and QPSK demodulation process is added to the simulation environment. After demodulation, the receiver attains the transmitted bit stream.

Code Snippet:

```
r = received .* sin(2*pi*freq(1)*t);
r = reshape(r, k, n);
yd = mean(r);
% BASK
threshold = (a0 + a1) / 4;
bit_stream = (yd >= threshold);
% BPSK
threshold = 0;
bit_stream = (yd >= threshold);
% QPSK
```

```
r1 = received .* sin(2*pi*f*t);
r2 = received .* cos(2*pi*f*t);
ai = mean(r1);
aq = mean(r2);
phi = (0:M-1) * 2 * pi ./ M;
distance = (A*cos(phi) - 2*ai).^2 + (A*sin(phi) - 2*aq).^2;
[~, min_ind] = min(distance);
min_ind = min_ind - 1;
sym = de2bi(min_ind, bn);
bit_stream = [bit_stream sym];
```

➢ Channel Decoding:

As redundant bits are added to the transmitted bit stream before modulation, the same way after demodulation original bit stream has to be recovered from the bit stream received after demodulation. A decode convolutional coded bit stream, Viterbi decoder is implemented. The Viterbi algorithm is the most resource-consuming, but it does the maximum likelihood decoding of the convolutional coding process. Viterbi decoding was developed by Andrew J. Viterbi and published in the paper "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm". [*view*]

The decoding algorithm uses two metrics. The branch metric (BM) and the path metric (PM). The branch metric is a measure of the “distance” between what was transmitted and what was received and is defined for each arc in the trellis diagram. The path metric is a value associated with a state in the trellis it corresponds to the Hamming distance over the most likely path from the initial state to the current state in the trellis. The path with the smallest Hamming distance minimizes the total number of bit errors and is most likely when the BER is low. The key insight in the Viterbi algorithm is that the receiver can compute the path metric for a pair incrementally using the path metrics of previously computed states and the branch metrics.

Code Snippet:

```
rcvd = y(:, i)';
% calculating hamming distance
branch_metric_0 = sum(abs(bsxfun(@minus, output_for_zero, rcvd)), 2);
branch_metric_1 = sum(abs(bsxfun(@minus, output_for_one, rcvd)), 2);
branch_metric = [branch_metric_0 branch_metric_1];
pos_out = [];
for j = 1:size(states, 1)
    pos_out = [pos_out states(j, 2)];
    a = path_metric(alpha(j), i) + branch_metric(alpha(j), states(j,
1)+1);
    b = path_metric(beta(j), i) + branch_metric(beta(j), states(j,
1)+1);
    path_metric(j, i+1) = min(a, b);
end
[~, min_ind] = min(path_metric(:, i+1));
out = [out pos_out(min_ind)];
```

➢ Huffman decoding:

Channel decoding returns the original bit stream of the transmitted text data which is created by huffman encoder. The next step is to decode the huffman encoded bit stream to recover the decoded message. For that purpose, original unique characters and their code words are used. The message is decoded by replacing a bit sequence by the characters of the unique character list.

Code Snippet:

```
for i = i_min:length(bit_stream)
    if isempty(find(strcmp(code_word, char(bit_stream(ptr:i) + '0')),
1)) ~= 1
        ind = find(strcmp(code_word, char(bit_stream(ptr:i) + '0')),
1);
        decoded_msg = [decoded_msg char(unique_symbol(ind))];
        ptr = i + 1;
        i = i + i_min;
    end
end
```

➢ Received Source Data:

The received source text data is written in a text file named ‘received.txt’.

Code Snippet:

```
f = fopen('Data/test.txt','w+');
fprintf(f, decoded_msg);
fclose(f);
```

MATLAB simulation of the overall system: [*code file: main_code.m*]

A MATLAB simulation of the above block diagram is performed using the 'source_data.txt' file for the following properties listed in the table and the input text file is successfully recovered at the receiver end and written in another text file named 'received.txt'.
The system properties are used:

Modulation Name	BPSK
Samples per bit	40
Bitrate	1000
Carrier frequency	1 V
Carrier amplitude	1000 Hz
Signal to noise ratio	10 dB
Generator matrix for channel coder	[1 1 1; 1 0 1]
No. of bit shift in the register	1

Command Window Output:

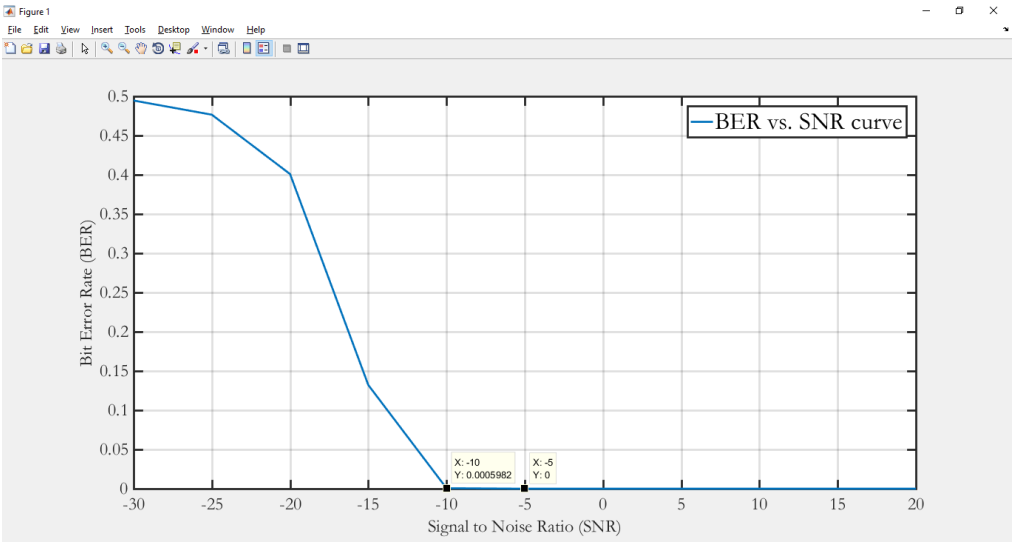
```
Command Window
  Reading data:Elapsed time is 0.001877 seconds.
  Source statistics: Elapsed time is 0.015120 seconds.
  Huffman encoding: Elapsed time is 0.072447 seconds.
  Stream generator: Elapsed time is 1.157318 seconds.
  Channel coding: Elapsed time is 0.021386 seconds.
  Modulation: Elapsed time is 4.065147 seconds.
  Channel: Elapsed time is 1.639971 seconds.
  Demodulation: Elapsed time is 3.728550 seconds.
  Channel decoding: Elapsed time is 17.002255 seconds.
  Huffman decoding: Elapsed time is 4.847662 seconds.
  Writing data: Elapsed time is 0.013435 seconds.
  Total execution time: Elapsed time is 32.568118 seconds.
  Bit Error Rate (BER): 0
fx >>
```

From reading the 'source_data.txt' file to write the received data to a 'received_data.txt' file requires only 32.56s.

Effect of Noise

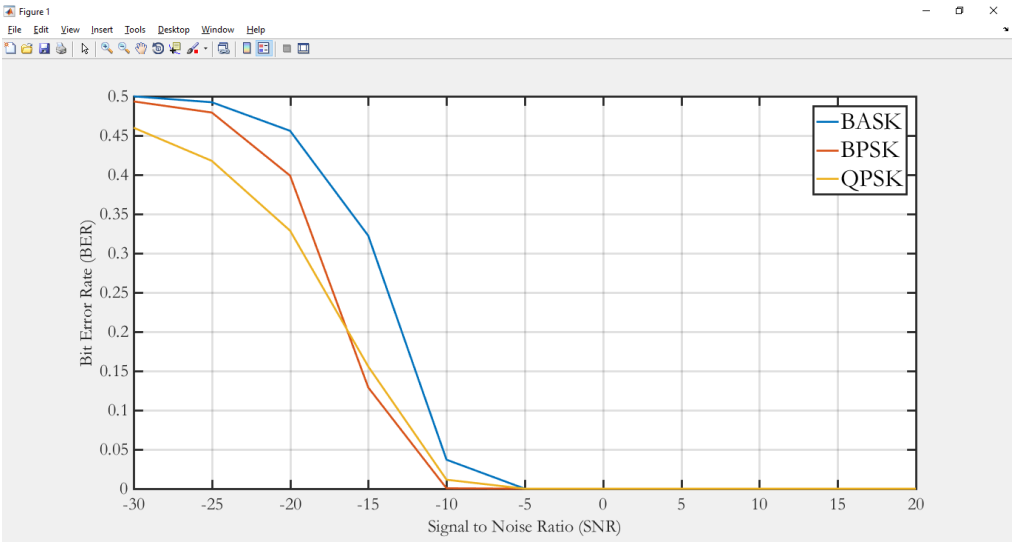
- Q2: The effect of channel noise variance on the error rate (=number of wrong symbols/total number of symbols) of the digital communication system should be discussed. What is the maximum noise variance for which all the texts in source_data.txt and received.txt match? What is the minimum noise variance for which at least 40% of the symbols in the source and received text does not match?

The Bit Error Rate (BER) vs. Signal to Noise Ratio (SNR) curve is plotted for variation of SNR from -30dB to 20 dB using the 'source_data.txt' file. For different SNR value BER is calculated and from the Figure, it can be observed that between -10dB and -5dB SNR bit error has started. So, the recovered message will be erroneous. To successfully recover the transmitted message received message has to have SNR value ≥ -5dB.



[*code file: effect_of_noise.m*]

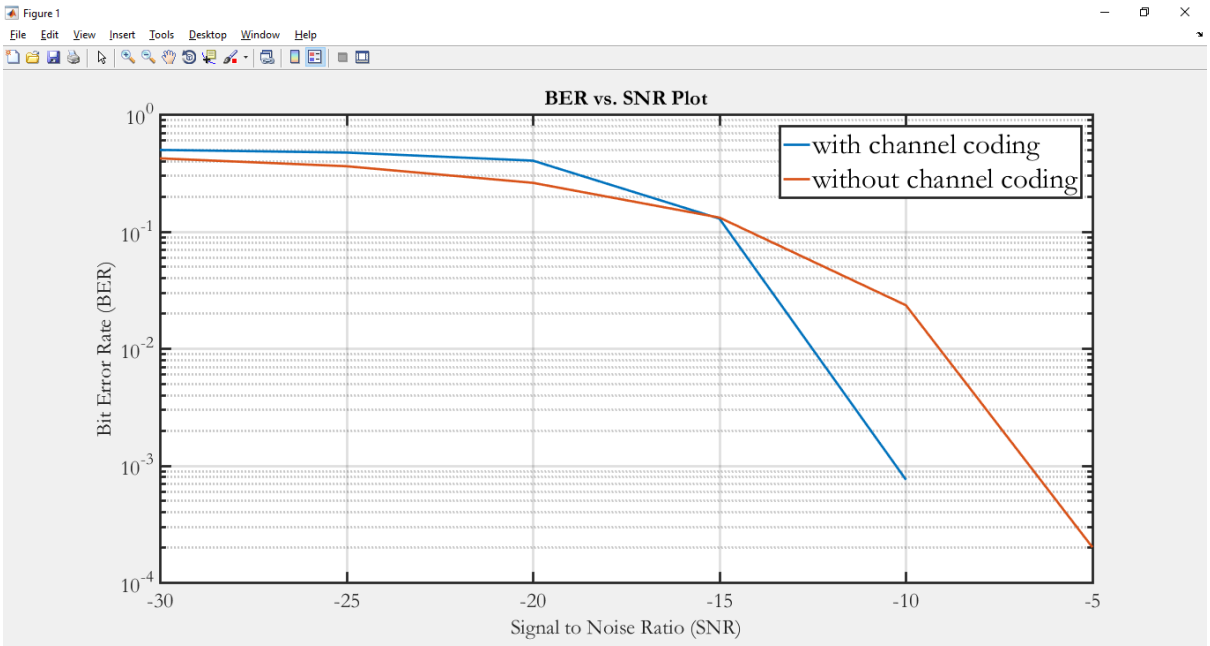
The maximum SNR is -5dB or noise variance = 3.16 to successfully recover the 'source_data.txt' file. From the following Figure, it can be found that minimum SNR for 40% text not having a match is for the 'QPSK' modulation and the SNR value is around -24dB.



Presence of Channel Coder

- Q3: Does the presence of channel coding (convolution coding) have any effect on the error rate of the system? If channel coding is turned off for the same level of noise variance, what happens to the error rate?

The BER vs. SNR semiology plot with the presence of channel coder and the absence of channel coder is shown in the following Figure. BPSK modulation is used here. From the plot is can be seen that up to -15dB BER value is lower for channel coder compared to the BER value without using channel coder. After -15 channel coder performance deteriorates and BER goes higher than without channel coder. This is because of the burst error. As convolutional channel coding is used here, encoding and decoding depend not only on the current bits but also on the previous bits.



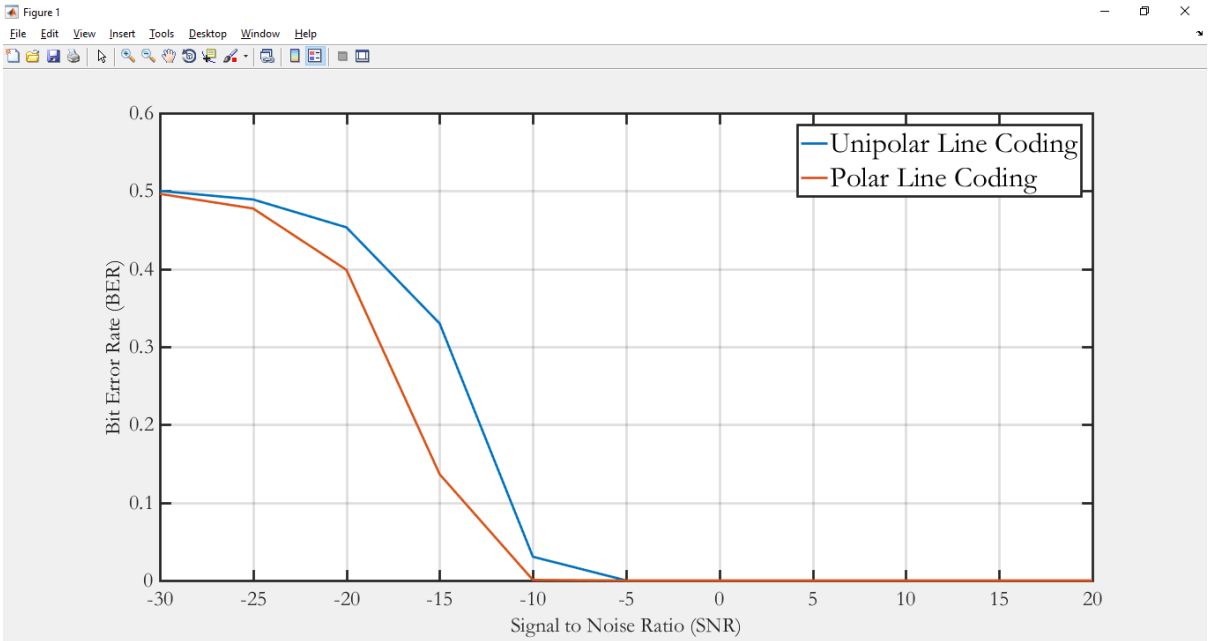
[*code file: ber_vs_snr.m*]

So, in conclusion, the presence of channel coder will lower the BER value up to a certain SNR range compared the without channel coder using up to certain SNR. If channel coding is turned off for the same level of noise variance, the bit error rate will increase.

Impact of Line Coding

- Q4: Does the error rate change for the same level of noise variance but different methods of line coding? What method leads to the lowest error rate and why?

The BER vs. SNR plot for two different line coding schemes Unipolar NRZ and Polar NRZ has been shown in the following Figure. It can be seen that the polar line coding scheme has much better noise immunity compared to a unipolar counterfeit. The BER vs. SNR curve always for polar line coding always stays below the curve for unipolar line coding. As polar line coding scheme mean stays around zero, so it is less susceptible to noise thus bit error rate decreases.

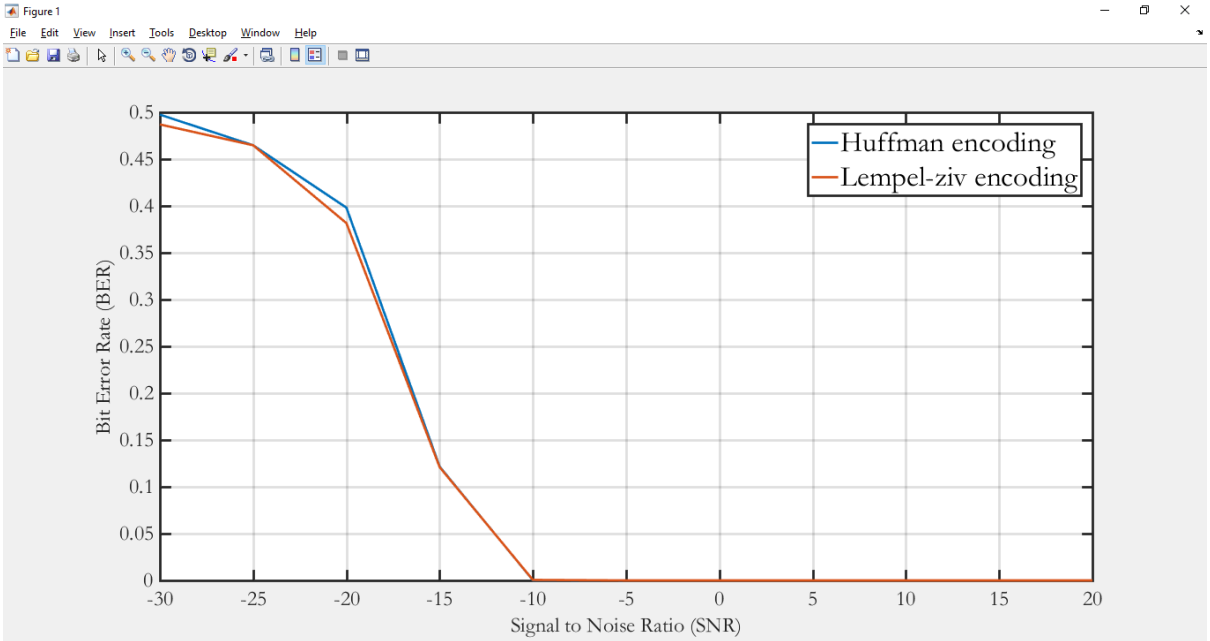


[*code file: line_coding_impact.m*]

Impact of Source Coding

- Q5: Does Huffman coding help to decrease the error rate? What happens to the error rate if Huffman coding is not used?

BER vs. SNR plot is shown for both Huffman encoding and Lempel-ziv encoding. As both of the source coder just only converts characters to the bitstream, so there is no significant improvement in terms of bit error rate. So, it can be said that Huffman encoding does not help to decrease the bit error rate. However, it is also worth noting that the BER in very low SNR value will be better in case of Lempel-ziv encoder.



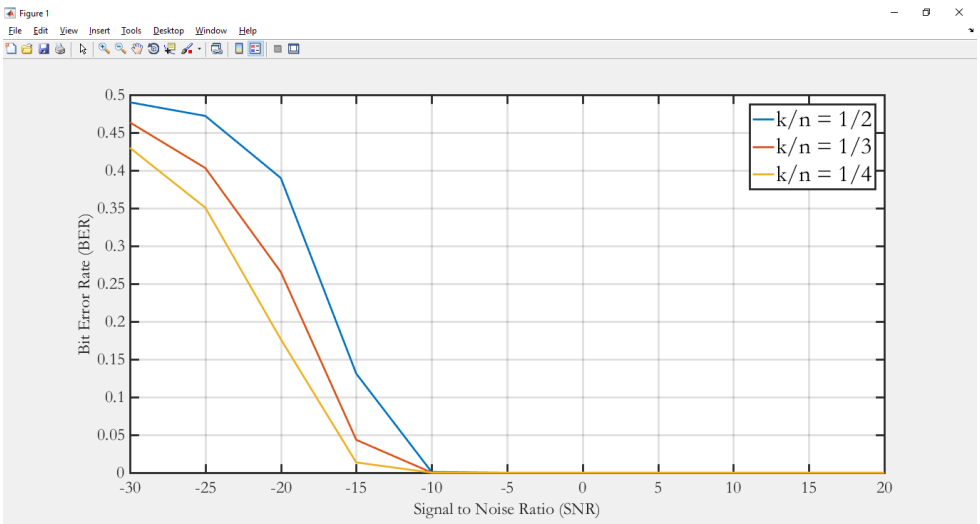
[*code file: impact_of_huffman.m*]

So, in conclusion, source coding has no significant impact on bit error rate performance.

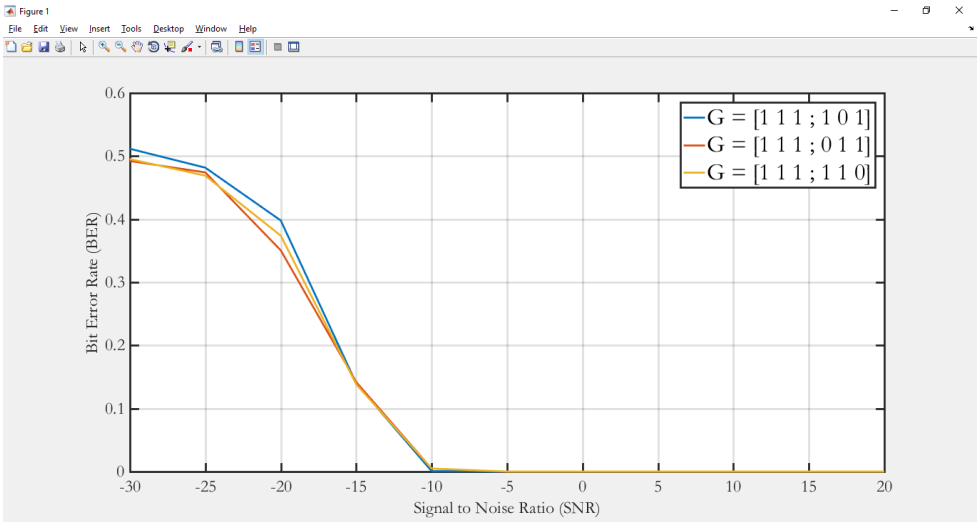
Impact of Generator Sequence

Q6: Does changing the generating sequence of the convolution code affect the error rate?

The code rate (k/n) ratio where k is the entry size and n is the output size is varied and their BER vs. SNR curve is shown. Keeping k fixed to 1, output size of the generator is varied to 2, 3 and 4. As from the Figure, it can be observed that code rate the lower the code rate, the better the BER performance.



Again, keeping the other parameter fixed, the generator sequence has been changed. Changing the generator sequence has little impact on the bit error rate performance in low SNR value. For SNR value lower than -15dB, there is some variation in the error rate which is not significant.



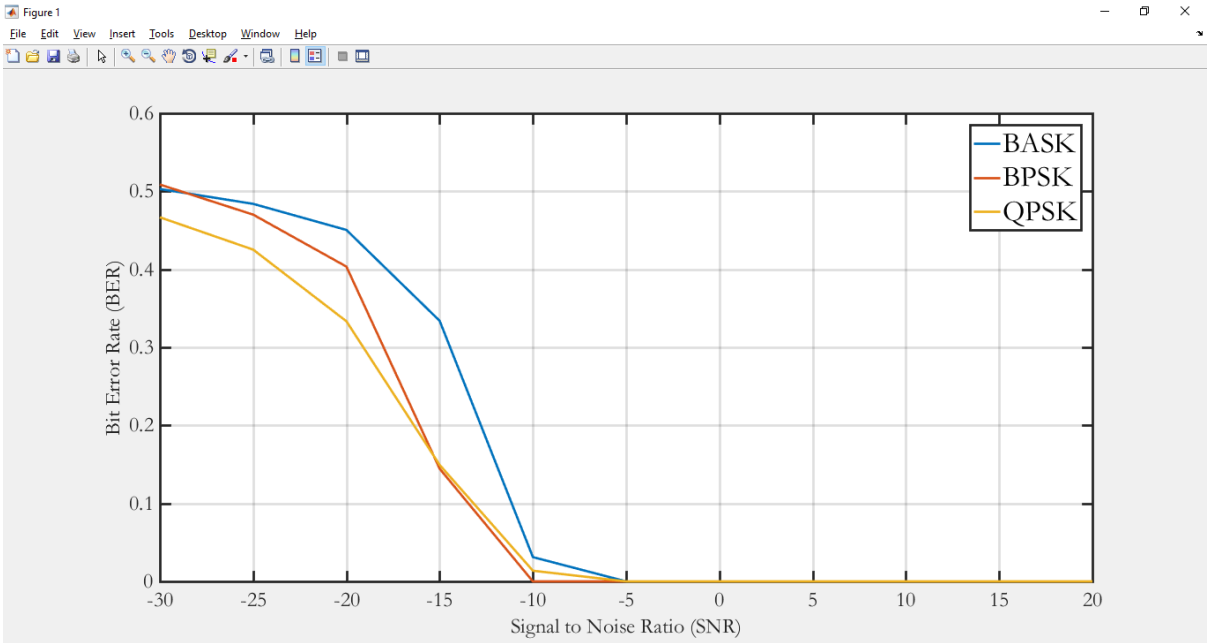
[*code file: effect_of_noise.m*]

In conclusion, lowering the code rate increases bit error rate performance but changing generator sequence has no impact.

Additional Investigation

Impact of Modulation Technique:

As three different modulation technique is implemented as a part of the project, it is worth finding the best modulation technique and to have a comparison. For the three-modulation technique, BER vs. SNR curve is plotted for G = [1 1 1; 1 0 1] with shift 1, in each case carrier amplitude and frequency is 1 (1 & 0 for BASK) and 3000 Hz and the following figure is observed.



[*code file: effect_of_noise.m*]

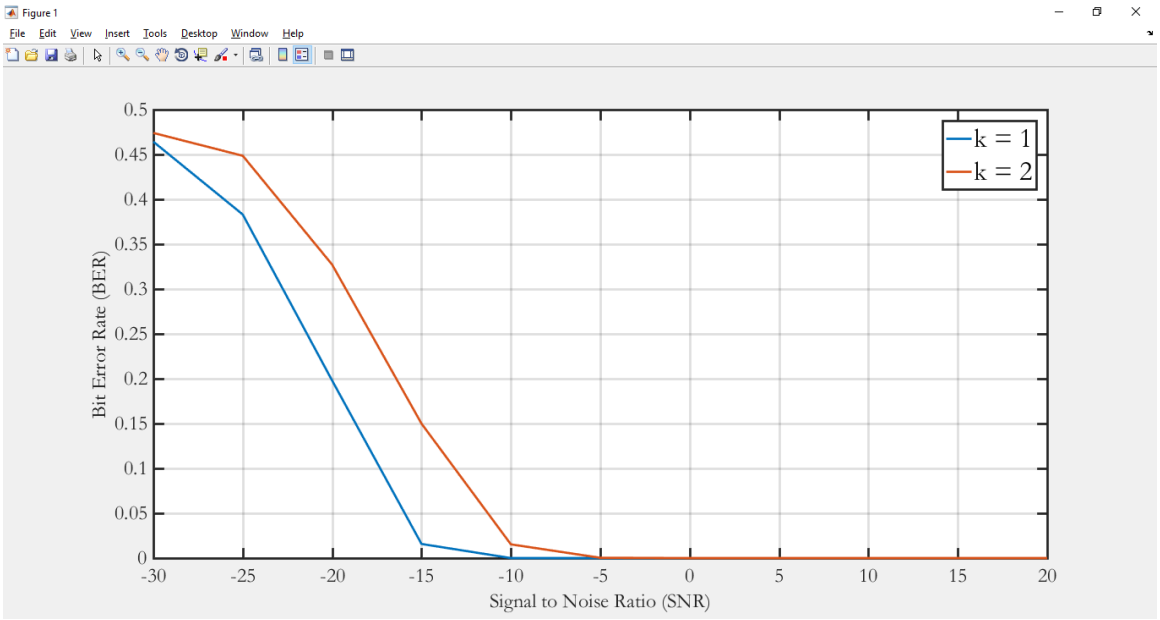
Performance Analysis:

SNR	*-20 dB*	*-15 dB*	*-10 dB*	*-5 dB*
BASK	0.4505	0.3340	0.0311	0.0000
BPSK	0.4035	0.1444	0.0000	0.0000
QPSK	0.3334	0.1493	0.0137	0.0000

From the Table after -15 dB, it is found that the order of modulation technique from the best to worst is BPSK>QPSK>BASK.

Impact of entry size in the register in BER performance

As bits input in the register can be more than 1, so BER performance is measured on the basis of entry size (k) in the register while keeping generator output size fixed at 4. The generator matrix is taken as: G = [1 1 1 ; 1 0 1 ; 1 0 0 ; 1 1 0] and entry size k is varied as 1 and 2. As k increases, BER performance deteriorates. Here, n is 4 which is kept fixed thus as k increases, (k/n) ratio increases too and hence for the same SNR value BER increases.



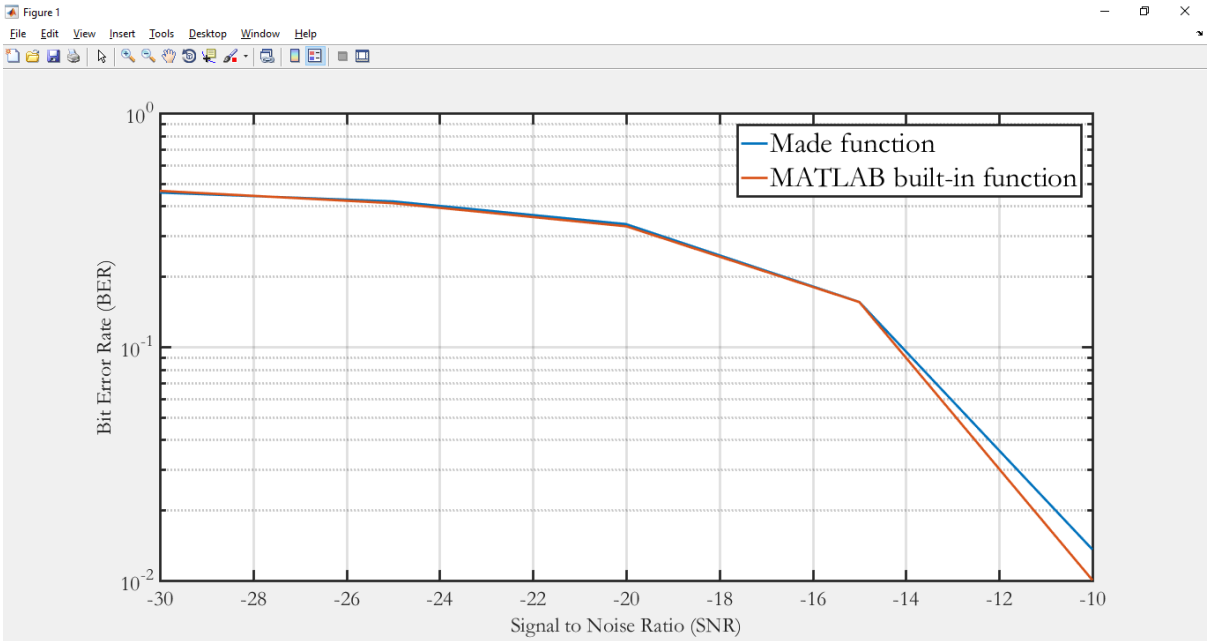
[*code file: effect_of_noise.m*]

So, increasing entry size while keeping the generator output fixed will deteriorate the BER performance of the system.

Comparison of Viterbi decoder function with MATLAB built-in function:

MATLAB has its own built-in convolutional encoder and Viterbi decoder function named as 'convenc' and 'vitdec'. As convolutional encoding and decoding are implemented here, it is worth finding the overall performance of the encoder and decoder compared to the built-in function.

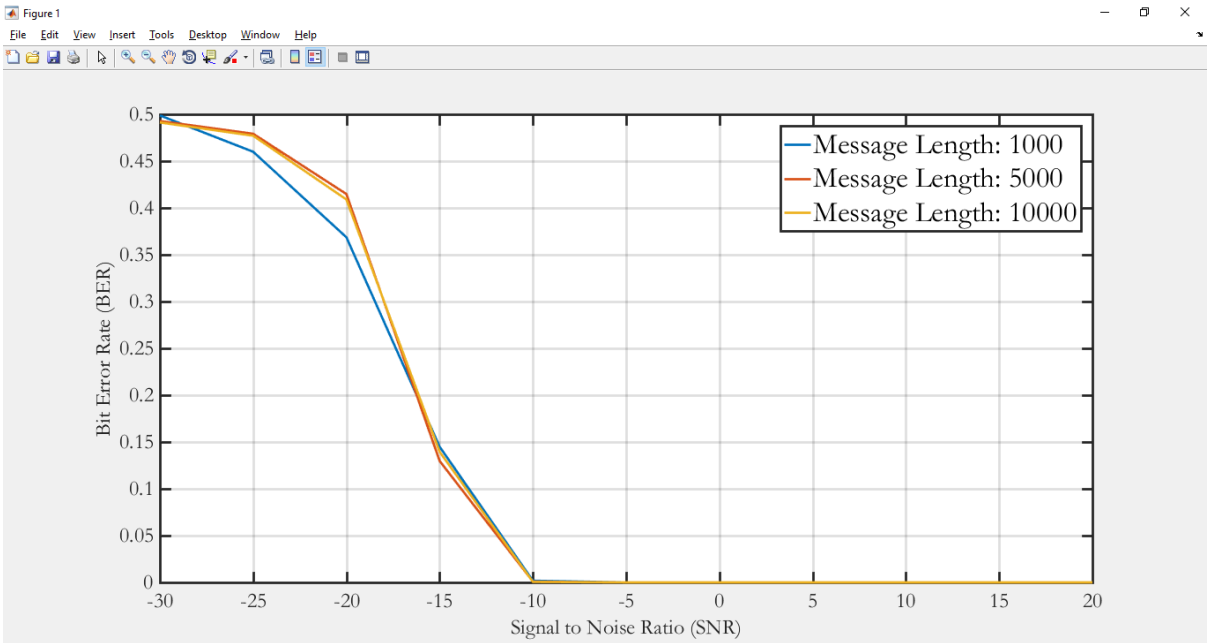
The following Figure shows the BER vs. SNR curve in the semilog grid where the BPSK modulation technique is used in the process. Here, from -30 dB to -15 dB error rate performance is almost same for both of the functions but in the higher SNR part from -15 dB to 10 dB MATLAB built-in function performs better. MATLAB uses much more memory efficiently to trackback to decode output, so it performs better than the function made for this project.



[*code file: effect_of_noise.m*]

Effect of message length in Bit Error Rate performance:

To check that if there is any effect of the message length in the BER performance, BER vs. SNR curve is plotted with BPSK modulation for different message length. The following figure shows the BER performance for different message length. As the message length varies BER vs. SNR curve remains the same and thus it can be told that message length has no effect in BER performance.



[*code file: effect_of_noise.m*]