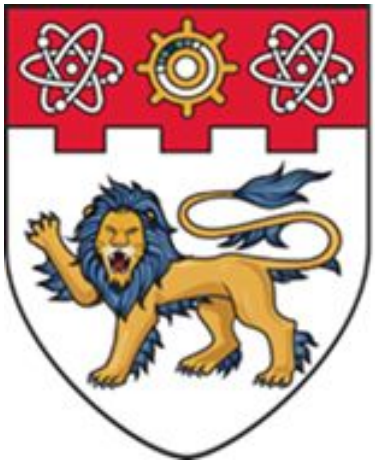

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CE 3006 Digital Communication
Course Project
Group 7

Group Members	Matriculation Number
SUN QINAN (Leader)	U1620772L
NG QI MING	U1621899B
MUHAMMAD AMIRUL AFIQ BIN JA'AFAR	U1620684A
GERALD TAN JING KAI	U1622611H
THA TOE OO @JODI	U1621448G

School of Computer Science and Engineering

Table Of Content

Introduction

Communication takes place in our daily lives in the form of signals. For example, when communication occurs in a group, information is transmitted in the form of an analog signal. However, analog signal is more prone to noise and distortion when travelling through a communication channel and in the signal processing operation. Hence, an analog signal is often converted to a digital signal in digital communication. In digital communication, the input signal (analog) is transmitted into binary signal. After encoding, the binary signal is modulated to reduce noise and interference. Furthermore, modulation converts the base-band signal to a modulated signal at a higher frequency that is suitable for transmission. When the signal reaches the receiver, the binary signal will be demodulated, de-encoded and convert back to analog signal at the receiver side.

Objective

The goal is to design a basic digital communication system using MATLAB. The implementation of this digital communication system will consist of three phases in the following order. Data generation, followed by modulation for communication, and lastly basic error control coding to improve the performance. The structure for modulation and demodulation of the communication signal is illustrated in the diagram below. All the observations seen in this report can be replicated following the instructions in the Appendix.

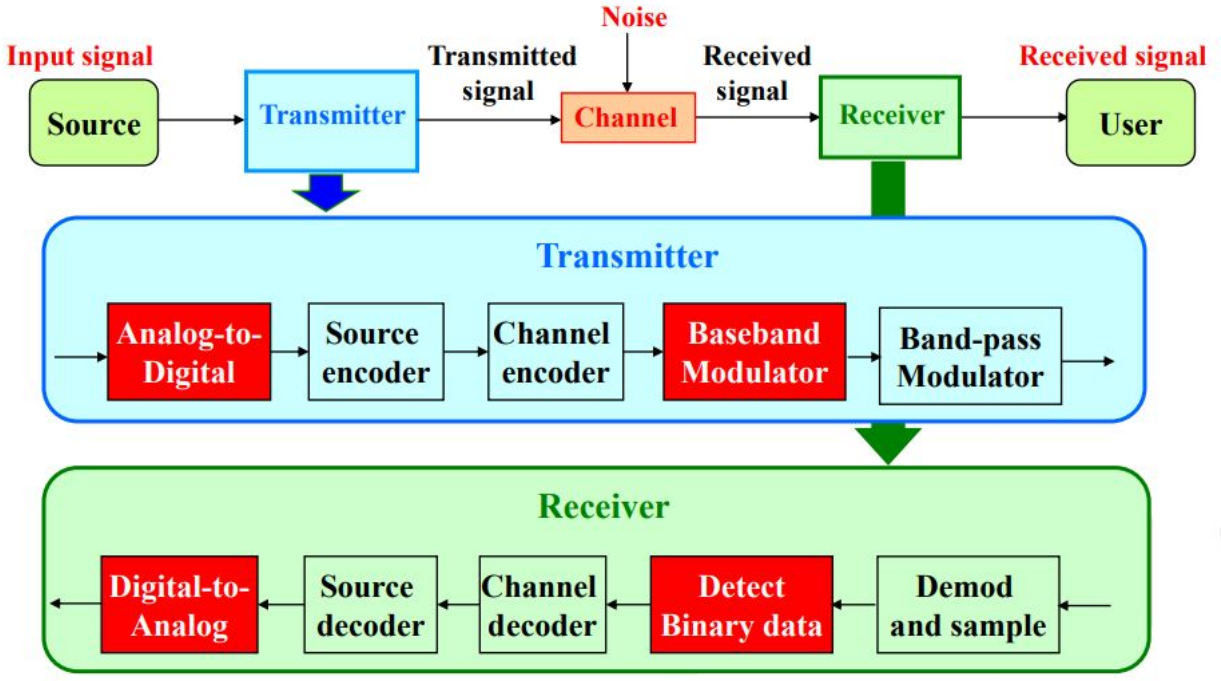

Fig 1

Phase 1: Data generation

In this section of the project, we were tasked to generate a binary data (0 or 1) of size 1024 bit. This was done using the following line of code:

```
NumOfBits = 1024;
input = randi([0, 1], [1, NumOfBits]);
```

The data will then be converted into ±1 where values 0 will be converted to -1 and 1 remains as +1 as shown here:

```
%Convert binary digits to +1 and -1
input(input==0) = [-1];
```

This converted data will be used for transmission through an Additive White Gaussian noise channel. We generate the noise required with zero mean and required Signal to Noise Ratio variance with the following code snippet:

```
b = 0; %mean
NoiseVariance = 1/ 10^(SNR/10);
NoiseSTD = sqrt(NoiseVariance); %Standard Deviation
noise = NoiseSTD.*randn(1,1024) + b;
```

Generated binary data is then transmitted through the Additive White Gaussian Channel and passed through a thresholding logic at the receiver.

```
%Add noise sample with input data
output = input + noise;
%Fix the threshold value as 0 (the transmitted data is +1 and -1, and
%0 is the mid value. if >=0, 1 else 0.
output(output>=0) = [1];
output(output<0) = [0];
```

We are tasked to investigate the effects of varying the Signal to Noise ratio values on the bit error rate performance. To investigate the effects of bit error rate performance, we compare the output values after thresholding with the input values that we transmitted. We can calculate the bit error rate with the following equation:

$$Bit\ Error\ Rate\ =\ \frac{number\ of\ errors\ during\ transmission}{total\ number\ of\ bits\ for\ transmission}$$

The following code snippet shows the way we attained the number of errors during transmission and the calculation of Bit Error Rate:

```
bitError = 0
for i=1 : length(output)
    if (tempInput(i) > threshold && output(i) == 0) || (tempInput(i)
<= threshold && output(i) == 1)
        bitError = bitError + 1;
    end
    bitErrorRate= bitError/NumOfBits;
end
```

In our experiment, we measure the bit error rate performance against signal to noise ratio in intervals of 5 from 0 to 50.

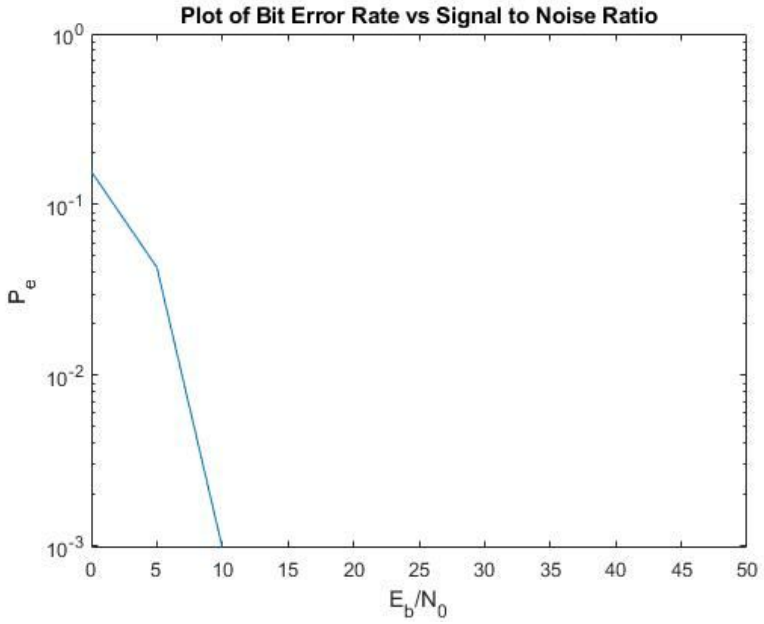

Fig 2.

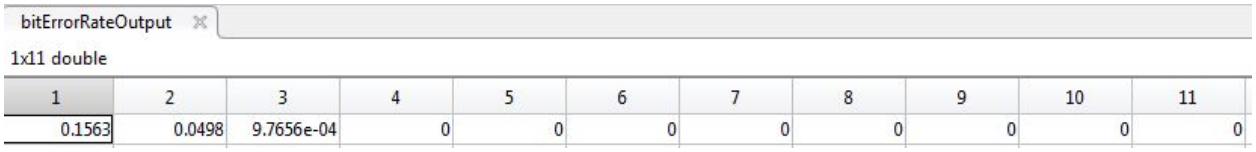

Fig 3.

The plot above shows the trend of bit error rate versus signal to noise ratio that we attained. From the above plot, we notice the bit error performance increased exponentially as the signal to noise ratio value increases. As shown above, 'bitErrorRateOutput' decreases to zero after a signal to noise ratio of 10.

Phase 2: Modulation for communication

Modulation is used to shift the signal is shifted to a higher frequency before transmission. This is usually done to make the transmitted signal more robust against noise and interference so that the errors produced by the effects of noise and interference will be reduced.

In this project, we are tasked with using two modulation methods: On-Off Keying(OOK) and Binary Phase Shift Keying(BPSK). In OOK, the data signals will be considered 1 or 0 format while in BPSK, it has to be +1 or -1 format.

In this project, the carrier frequency is 10 KHz, sampling frequency is 16 times the carrier frequency and baseband data rate is 1kbps. Thus, the sampling frequency will be 160 times the baseband data rate. Thus the 1s and 0s in the original generated baseband data needs to be extended by 160 so that it will correctly reflects the values at the sampling intervals. This was done using the following code:

```
extension_vector = ones(1, Fs/dataRate);
sampled_input = kron(input, extension_vector);
```

The amount of time that will be needed to transmit the whole data can be found by the number of bits for transmission divided by data rate. In this case, the total amount of time that will be needed to transmit the data is 1024/1000 which would be 1.024 seconds. The sampling intervals can be obtained by the inverse of the carrier frequency. However, the sampling intervals will start from 1 / (2 * sampling frequency) = 3.125000000000000e-06 . This is to ensure that every interval between 0 and 1.024 seconds is represented properly with the total number of intervals being 163840 which is obtained by extending the bits in the generated baseband data by 160 times.

```
time = 1/(2 * Fs): 1/Fs: numOfBits/dataRate;
```

In order to carry out the modulation with the OOK modulation method, the extended generated baseband data can be multiplied elementwise with the carrier signal (cos(2 * pi * f * t)). This was done using the following code snippet:

```
carrier = cos(2 * pi * Fc * time);
sampled_ook = sampled_input .* carrier;
```

In order to carry out the modulation with the BPSK modulation method, the extended generated baseband data first needs to be converted such that 1 will remain 1 while 0 will become -1. The converted signal can then be multiplied element-wise with the carrier signal (cos(2 * pi * f * t)). This was done using the following code snippet:

```
sampled_bpsk = (2 * sampled_input - 1) .* carrier;
```

The modulated signal will then be added with additive white noise and sent to the receiver which is shown in the following code snippet:

```
noise_variance = 1 / 10^(SNR/10);
noise_std = sqrt(noise_variance);
noise = noise_std .* randn(1, 1024 * Fs/dataRate);
received_signal = sampled_bpsk + noise;
```

The received signal will then be demodulated by performing an element-wise multiplication with twice the carrier signal(i.e. 2 * (cos(2 * pi * f * t))) as shown in the following code snippet:

```
demodulated_signal = received_signal .* (2 * carrier);
```

The resulting signal will then be passed through a low pass filter as shown in the following code snippet:

```
filtered_signal = filtfilt(b, a, demodulated_signal);
```

Afterwards, the decoding of the received signals can be performed to convert the signal back to 1024 bits and the midpoint of the bits will be used for decision logic. For signal that have undergone OOK modulation, the following logic will be used. If the resulting signal from the low pass filter is above 0.5, the decoded signal will be 1. Else the decoded signal will be 0. This can be seen from the following code snippet:

```
for i = 1:1:1024
             interested_signal = filtered_signal(1 /2 * Fs/dataRate +
(i - 1) * Fs/dataRate);
            if interested_signal > 0.5
                decoded_signal(i) = 1;
            else
                decoded_signal(i) = 0;
            end
end
```

For signal that have undergone BPSK modulation, the following logic will be used. If the resulting signal from the low pass filter is above 1, the decoded signal will be 1. Else the decoded signal will be 0. This can be seen from the following code snippet:

```
for i = 1:1:1024
     interested_signal = filtered_signal(1 /2 * Fs/dataRate + (i -
1) * Fs/dataRate);
            if interested_signal > 0
                decoded_signal(i) = 1;
            else
                decoded_signal(i) = 0;
            end
```

```
end
```

The decoded signal will be checked with the original signal to find the bit error rate. This is done in the following code snippet:

```
bitErrorRate = calculate_error_rate(decoded_signal, input);
function bitErrorRate = calculate_error_rate(input, tempInput)
    %Generate noise having normal distribution with zero mean
    error = 0;
    numOfBits= 1024;
    for i = 1:1:numOfBits
        if input(i) ~= tempInput(i)
            error = error + 1;
        end
    end
    bitErrorRate = error/numOfBits;
end
```

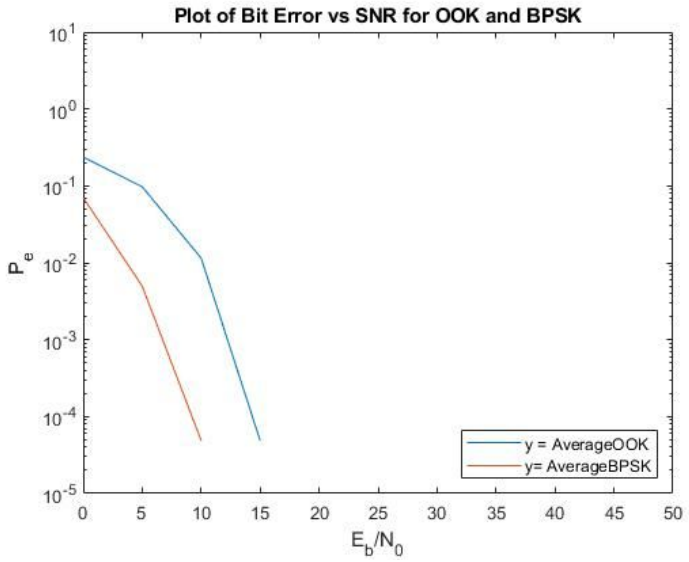

Fig 4.

The following plot shows the performance of On-Off keying (OOK) vs Binary Phase Shift Keying (BPSK). From the above plot we can observe that BPSK modulation method has a better performance as compared to OOK.

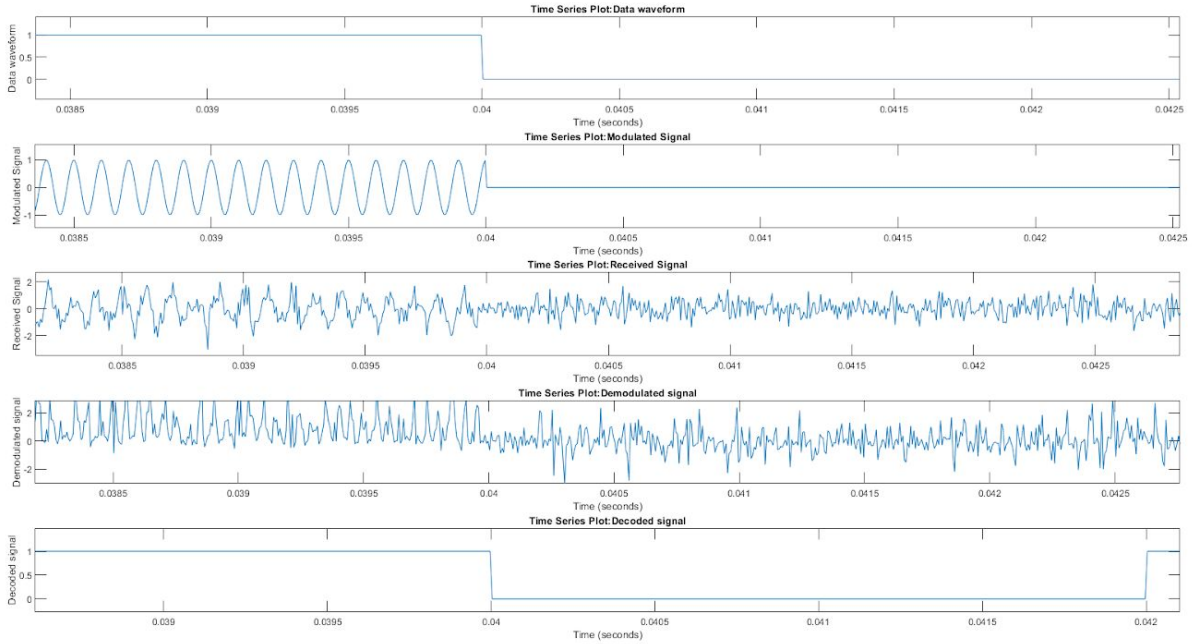

Fig 5.

From the above sample plot of waveforms, we can observe the effects of On-Off keying. When the data waveform transitions from 1 to 0, we can observe in the modulated signal that there are no waveforms transmitted. Waveforms are generated at a SNR value of 5db. This is an expected observation On-Off keying modulation.

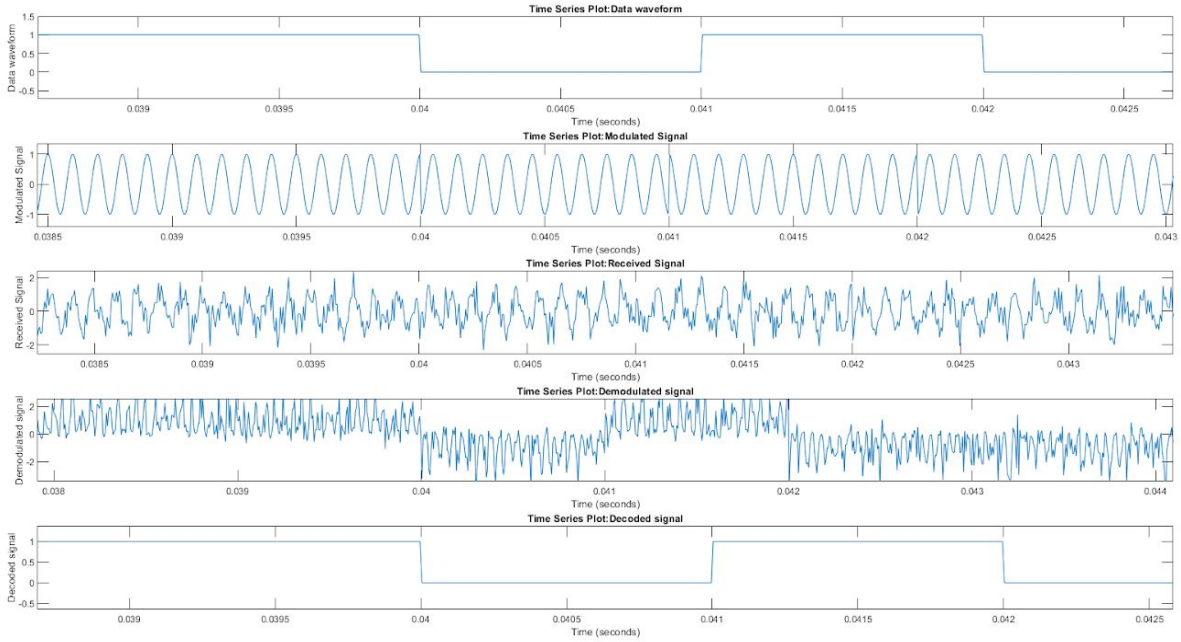

Fig 6.

From the above sample plot of waveforms, we can observe the effects of BPSK. When the data waveform transitions from 1 to 0, we can observe in the modulated signal that there is a 180

degree phase shift or an inversion of the modulated signal. Waveforms are generated at a SNR value of 5db. This is an expected observation of BPSK modulation.

Phase 3: Basic error control coding to improve the performance

In this section, we are tasked to explore the various error control coding methods on the performance of the developed communication system. Our team has decided to use Hamming Code (7,4) , Linear Coding and Cyclic coding. The implementation for encoding methods are as follows:

Hamming Code

```
%encode
sampled_input = encode(input, 7, 4, 'hamming/fmt');
%decode
decoded_signal = decode(decoded_signal,7,4,'hamming/fmt');
```

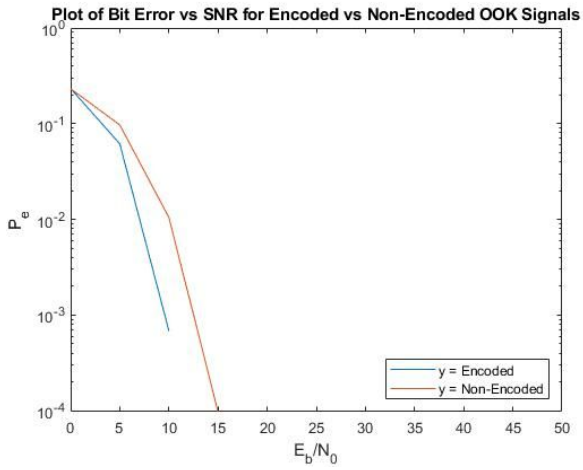

Fig 7.

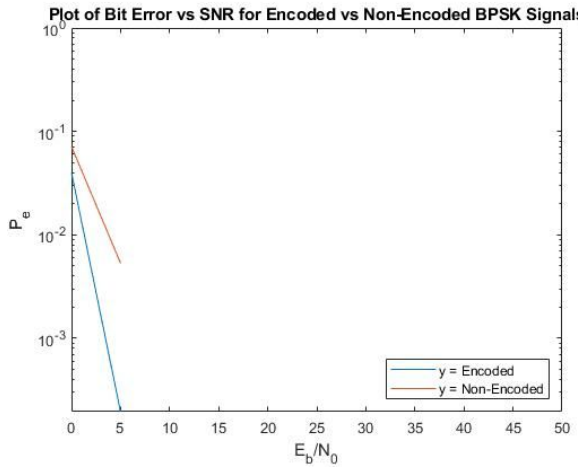

Fig 8.

As seen from Figure 7 and 8, we have attained expected improvements in the bit error rate performance across all Signal to Noise ratio values when the signal is encoded. This is observed for both the OOK modulated signal as well as the BPSK encoded signal.

Linear Coding

```
%encode
sampled_input = encode(input, 7, 4, 'linear/fmt',genmat);
%decode
decoded_signal = decode(decoded_signal,7,4,'linear/fmt',genmat);
```

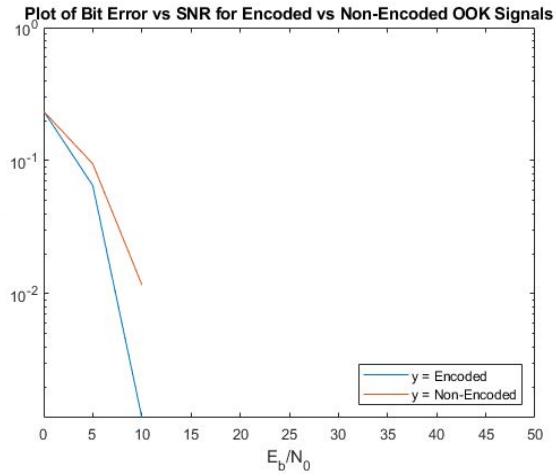

Fig 9.

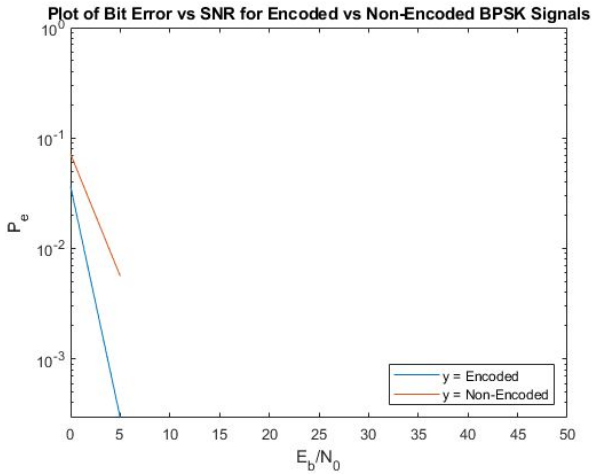

Fig 10.

As seen from Figure 9 and 10, we have attained expected improvements in the bit error rate performance across all Signal to Noise ratio values when the signal is encoded. This is observed for both the OOK modulated signal as well as the BPSK encoded signal.

Cyclic Coding

```
%encode
sampled_input = encode(input, 7, 4, 'cyclic/fmt',gpol);
%decode
decoded_signal = decode(decoded_signal,7,4,'cyclic/fmt',gpol,trt);
```

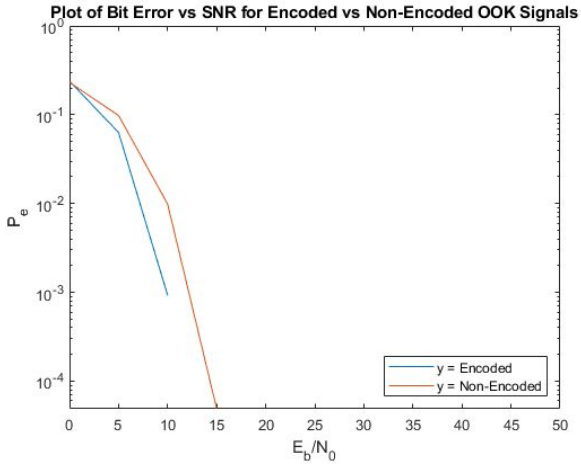

Fig 11.

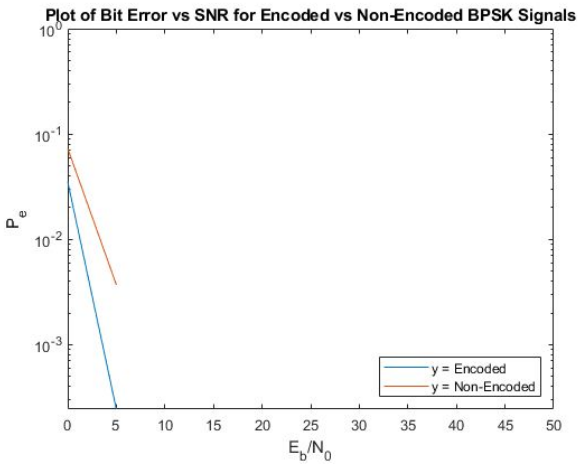

Fig 12.

As seen from Figure 11 and 12, we have attained expected improvements in the bit error rate performance across all Signal to Noise ratio values when the signal is encoded. This is observed for both the OOK modulated signal as well as the BPSK encoded signal.

We will now be comparing the effects of various error control methods on its bit error performance. We will only be observing the above specified effects on an OOK modulated signal.

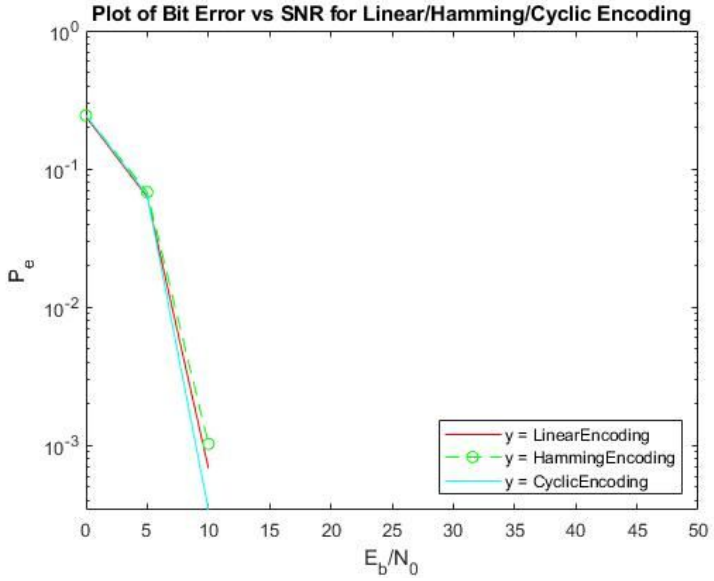

Fig 13.

Based on Figure 9, we can observe that each error control coding methods have a minimal advantage over the other in terms of bit error performance over various Signal to Noise Ratio.

Conclusion

In conclusion, we generated digital data and modulated the data for transmission across an additive white gaussian channel. We have gathered that the addition of noise from the channel has caused changes to the received signal in the receiver. However, by adjusting the Signal to Noise ratio, we can expect a higher bit error rate performance in the received signal.

In Phase 2, we explored various modulation methods, namely, On-Off keying (OOK) and Binary Phase Shift Keying (BPSK) and the properties of the generated waveform. We have learned that at the same Signal to Noise ratio values, BPSK has a better bit error rate performance as compared to OOK.

In the final phase, we explored simple error coding methods such as hamming code to improve the bit error rate performance. We went on further to explore the performance difference between various error coding methods and noticed little to no advantage on performance improvement.

Appendix

The following section describes the required steps needed to replicate the plots that were achieved in each section. All the required files will be found enclosed in the zip file submitted in NTULearn. As the data generation and noise generation is random, the plots may not appear to be the exact same but will follow a similar trend.

Phase 1

To attain Figure 2, run the “LabProject3006.m” file.
Figure 3 can be attained by clicking on the “bitErrorRateOutput” variable in the workspace.

Phase 2

To attain Figure 4, run the “LabProject3006Part2.m” file.
To attain Figure 5, run the “LabProject3006Part2ExtraPlots.m” file.
To attain Figure 6, run the “LabProject3006Part2ExtraPlotsBPSK.m” file.

Phase 3

To attain Figure 7, run the “HammingEncodedOOKvsNonEncodedOOK.m” file.
To attain Figure 8, run the “HammingEncodedBPSKvsNonEncodedBPSK.m” file.
To attain Figure 9, run the “LinearEncodedOOKvsNonEncodedOOK.m” file.
To attain Figure 10, run the “LinearEncodedOOKvsNonEncodedBPSK.m” file.
To attain Figure 11, run the “CyclicEncodedOOKvsNonEncodedOOK.m” file.
To attain Figure 12, run the “CyclicEncodedOOKvsNonEncodedBPSK.m” file.
To attain Figure 13, run the “OOKLinearvsHammingvsCyclic.m” file.