NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CE3006 Digital Communication

PROJECT REPORT

Thomas Stephen Felix	U1722308D
Pereddy Vijai Krishna Reddy	U1722307E
Palaniappan Sneha	U1722528E
Gustav Villwock	N1902671B
Jakob Aronsson Kallgren	N1903118J

Introduction

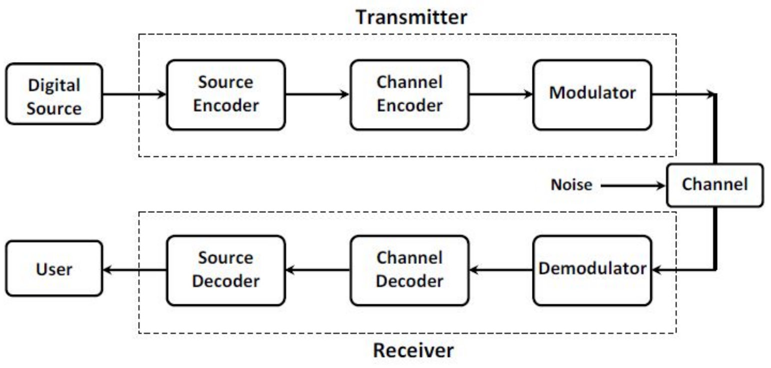
What is digital communication?

The transmission of information from the sender to the recipient through some medium is generally understood as communication. The lecture notes have defined it as the transmission of information from one point to another through a succession of processes. These processes include the following :

1. Generation of a message signal (ex.voice),
2. Description of the message with certain measures of precision,
3. Encoding of symbols, suitable for transmission over a physical medium of interest
4. Transmission of encoded symbol to the desired destination
5. Decoding and reproduction of original symbols
6. Recreation of the original message signal with a definable degradation in quality.

How are signals transmitted in a digital communication system?

Due to the modern-day digital age, digital signals are greatly preferred over analog signals. In order to transmit a signal, we have to perform the following steps:



Phase 1 - Signal Generation & Decoding

This section is related to baseband modulation and demodulation. Here the binary data will be generated and will be transmitted through an additive white Gaussian noise channel with different SNR values. The bit error rate performance will be analyzed and plotted against various SNR values.

BACKGROUND

Let's try to understand a few key terms stated in the Phase 1 requirements mentioned above. **Additive White Gaussian Noise** is a theoretical representation of distortion added to a signal consisting of a range of frequencies. It is *additive* because the received signal is an addition of transmitted signal and noise. It is *white* as it is a culmination of a wide range of frequencies (eg. the overlapping of electromagnetic frequencies in the visible range leads to the formation of white light). It is *gaussian* because no matter how closely the values are placed together they are all independent of each other and are most commonly found in nature. It is an example of a continuous probability distribution.

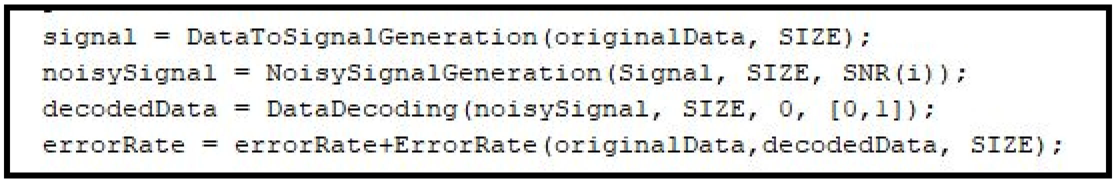
Signal to Noise Ratio (SNR) is the ratio of average signal power to average noise power. Generally, a larger value of SNR is preferred as it indicates better performance. We can increase SNR value by increasing the power but power is limited which means Signal is limited. Hence, the signal power is always restricted and the only way is to reduce the Noise value.

Bit Error Rate is understood as the number of bit errors that occurs in a given number of bit transmissions.

IMPLEMENTATION

1.1 Main Code

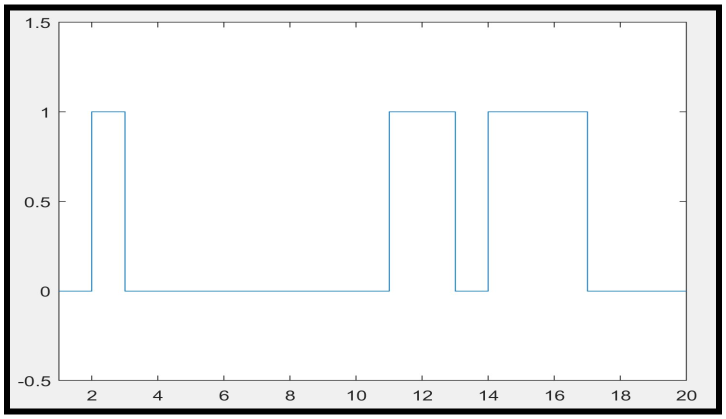
The code for Phase 1 is dissolved into functions for easier understanding, execution & reusability.



```
signal = DataToSignalGeneration(originalData, SIZE);
noisySignal = NoisySignalGeneration(Signal, SIZE, SNR(i));
decodedData = DataDecoding(noisySignal, SIZE, 0, [0,1]);
errorRate = errorRate+ErrorRate(originalData,decodedData, SIZE);
```

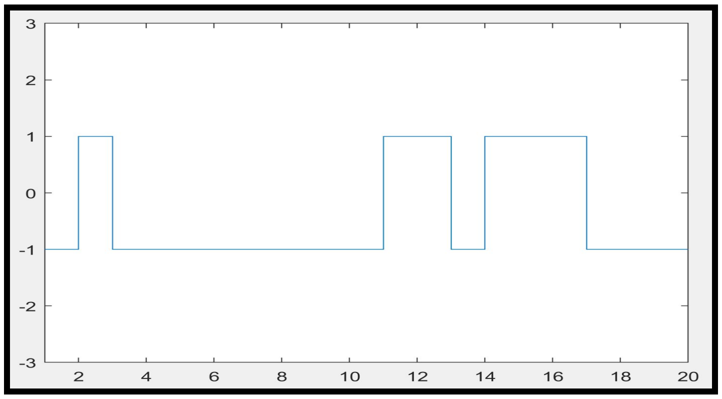
Binary Data Generation

1. Binary Data Generation is being done in the Main Code. As the assignment required the use of 1024 bits for the transmission, we are setting a variable in the name of 'SIZE' to 1024.
2. We then proceeded on to use the Matlab function "randi([0 1],1,SIZE)" to create an array of random binary numbers with a sample interval of [0 1] and matrix size of (1x1024), assigned to the variable '**originalData**'.
3. Data: [0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0]



Data to Signal Generation (DataToSignalGeneration.m)

1. The generated '**originalData**' is passed into this function to generate the '**signal**' which can be done by converting the '0' digits to -1 and '1' digits to 1. This process is also known as Bipolar Encoding.
2. This is implemented through a simple 'for' loop to traverse through each and single value of original data and converts it to its respective value through an 'if-else' statement.
3. Signal: [-1, 1, -1, -1, -1, -1, -1, -1, -1, 1, 1, -1, 1, 1, 1, -1, -1, -1]



Noisy Signal Generation (NoisySignalGeneration.m)

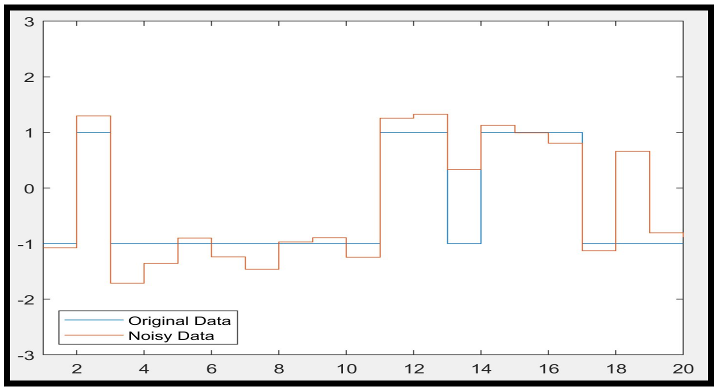
1. The '**signal**' generated from the previous function is passed in this function to add the transmission noise to the signal making it the '**noisySignal**'.
2. To generate noise, we used the formula

trans_noise = sqrt(N) * *noise_gen*

N = S /(10^ (SNR /10))

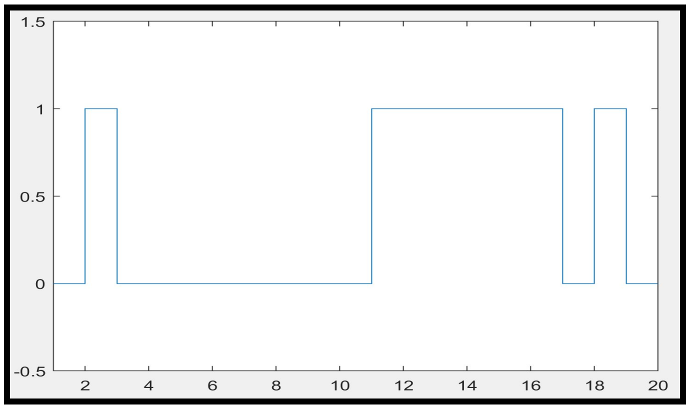
noise_gen = randn(1,SIZE)

3. 'randn(1,SIZE)' is a Matlab function that helps to return an array of Noise that is normally distributed.



Data Decoding (DataDecoding.m)

1. The '**noisySignal**' generated previously is passed into this function and the '**decodedData**' is generated.
2. Here, we set the threshold value/boundary as 0 and convert digits less than or equal to the boundary value to 0 and the rest to 1.
3. Decoded Data: [0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0]

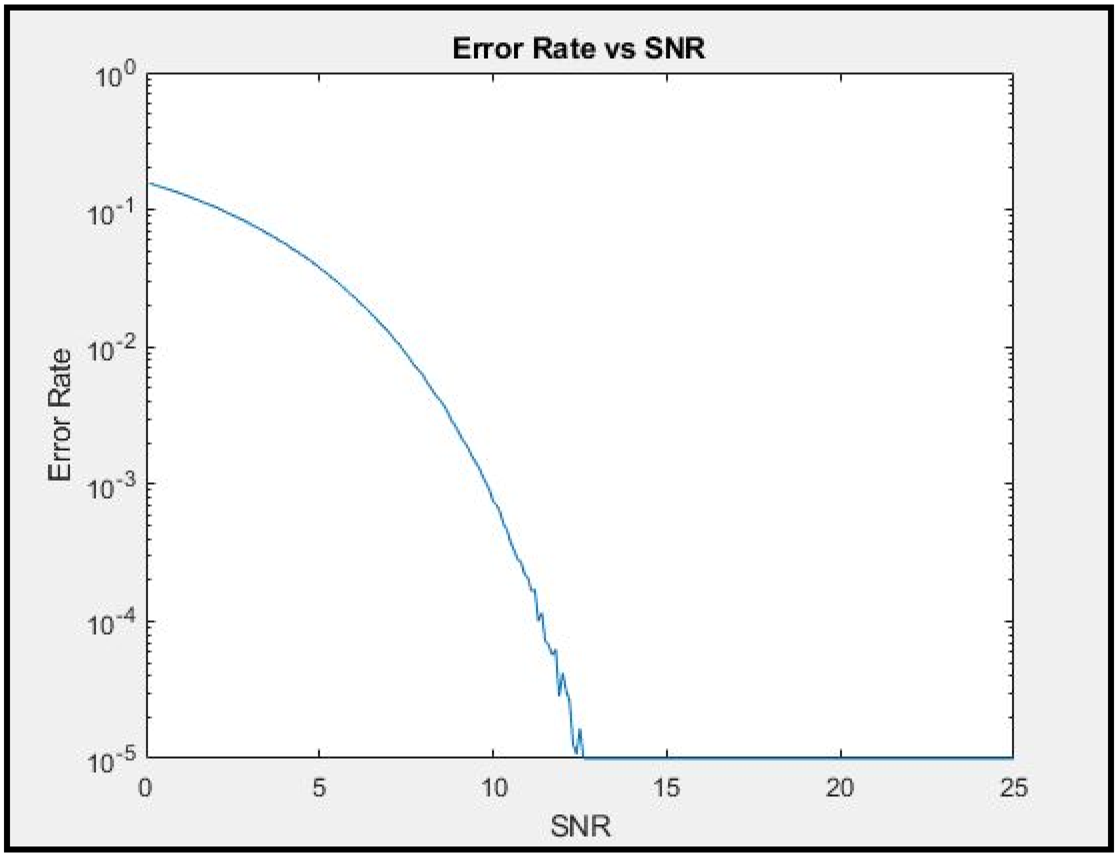


<u>Error Rate</u> (ErrorRate.m)

1. The '**decodedData**' is used in this function to calculate the '**ErrorRate**'.
2. The Error rate is calculated by comparing the '**originalData**' with '**decodedData**' in every iteration. Every time they are not equal we increase the error count and finally we divide the total number of errors with the size of the original data to compute bit error rate.

<u>ANALYSIS</u>

We can understand phase 1 by plotting the error rate to different SNR values. As seen in the graph below, the error rate is inversely proportional to SNR values. Therefore higher SNR value of the system is preferred.



The graph can be generated by running part1.m code

Phase 2 - Signal Modulation & Demodulation

This section is related to band-pass modulation and demodulation. We were asked to implement different band-pass modulation schemes and record relevant observations. We implemented binary modulation methods such as On-Off Keying (OOK), Binary Frequency Shift Keying (BFSK) and Binary Phase Shift Keying (BPSK). Additionally, M-ary modulation schemes such as M-ary ASK and M-ary PSK has also been implemented.

BACKGROUND

In a *Baseband transmission*, information is sent without frequency shifting while in *Bandpass transmission*, the signal is shifted to a higher frequency before transmission. We do this in order to reduce the effect of noise and interference.

Modulation is defined as “It is a process of encoding information from a message source in a manner suitable for transmission.” This basically means the conversion of the signal from a lower frequency to a higher frequency signal. There are 3 main modulation schemes, ‘Amplitude Modulation (AM),’ ‘Frequency Modulation (FM),’ and ‘Phase Modulation (PM).’ All these are known as analog modulation. Whereas for digital modulation the name changes to the Analog Shift Key (ASK), Frequency Shift Key (FSK) and Phase Shift Key (PSK)’

Demodulation is the process in which the received signal is shifted back to original frequency. There are 2 types of category for demodulation methods. One is known as *Coherent detection* and the other is known as *Non-coherent detection*.

- *Coherent detection* : When the receiver make use of carrier signal information in order to detect the incoming signal with good accuracy.
- *Non-coherent detection* : When phase information or carrier signal information is not utilised. A trade off between complexity and performance

There are 3 main Band-pass modulation methods, Amplitude Shift Keying (ASK), Frequency Shift Keying (FSK) and Phase Shift Keying (PSK). For each of these methods, only one of the variables can be changed at a time. For example, for ASK, only the amplitude will be changed and everything else remains the same.

In **Amplitude Shift Keying**, we have two types of modulation. One is known as On-Off Keying (OOK) and another is known as M-ary ASK. M-ary ASK is used in cases in which 2 or more symbol states are used which means instead of 1-bit, 2 or more bits are transmitted a time. Taking reference to this formula: $m=\log_2 M$ where m equal number of bits, in M-ary ASK, the amplitude of the carrier signal takes on M different levels.

In **Frequency Shift Keying**, digital information is transmitted through discrete frequency changes of a carrier wave. For example, the output of an FSK modulated will be higher in frequency for binary high inputs and lower in frequency when it wants to send a binary low input. The most common type of FSK is known as Binary FSK which simply means the input signal can only have 2 types of values.

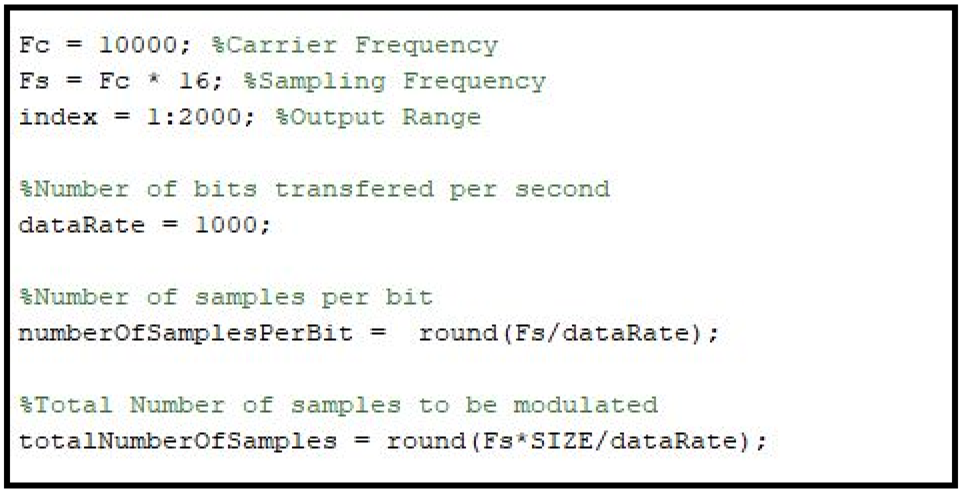
In **Phase Shift Keying**, digital information is transmitted by changing the phase of the carrier signal which can be done by changing/switching up the sine and cosine inputs at a particular instant. The simplest form of PSK is known as BPSK, in BPSK as the name suggests, it only uses two phases which are separated by 180 degrees (0 & 180). Under PSK we also have M-ary PSK for which the phase of the carrier signal takes on M different phases. The phase difference with the carrier signal is used to identify the data.

IMPLEMENTATION

Let's understand the above codes by accessing them based on their modulation techniques.
There are certain details that have been set and followed through the rest of the modulation techniques. They are:

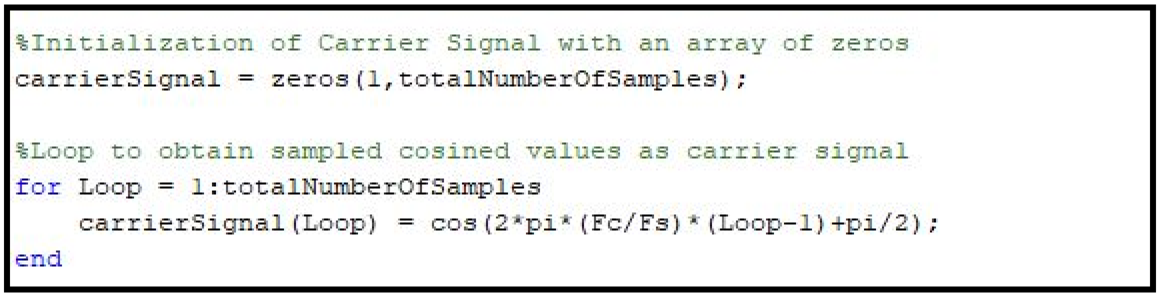
- Carrier Frequency, Fc = 10KHz.
- Sampling Frequency, Fs = Fc * 16 % Since the carrier signal is oversampled 16 times,
- Baseband Data Rate = 1 kbps (1000 bits per second)

From the above values, '**totalNumberOfSamples**' can be computed as shown below:



```
Fc = 10000; %Carrier Frequency
Fs = Fc * 16; %Sampling Frequency
index = 1:2000; %Output Range

%Number of bits transfered per second
dataRate = 1000;

%Number of samples per bit
numberOfSamplesPerBit =  round(Fs/dataRate);

%Total Number of samples to be modulated
totalNumberOfSamples = round(Fs*SIZE/dataRate);
```

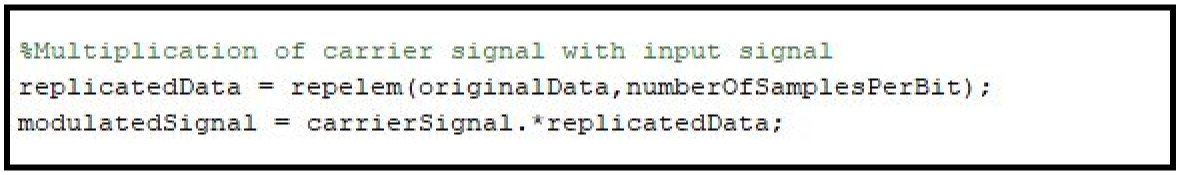
<u>Carrier Signal</u>
A carrier signal can be generated using the cos function as shown below. The carrier signal is used to encode the low frequency binary data.



```
%Initialization of Carrier Signal with an array of zeros
carrierSignal = zeros(1,totalNumberOfSamples);

%Loop to obtain sampled cosined values as carrier signal
for Loop = 1:totalNumberOfSamples
    carrierSignal(Loop) = cos(2*pi*(Fc/Fs)*(Loop-1)+pi/2);
end
```

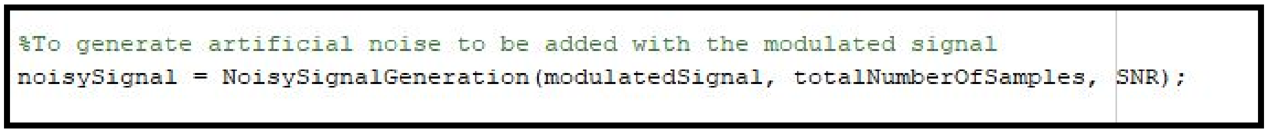
On-Off Keying (OOK)

<u>Modulated Signal</u>
A modulated signal is generated by multiplying the original data with the carrier wave as shown below. Modulation is generally done to avoid/reduce the loss of data.



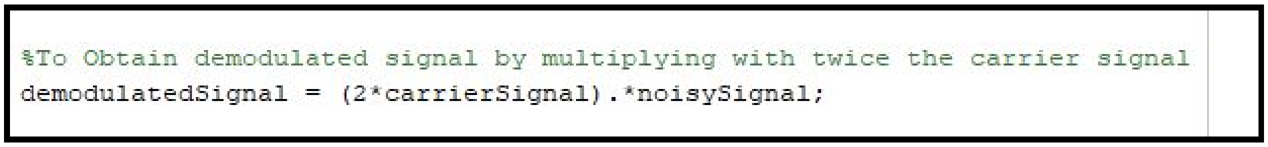
```
%Multiplication of carrier signal with input signal
replicatedData = repelem(originalData,numberOfSamplesPerBit);
modulatedSignal = carrierSignal.*replicatedData;
```

<u>Noisy Signal</u>
The modulated signal is passed to the '**NoisySignalGeneration**' function to add transmission noise to the modulated signal as shown below.



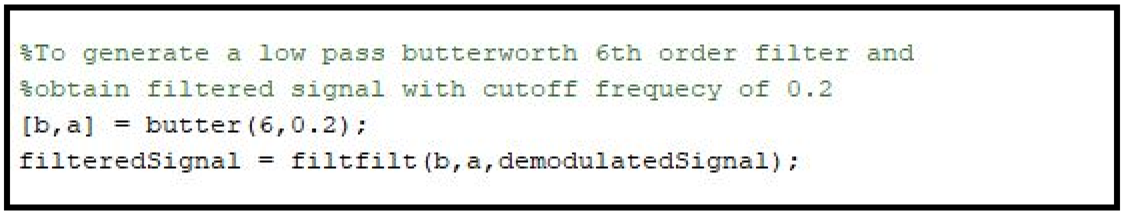
```
%To generate artificial noise to be added with the modulated signal
noisySignal = NoisySignalGeneration(modulatedSignal, totalNumberOfSamples, SNR);
```

<u>Demodulated Signal</u>
The demodulated signal is generated by multiplying twice the '**carrierSignal**' with the '**noisySignal**' generated from the previous function.



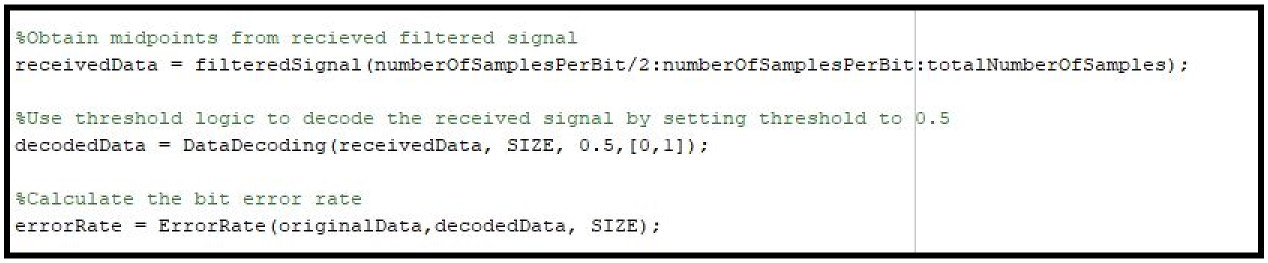
```
%To Obtain demodulated signal by multiplying with twice the carrier signal
demodulatedSignal = (2*carrierSignal).*noisySignal;
```

<u>Filtered Signal</u>
Here, we use Matlab's pre-defined functions *butter() and fltfilt()* to do a low pass Butterworth filter and to generate the filtered signal by passing in the '**demodulatedSignal**'. It can be done as shown below.



```
%To generate a low pass butterworth 6th order filter and
%obtain filtered signal with cutoff frequecy of 0.2
[b,a] = butter(6,0.2);
filteredSignal = filtfilt(b,a,demodulatedSignal);
```

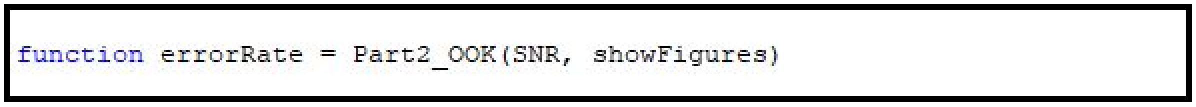
<u>Error Rate</u>
First, we obtain the midpoints from the received filtered signal and assign it to '**receivedData**' and we pass this received data to the '**DataDecoding**' function with a threshold of 0.5 (to achieve minimum error) to generate the '**decodedData**'. Then we pass the decoded data to the '**ErrorRate**' function to calculate the bit error rate.



```
%Obtain midpoints from recieved filtered signal
receivedData = filteredSignal(numberOfSamplesPerBit/2:numberOfSamplesPerBit:totalNumberOfSamples);

%Use threshold logic to decode the received signal by setting threshold to 0.5
decodedData = DataDecoding(receivedData, SIZE, 0.5,[0,1]);

%Calculate the bit error rate
errorRate = ErrorRate(originalData,decodedData, SIZE);
```

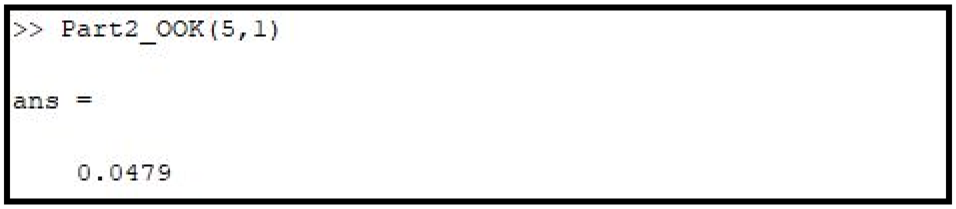
<u>GRAPHS</u>

Graphs of OOK can be plotted by entering the command ***Part2_OOK(desired_SNR, true)*** on the command window.

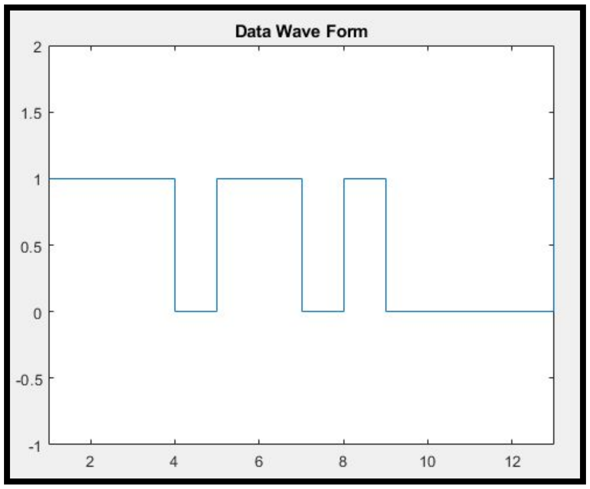


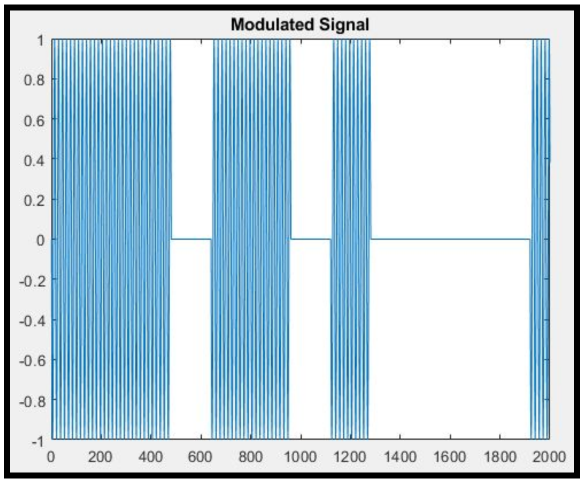
```
function errorRate = Part2_OOK(SNR, showFigures)
```

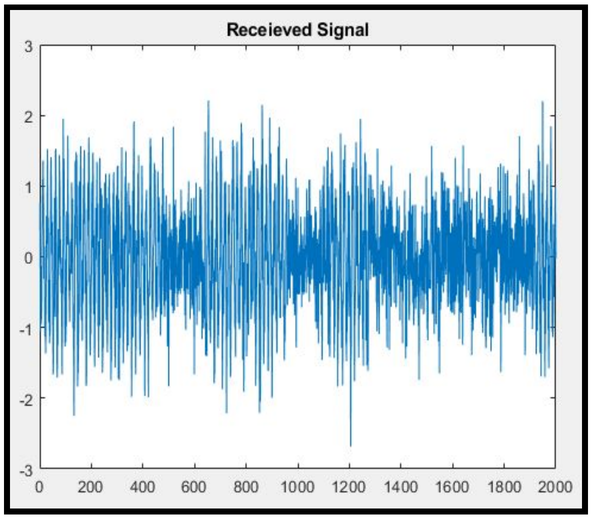
For example: If we enter Part2_OOK(5, 1), the following graphs are plotted

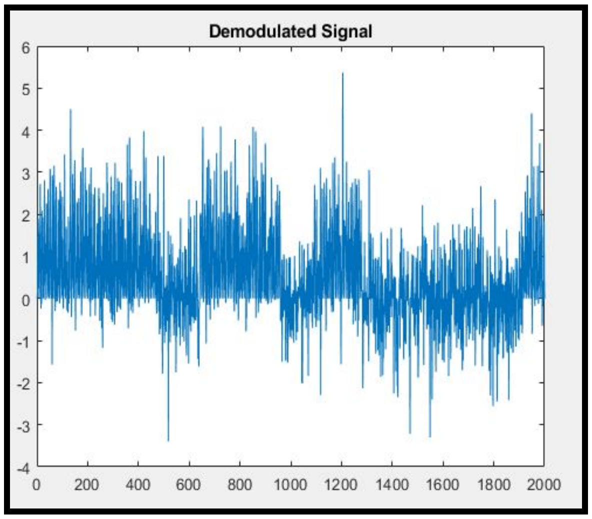


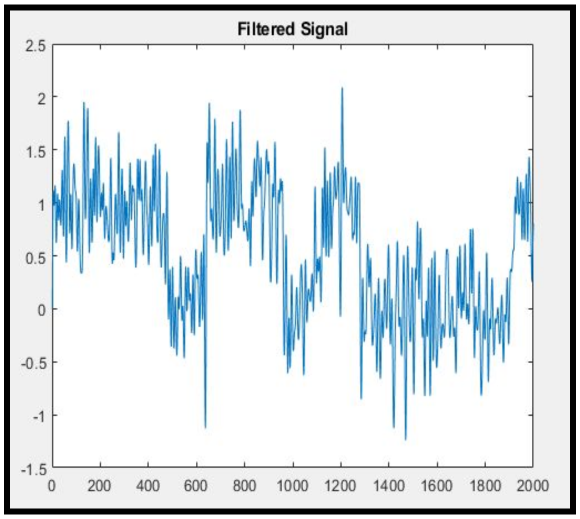
```
>> Part2_OOK(5,1)

ans =

    0.0479
```

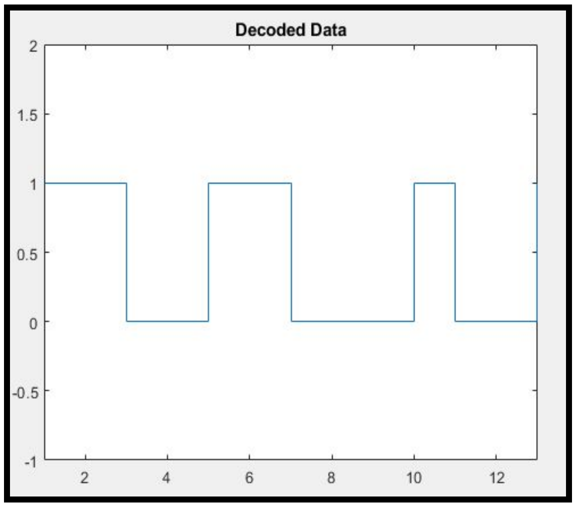

Data Wave Form
2
1.5
1
0.5
0
-0.5
-1
2
4
6
8
10
12


Modulated Signal
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
200
400
600
800
1000
1200
1400
1600
1800
2000


Receieved Signal
3
2
1
0
-1
-2
-3
0
200
400
600
800
1000
1200
1400
1600
1800
2000

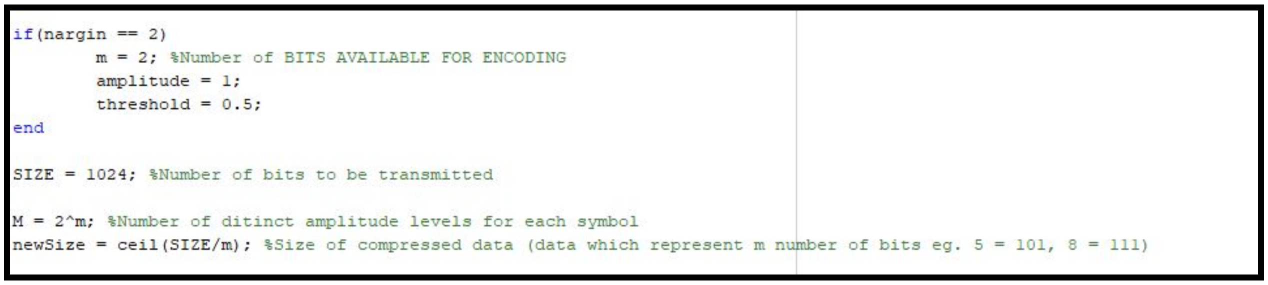

Demodulated Signal
6
5
4
3
2
1
0
-1
-2
-3
-4
0
200
400
600
800
1000
1200
1400
1600
1800
2000


Filtered Signal
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
0
200
400
600
800
1000
1200
1400
1600
1800
2000


Decoded Data
2
1.5
1
0.5
0
-0.5
-1
2
4
6
8
10
12

M-ary Amplitude Shift Keying

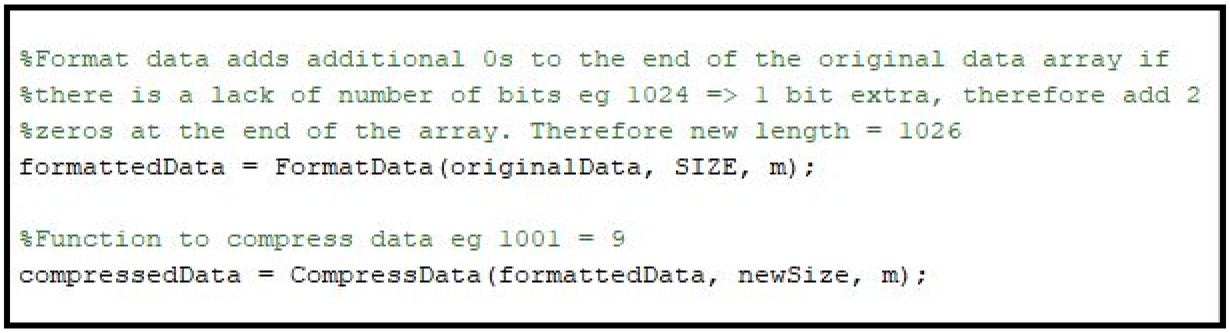
We have assigned m to 2 bits, amplitude to 1 and threshold to 0.5. Since m is equal to 2, the number of amplitude levels for this case will be 4 (M). In this case, this is known as a 4-ASK transmission.



```
if(nargin == 2)
        m = 2; %Number of BITS AVAILABLE FOR ENCODING
        amplitude = 1;
        threshold = 0.5;
end

SIZE = 1024; %Number of bits to be transmitted

M = 2^m; %Number of ditinct amplitude levels for each symbol
newSize = ceil(SIZE/m); %Size of compressed data (data which represent m number of bits eg. 5 = 101, 8 = 111)
```

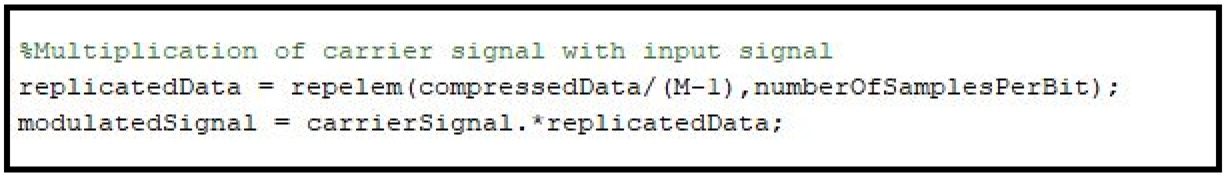
<u>Formatting the Data</u> (FormatData.m)
Format data pads additionals ‘0s’’to the end of the original data array. For example, for 3 bit encoding and data size = 1024, the data array has 1 bit extra therefore we will have to pad 2 zeros to the end of the array. Therefore the new length will be 1026 bits.

<u>Compressing the Data</u> (CompressData.m)
Compress Data is performed to convert a chunk of bits to symbols, in this case a decimal symbol. Eg. 1011 => 11



```
%Format data adds additional 0s to the end of the original data array if
%there is a lack of number of bits eg 1024 => 1 bit extra, therefore add 2
%zeros at the end of the array. Therefore new length = 1026
formattedData = FormatData(originalData, SIZE, m);

%Function to compress data eg 1001 = 9
compressedData = CompressData(formattedData, newSize, m);
```

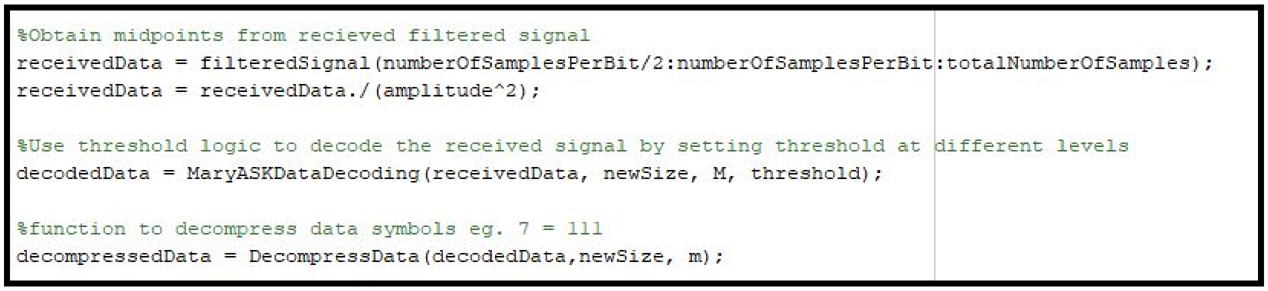
<u>Modulated Signal</u>
A modulated signal is generated by multiplying the scaled form of the compressed data with the carrier wave as shown below.



```
%Multiplication of carrier signal with input signal
replicatedData = repelem(compressedData/(M-1),numberOfSamplesPerBit);
modulatedSignal = carrierSignal.*replicatedData;
```

<u>MARY Data Decoding</u> (MaryASKDataDecoding.m)
We set signal amplitude by multiplying scaled signal data. For example for 3 bit encoding. To send value 2, we multiple the carrier signal with an amplitude ⅔ (M-1 = 3) and transmit it. In order to detect the signal, using the same logic we use the point between values to obtain the signal. Therefore to detect value 2, we set our threshold at 1.5/3 = 0.5. Therefore amplitude of received signal is above 0.5 and below 1.5 (we first check for value 3), it will be perceived as value 2.

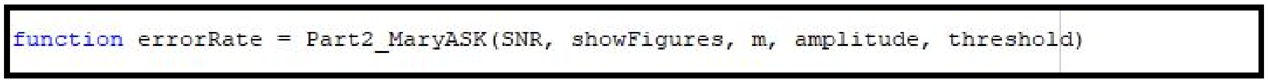
<u>Error Rate</u>
First, we obtain the midpoints from the received filtered signal and assign it to '**receivedData**' and we pass this received data to the '**DataDecoding**' function with a threshold of 0.5 (to achieve minimum error) to generate the '**decodedData**'. Then we pass the decoded data to the '**decompressedData**' function to decompress the symbols back to binary data.



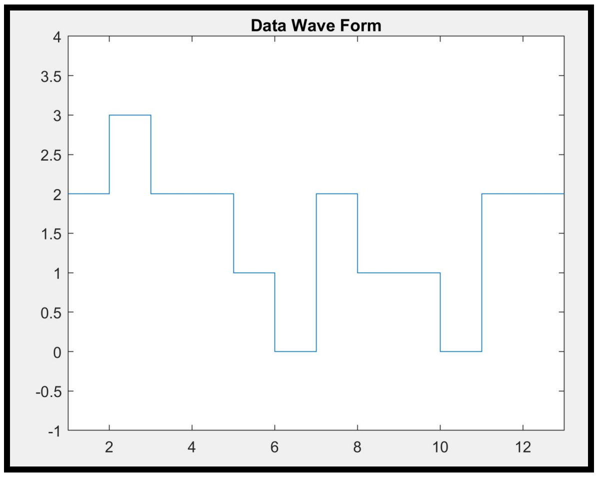
```
%Obtain midpoints from recieved filtered signal
receivedData = filteredSignal(numberOfSamplesPerBit/2:numberOfSamplesPerBit:totalNumberOfSamples);
receivedData = receivedData./(amplitude^2);

%Use threshold logic to decode the received signal by setting threshold at different levels
decodedData = MaryASKDataDecoding(receivedData, newSize, M, threshold);

%function to decompress data symbols eg. 7 = 111
decompressedData = DecompressData(decodedData,newSize, m);
```

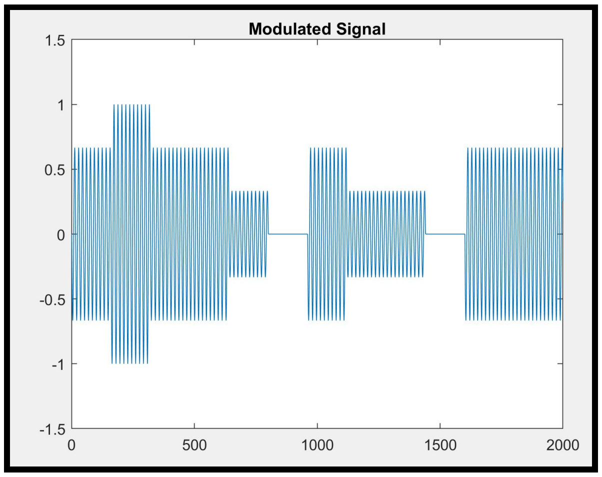
<u>GRAPHS</u>

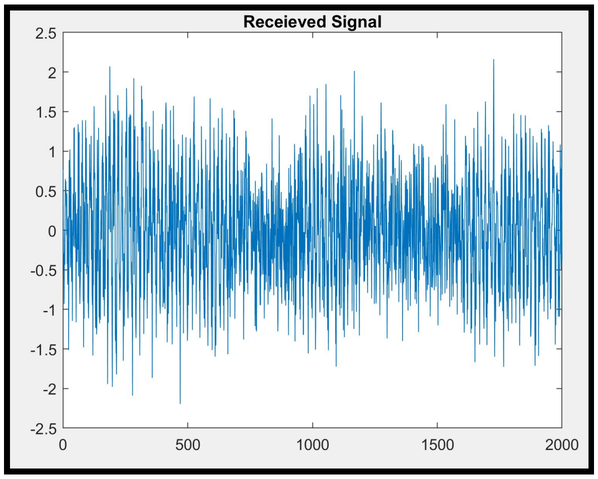
Graphs of M-ary ASK can be plotted by entering the command ***Part2_MaryASK(desired_SNR, true, desired_m, desired_amplitude, desired_threshold)*** on the command window.

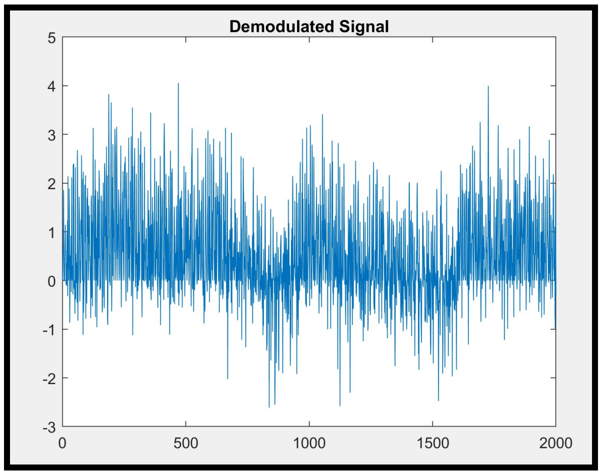


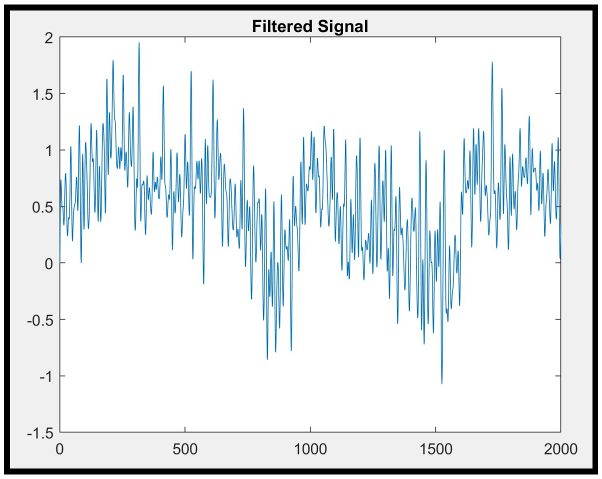
```
function errorRate = Part2_MaryASK(SNR, showFigures, m, amplitude, threshold)
```

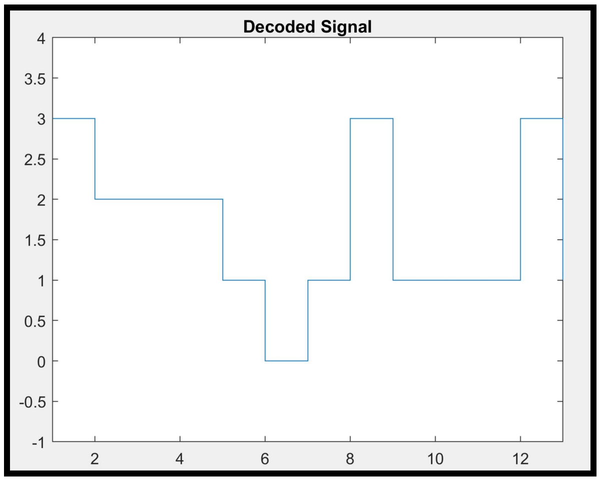

Data Wave Form


Modulated Signal


Receieved Signal

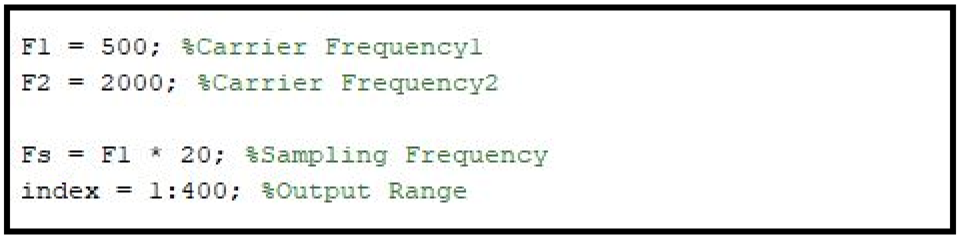

Demodulated Signal


Filtered Signal


Decoded Signal

Binary Frequency Shift Keying (BFSK)

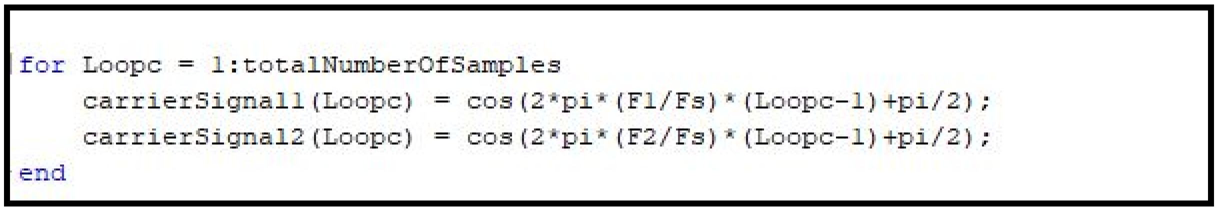
As this is an FSK, we are settling Frequency 1 to 500 and Frequency 2 to 2000. For this case, we are oversampling by 20 times.



```
F1 = 500; %Carrier Frequency1
F2 = 2000; %Carrier Frequency2

Fs = F1 * 20; %Sampling Frequency
index = 1:400; %Output Range
```

<u>Carrier Signal</u>

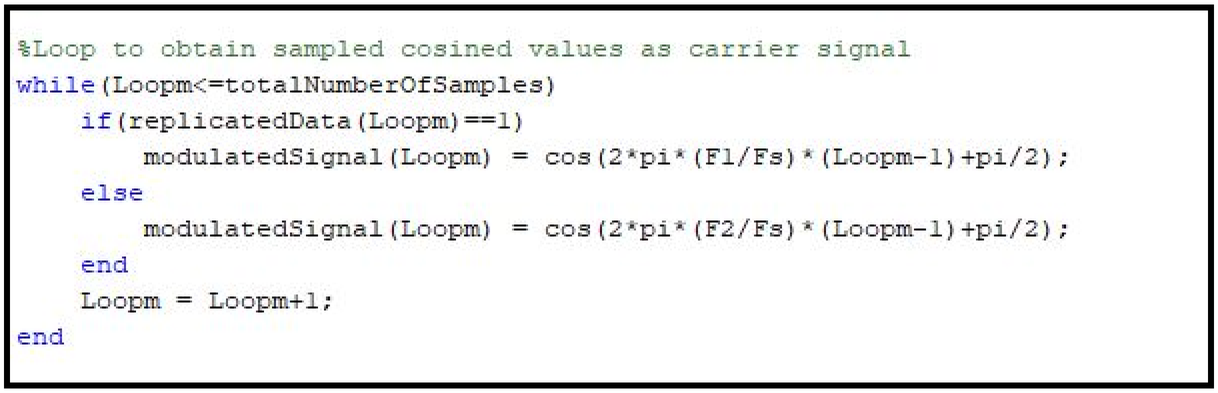
A carrier signal can be generated using the cos wave as shown below.



```
for Loopc = 1:totalNumberOfSamples
    carrierSignal1(Loopc) = cos(2*pi*(F1/Fs)*(Loopc-1)+pi/2);
    carrierSignal2(Loopc) = cos(2*pi*(F2/Fs)*(Loopc-1)+pi/2);
end
```

<u>Modulated Signal</u>

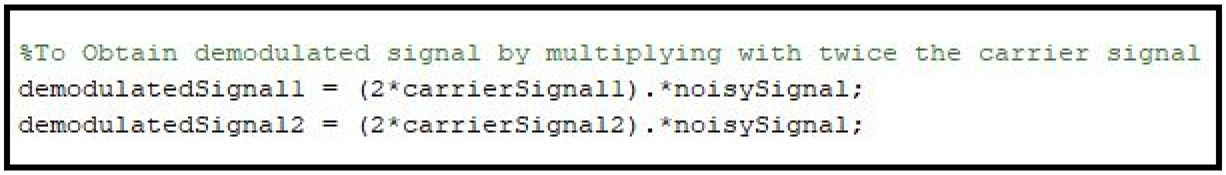
A modulated signal is generated by multiplying the original data with the carrier wave as shown below. A simple 'if-else' branch is used, in order to separate the binary 1s and 0s. F1 is used for binary 1s and F2 is used for binary 0s.



```
%Loop to obtain sampled cosined values as carrier signal
while(Loopm<=totalNumberOfSamples)
    if(replicatedData(Loopm)==1)
        modulatedSignal(Loopm) = cos(2*pi*(F1/Fs)*(Loopm-1)+pi/2);
    else
        modulatedSignal(Loopm) = cos(2*pi*(F2/Fs)*(Loopm-1)+pi/2);
    end
    Loopm = Loopm+1;
end
```

Demodulated Signal

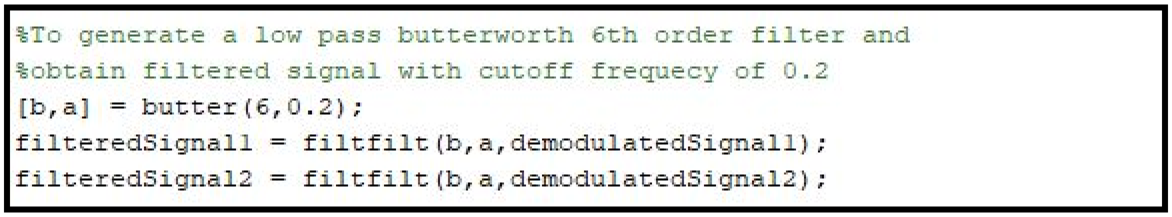
The demodulated signal is generated by multiplying twice the '**carrierSignal**' with the '**noisySignal**' generated from the previous function. It is to be noted this is done twice as we have 2 different frequencies.



```
%To Obtain demodulated signal by multiplying with twice the carrier signal
demodulatedSignal1 = (2*carrierSignal1).*noisySignal;
demodulatedSignal2 = (2*carrierSignal2).*noisySignal;
```

Filtered Signal

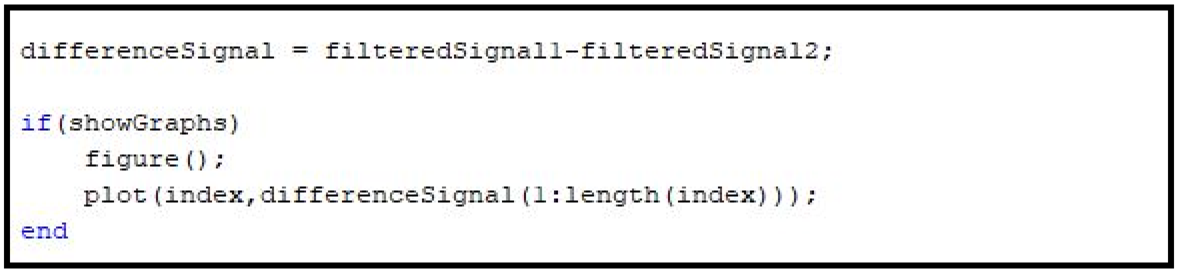
Here, we use Matlab's pre-defined functions like *butter(), fltfilt()* to do a low pass Butterworth filter and to generate the filtered signal by passing in the '**demodulatedSignal**'. It can be done as shown below. This again, is done for the 2 different filtered signal.



```
%To generate a low pass butterworth 6th order filter and
%obtain filtered signal with cutoff frequecy of 0.2
[b,a] = butter(6,0.2);
filteredSignal1 = filtfilt(b,a,demodulatedSignal1);
filteredSignal2 = filtfilt(b,a,demodulatedSignal2);
```

Difference Signal

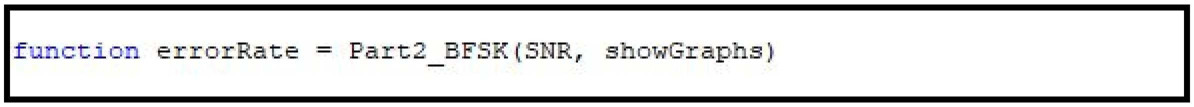
Since this an FSK modulation, we had to find the difference signal between both the filtered signals.



```
differenceSignal = filteredSignal1-filteredSignal2;

if(showGraphs)
    figure();
    plot(index,differenceSignal(1:length(index)));
end
```

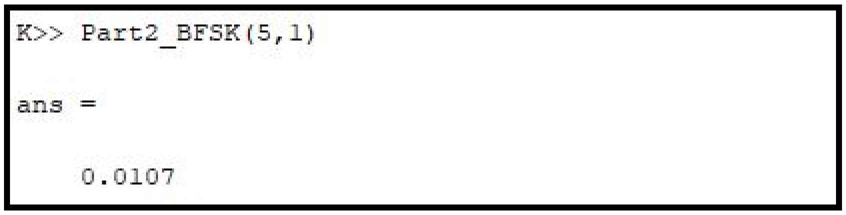
GRAPHS

Graphs of BFSK can be plotted by entering the command ***Part2_BFSK(desired_SNR, true)*** on the command window.

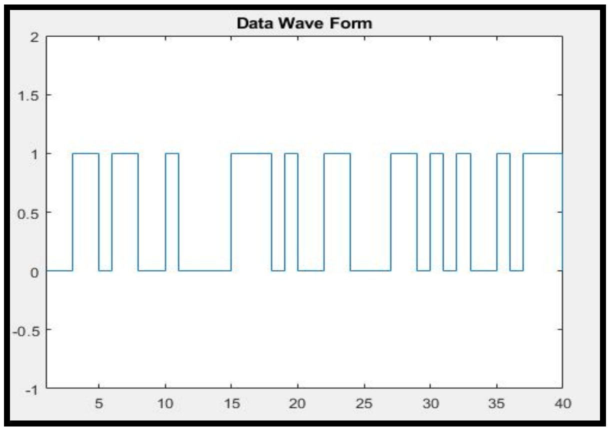


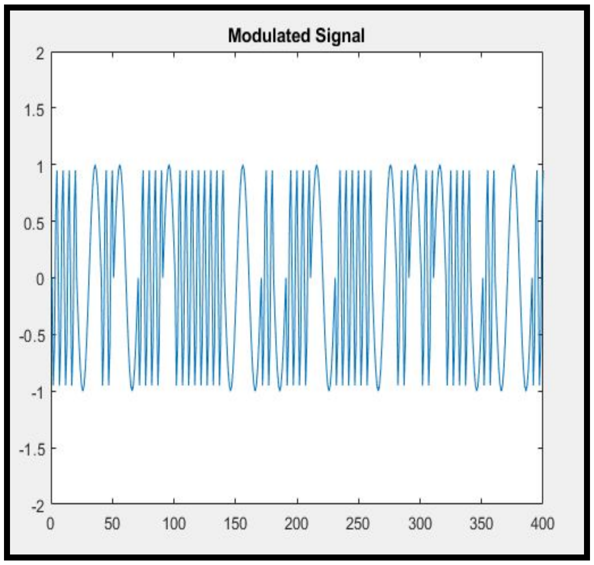
```
function errorRate = Part2_BFSK(SNR, showGraphs)
```

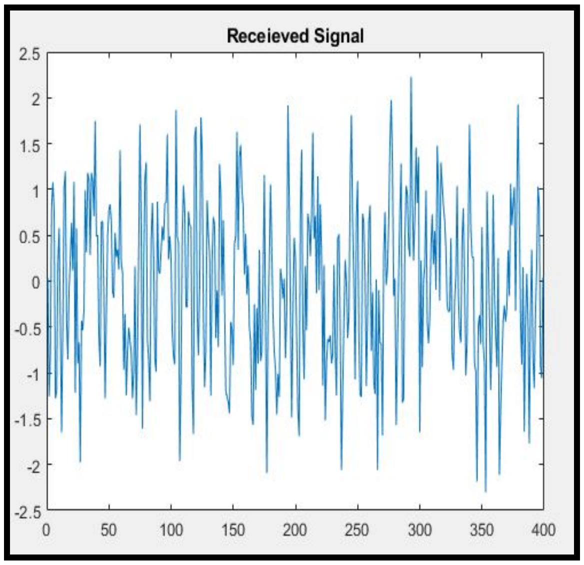
For example: If we enter Part2_BFSK(5, 1), the following graphs are plotted

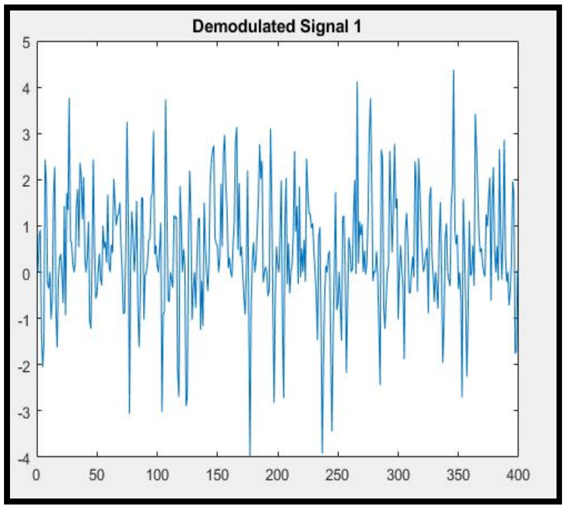


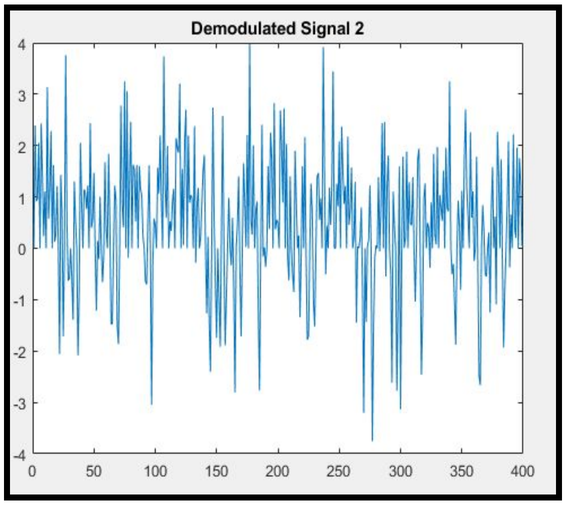
```
K>> Part2_BFSK(5,1)

ans =

    0.0107
```

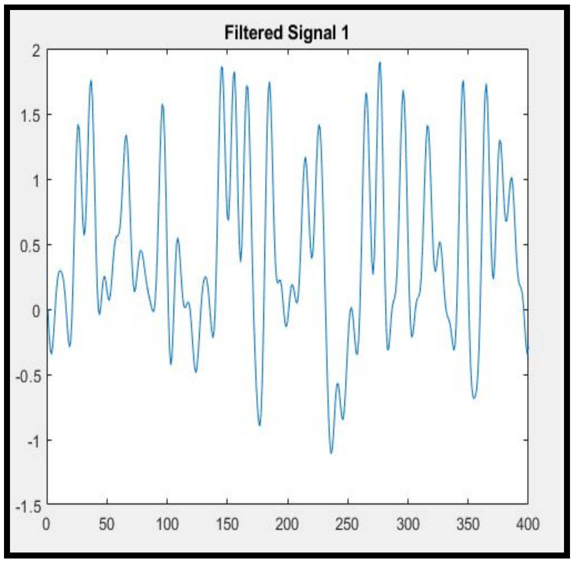


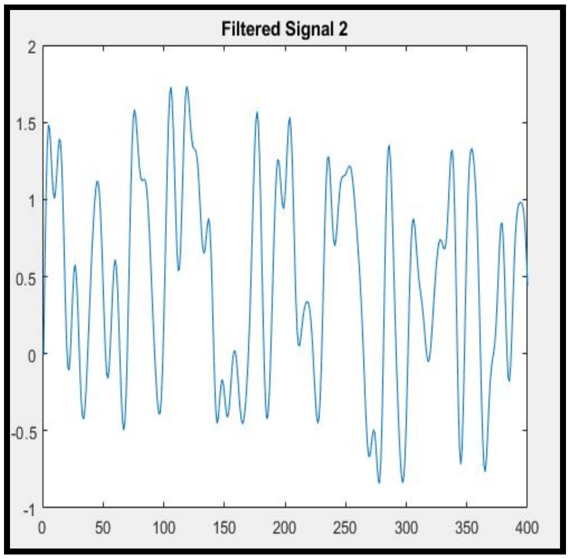


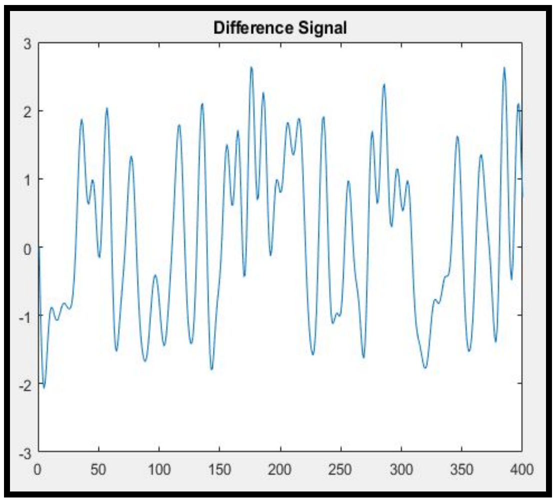


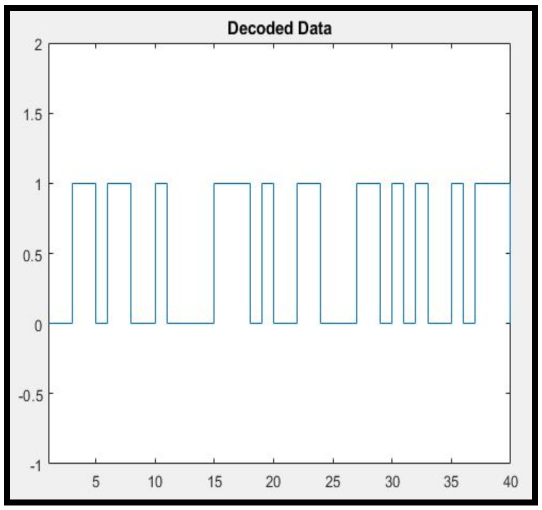

Demodulated Signal 1


Demodulated Signal 2


Filtered Signal 1


Filtered Signal 2

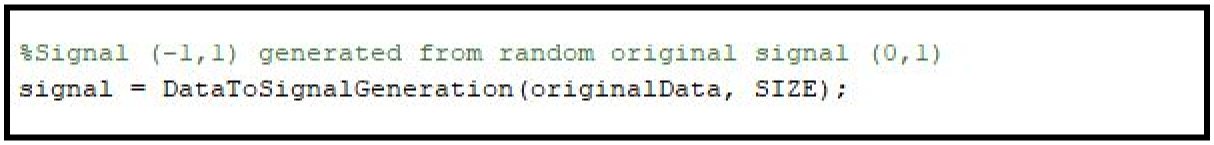

Difference Signal


Decoded Data

Binary Phase Shift Keying(BFSK)

Signal Generation

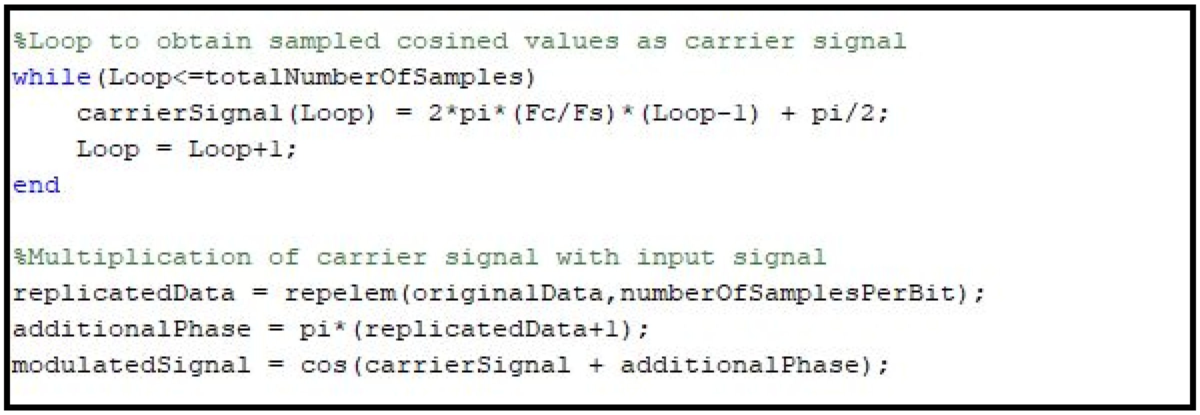
To generate Data signals in +1 and -1 format (as mentioned in the project file)



```
%Signal (-1,1) generated from random original signal (0,1)
signal = DataToSignalGeneration(originalData, SIZE);
```

Modulated Signal

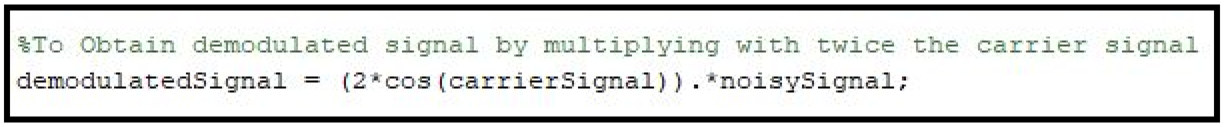
The modulated signal is produced by taking the cosine of the carrier signal added with an additional phase. Phase π when bit = 0 and 2π when bit = 1



```
%Loop to obtain sampled cosined values as carrier signal
while(Loop<=totalNumberOfSamples)
    carrierSignal(Loop) = 2*pi*(Fc/Fs)*(Loop-1) + pi/2;
    Loop = Loop+1;
end

%Multiplication of carrier signal with input signal
replicatedData = repelem(originalData,numberOfSamplesPerBit);
additionalPhase = pi*(replicatedData+1);
modulatedSignal = cos(carrierSignal + additionalPhase);
```

Demodulated Signal

Demodulated signal is produced by multiplying noisy signal with twice the carrier signal.



```
%To Obtain demodulated signal by multiplying with twice the carrier signal
demodulatedSignal = (2*cos(carrierSignal)).*noisySignal;
```

Error Rate

First, we obtain the midpoints from the received filtered signal and assign it to '**receivedData**' and we pass this received data to the '**DataDecoding**' function with a threshold of 0 (to achieve minimum error) to generate the '**decodedData**'. Since we have taken threshold as 0, anything below 0 will be taken as -1 and anything above will be taken as 1. Then we pass the decoded data to the '**ErrorRate**' function to calculate the bit error rate.

```
%Obtain midpoints from recieved filtered signal
receivedData = filteredSignal(numberOfSamplesPerBit/2:numberOfSamplesPerBit:totalNumberOfSamples);

%Use threshold logic to decode the received signal by setting threshold to 0.5
decodedData = DataDecoding(receivedData, SIZE, 0,[-1,1]);

%Calculate the bit error rate
errorRate = ErrorRate(signal,decodedData, SIZE);
```

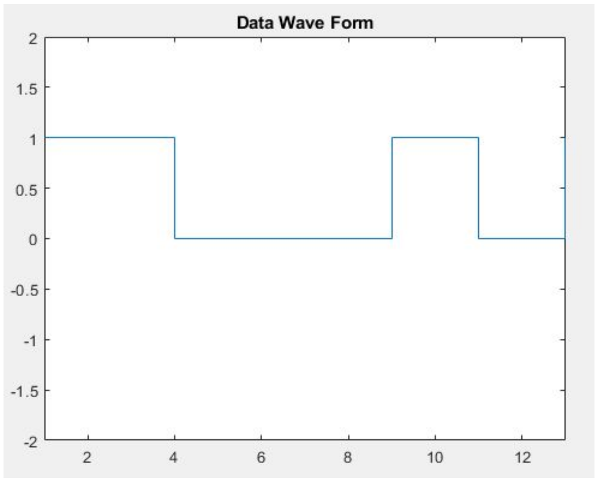
GRAPHS

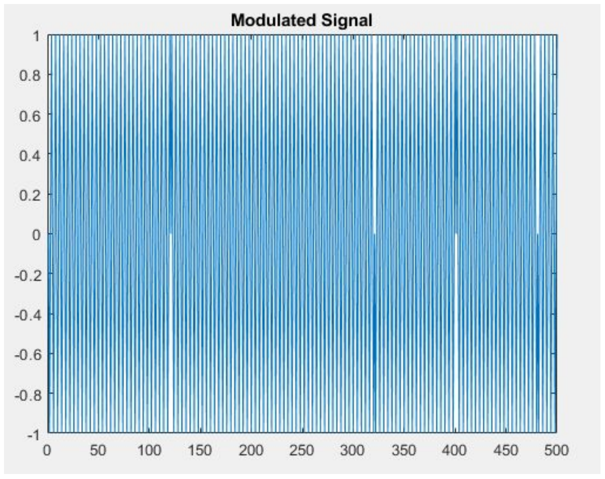
Graphs of BPSK can be plotted by entering the command ***Part2_BPSK(desired_SNR, true)*** on the command window.

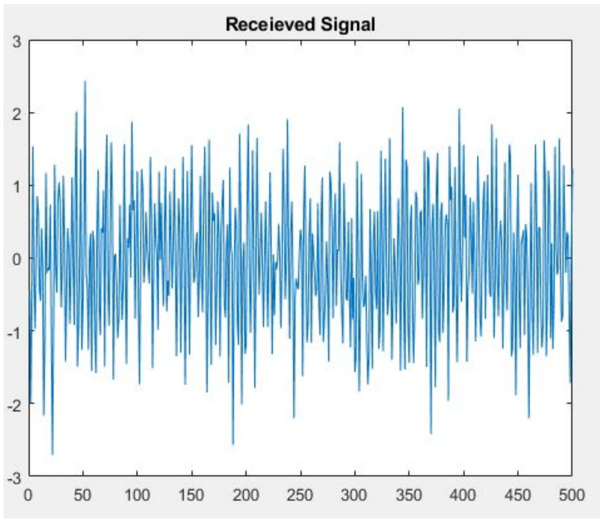
```
function errorRate = Part2_BPSK(SNR, showGraphs)
```

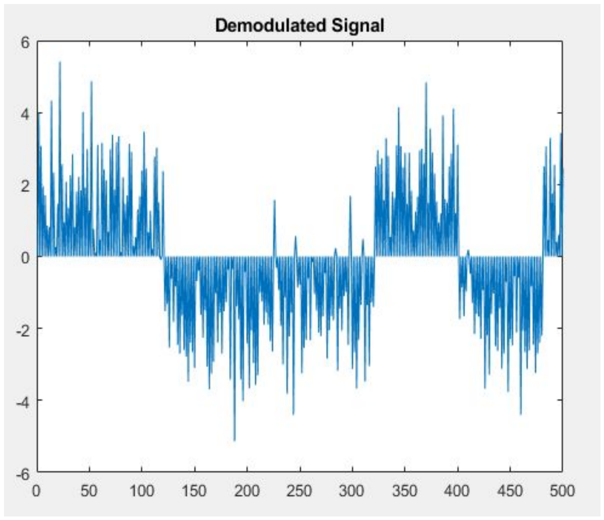
For example: If we enter Part2_BPSK(5, 1), the following graphs are plotted

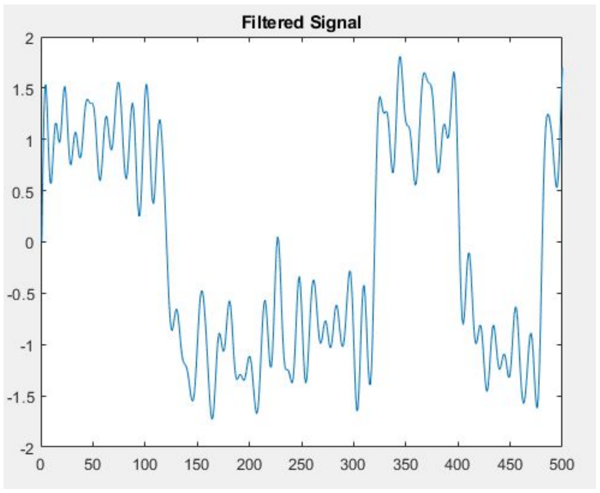
```
K>> Part2_BPSK(5,1)

ans =

    0.0029
```

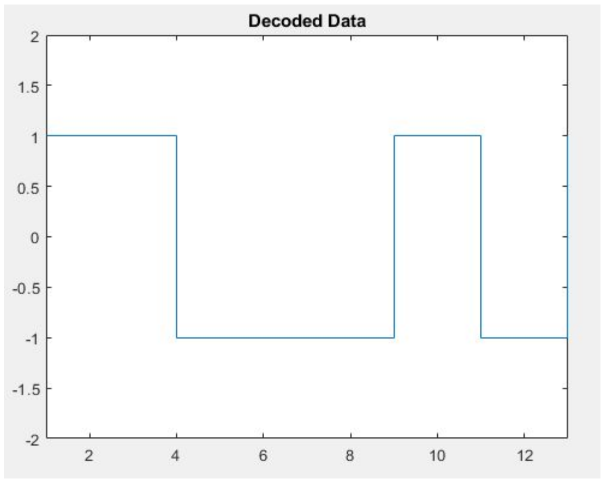






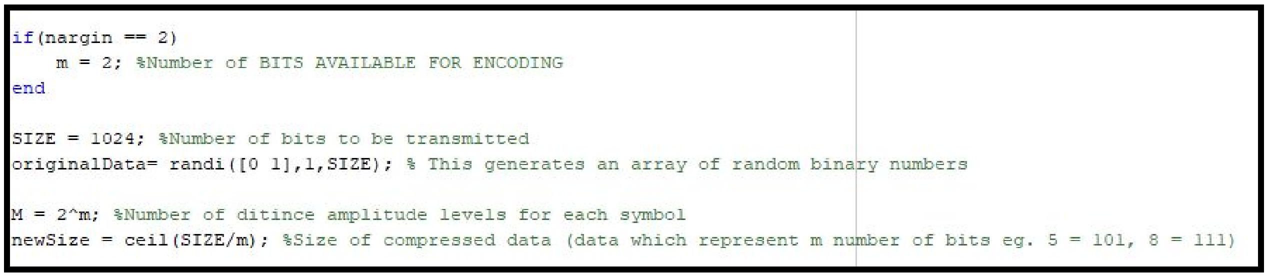






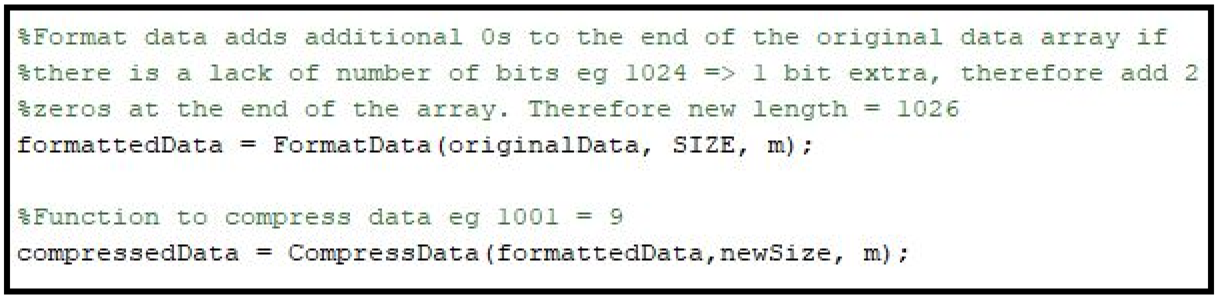
M-ary Phase Shift Keying

We have assigned m to 2 bits. This allows the selection of one of the M phase shifted versions of the carrier to transmit data. In this case, M is equal to 4.



```
if(nargin == 2)
    m = 2; %Number of BITS AVAILABLE FOR ENCODING
end

SIZE = 1024; %Number of bits to be transmitted
originalData= randi([0 1],1,SIZE); % This generates an array of random binary numbers

M = 2^m; %Number of ditince amplitude levels for each symbol
newSize = ceil(SIZE/m); %Size of compressed data (data which represent m number of bits eg. 5 = 101, 8 = 111)
```

Compress Data is performed to convert a chunk of bits to symbols, in this case a decimal symbol.



```
%Format data adds additional 0s to the end of the original data array if
%there is a lack of number of bits eg 1024 => 1 bit extra, therefore add 2
%zeros at the end of the array. Therefore new length = 1026
formattedData = FormatData(originalData, SIZE, m);

%Function to compress data eg 1001 = 9
compressedData = CompressData(formattedData,newSize, m);
```

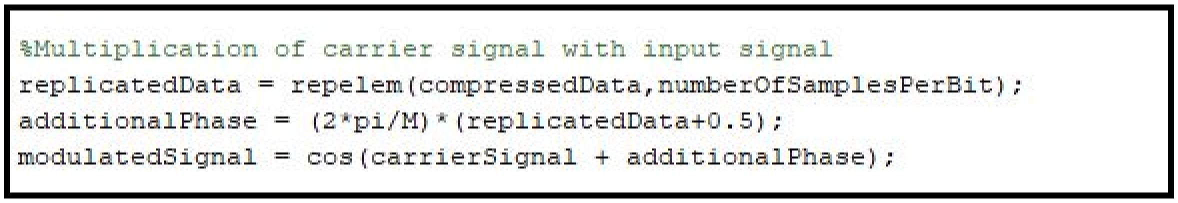
Modulated Signal

The modulated signal is produced by taking the cosine of the carrier signal added with an additional phase. The additional phase would like in mid way between its lower most value and upper most value. For example, for m=2 bit encoding. If value 2 is to be passed.

$$2+0.5 = 2.5$$

$$(2.5/4) * 2\pi = 5\pi/4$$

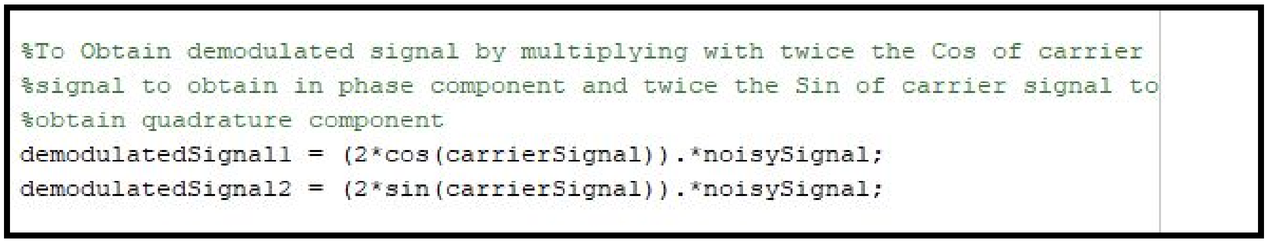
$5\pi/4$ lies between π & $3\pi/2$ which is in the 3rd quadrant. Hence any data received with phase difference with respect to the carrier signal lying in the 3rd quadrant would be considered as value 2.



```
%Multiplication of carrier signal with input signal
replicatedData = repelem(compressedData,numberOfSamplesPerBit);
additionalPhase = (2*pi/M)*(replicatedData+0.5);
modulatedSignal = cos(carrierSignal + additionalPhase);
```

Demodulated Signal

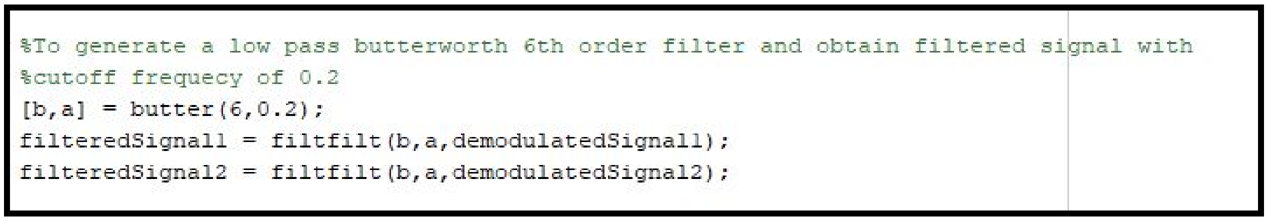
Demodulated signal 1 is produced by multiplying noisy signal with twice the Cos of carrier signal. This will help us obtain the in phase component portion. Demodulated signal 2 is produced by multiplying noisy signal with twice the Sin of carrier signal. This will help us obtain the quadrature component.



```
%To Obtain demodulated signal by multiplying with twice the Cos of carrier
%signal to obtain in phase component and twice the Sin of carrier signal to
%obtain quadrature component
demodulatedSignal1 = (2*cos(carrierSignal)).*noisySignal;
demodulatedSignal2 = (2*sin(carrierSignal)).*noisySignal;
```

<u>Filtered Signal</u>

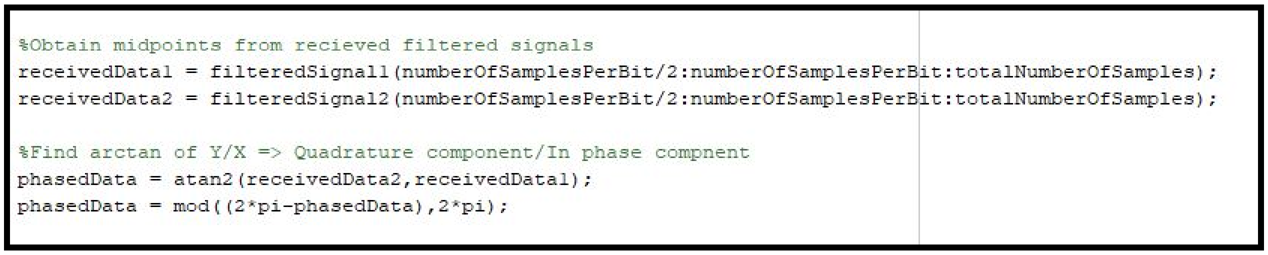
Here, we use Matlab's pre-defined functions like *butter(), fltfilt()* to do a low pass Butterworth filter and to generate the filtered signal by passing in the '**demodulatedSignal**'. It can be done as shown below.
We are doing this for demodulated signal 1 & 2 respectively.



```
%To generate a low pass butterworth 6th order filter and obtain filtered signal with
%cutoff frequecy of 0.2
[b,a] = butter(6,0.2);
filteredSignal1 = filtfilt(b,a,demodulatedSignal1);
filteredSignal2 = filtfilt(b,a,demodulatedSignal2);
```

<u>Error Rate</u>

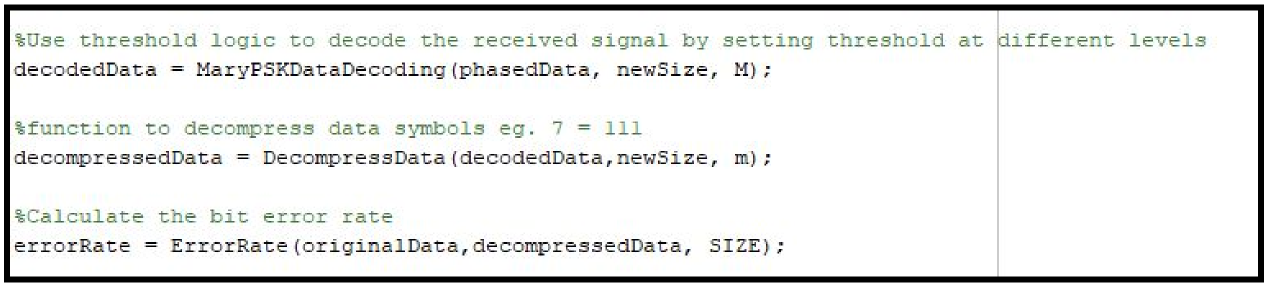
First, we obtain the midpoints from the received filtered signal and assign it to '**receivedData1**' and '**receivedData2**' respectively. In order to find the phase difference between the in phase component and the quadrature component we find $\tan^{-1}$ (quadData/inPhaseData) and bring it to the domain $[0, 2\pi]$ for easy analysis.



```
%Obtain midpoints from recieved filtered signals
receivedData1 = filteredSignal1(numberOfSamplesPerBit/2:numberOfSamplesPerBit:totalNumberOfSamples);
receivedData2 = filteredSignal2(numberOfSamplesPerBit/2:numberOfSamplesPerBit:totalNumberOfSamples);

%Find arctan of Y/X => Quadrature component/In phase compnent
phasedData = atan2(receivedData2,receivedData1);
phasedData = mod((2*pi-phasedData),2*pi);
```

<u>M-ARY PSK Data Decoding</u> (MaryPSKDataDecoding.m)

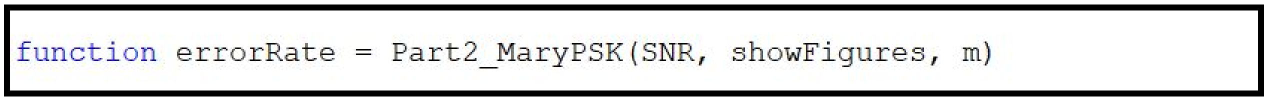
We pass '**phasedData**' through '**MaryPSKDataDecoding**' to generate '**decodedData**'. The function find the range in which the angle received lies and finally returns the corresponding value. Then we pass the decoded data to the '**decompressedData**' function to decompress the symbols back to binary data.



```
%Use threshold logic to decode the received signal by setting threshold at different levels
decodedData = MaryPSKDataDecoding(phasedData, newSize, M);

%function to decompress data symbols eg. 7 = 111
decompressedData = DecompressData(decodedData,newSize, m);

%Calculate the bit error rate
errorRate = ErrorRate(originalData,decompressedData, SIZE);
```

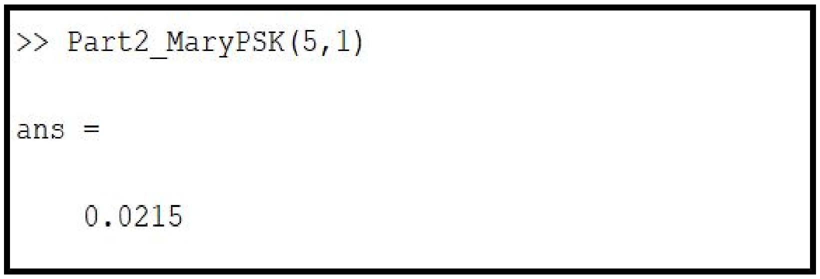
GRAPHS

Graphs of M-ary PSK can be plotted by entering the command ***Part2_MaryPSK(desired_SNR, true, desired_m)*** on the command window.

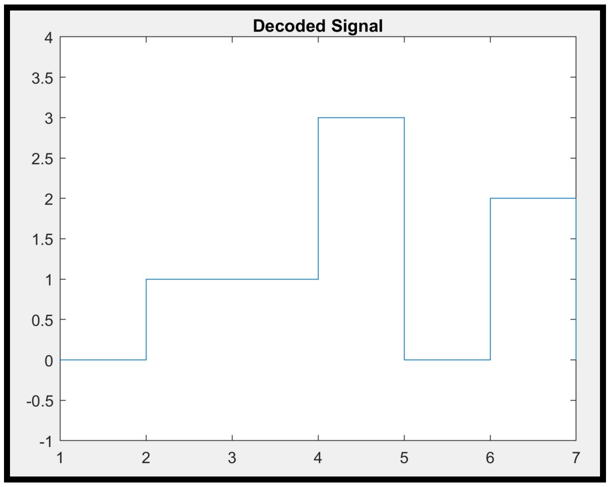


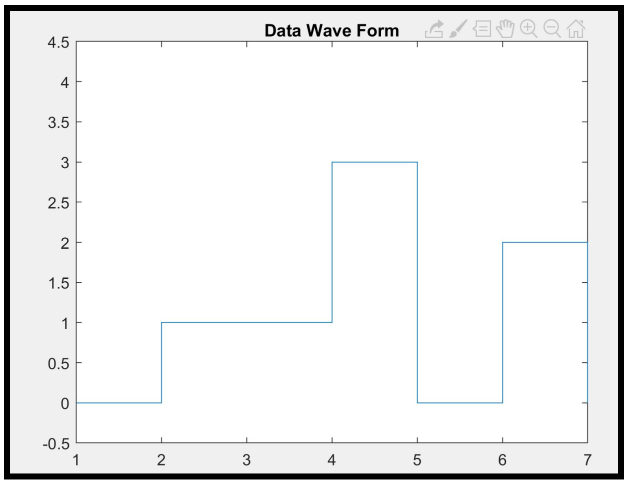
```
function errorRate = Part2_MaryPSK(SNR, showFigures, m)
```

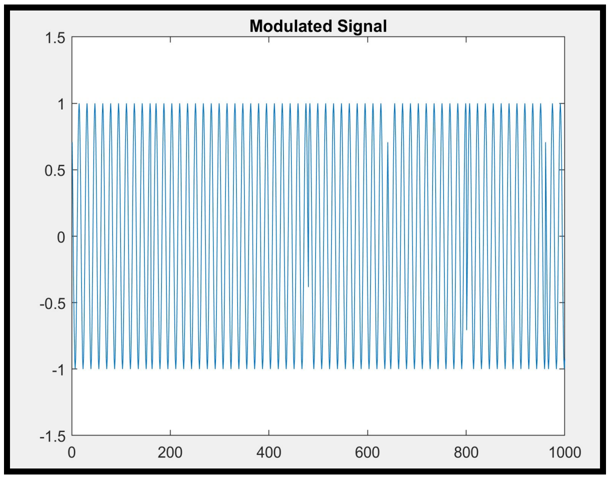
For example: If we enter Part2_MaryPSK(5, 1), the following graphs are plotted

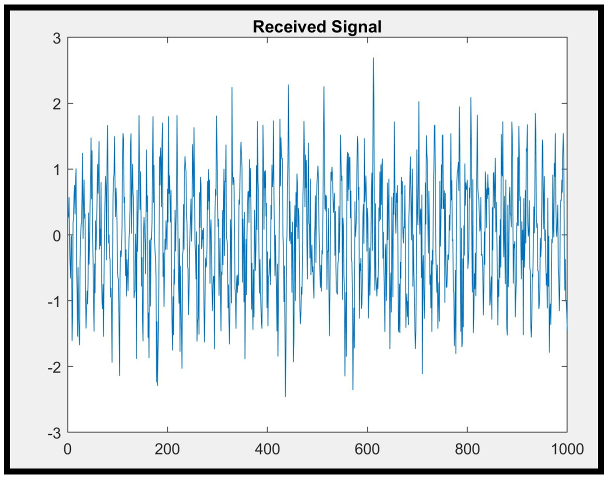


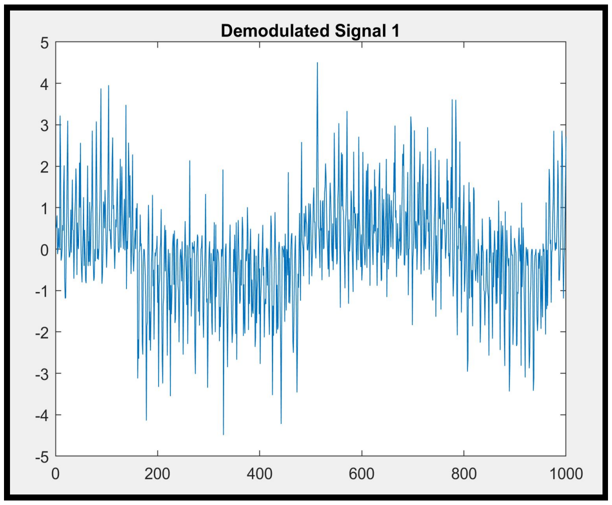
```
>> Part2_MaryPSK(5,1)

ans =

    0.0215
```

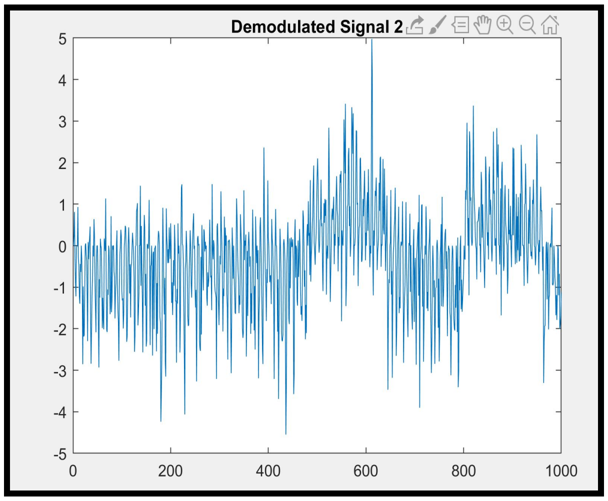


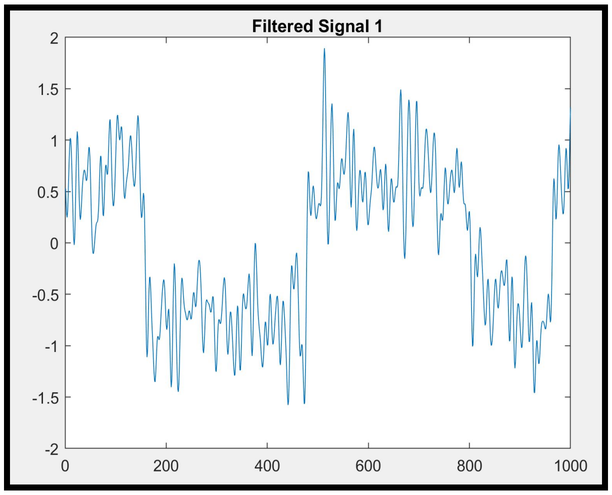


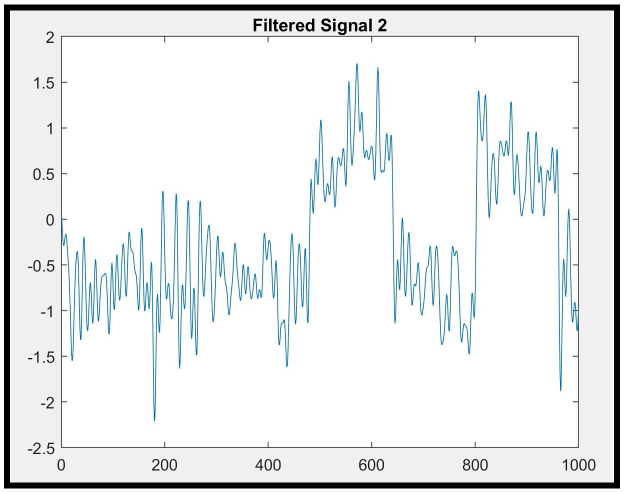

Modulated Signal
1.5
1
0.5
0
-0.5
-1
-1.5
0
200
400
600
800
1000


Received Signal
3
2
1
0
-1
-2
-3
0
200
400
600
800
1000


Demodulated Signal 1
5
4
3
2
1
0
-1
-2
-3
-4
-5
0
200
400
600
800
1000

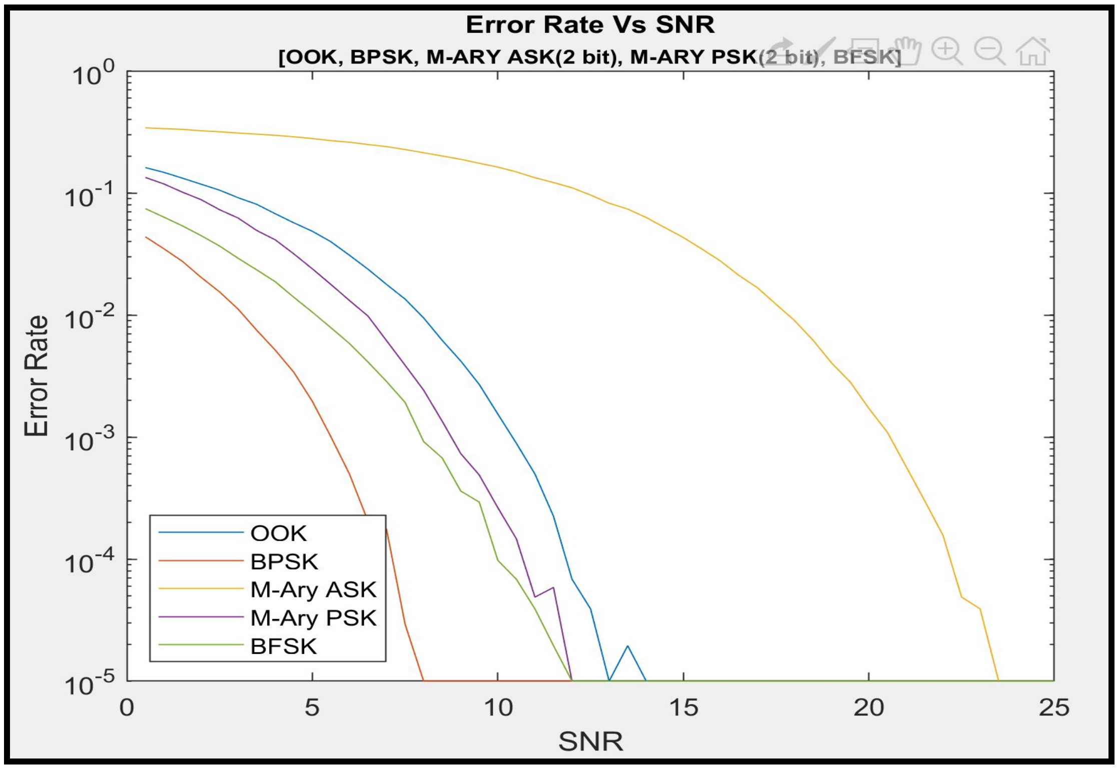

Demodulated Signal 2
5
4
3
2
1
0
-1
-2
-3
-4
-5
0
200
400
600
800
1000


Filtered Signal 1
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
200
400
600
800
1000


Filtered Signal 2
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
200
400
600
800
1000

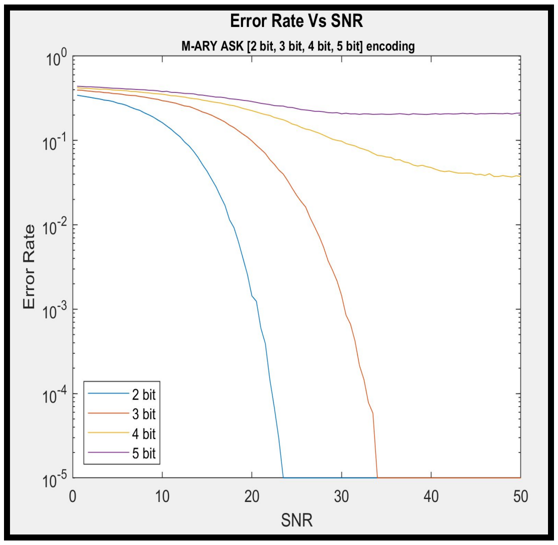
ANALYSIS

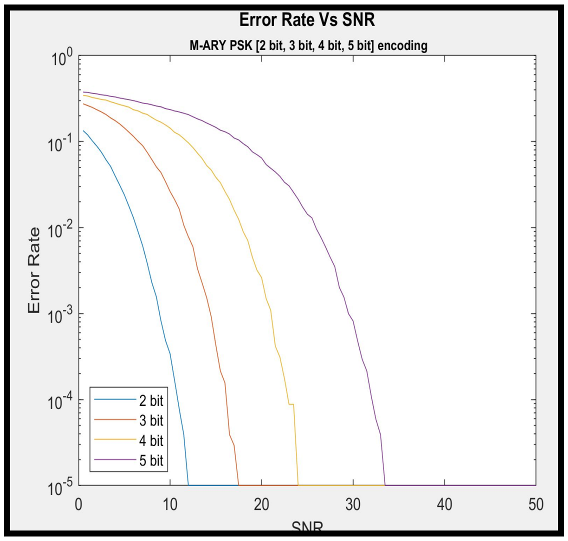
Comparison between different modulation/demodulation schemes



Observations

- It can be observed that BPSK performed the best in comparison to the other modulation techniques.
- M-Ary ASK encodes 2 bits for the same amplitude level. Therefore encoding 4 independent values over a range 0-1 whereas OOK encode 2 independent values in range 0-1. Thus we can conclude for Amplitude shift keying, encoding more number of bits for same amplitude range will lead to higher error rate per SNR.
- From the above graphs we can also conclude that a signal amplitude is more prone to error in comparison to the signal frequency and phase. Additionally, the phase of a signal is least prone to error under similar environments.
- M-Ary PSK or QPSK has considerably lower error rates that expected. Re-establishing the observation that the phase of a signal is least affected by noise.
- It is observed that for M-Ary PSK, if number of bits m=1 or M=2 , M-ary PSK has better performance than BPSK as they make use of different demodulation schemes. BPSK makes use a Bi-Polar detector whereas M-Ary PSK makes use of the phase difference to detect the transferred data providing more accuracy.





- It can be observed that, the increase in number bits, makes the system more prone to errors for both M-ary ASK & M-ary PSK, where the amplitude and threshold is constant throughout.
- In comparison, M-ary PSK does provide better results than M-ary ASK for any number of bits.

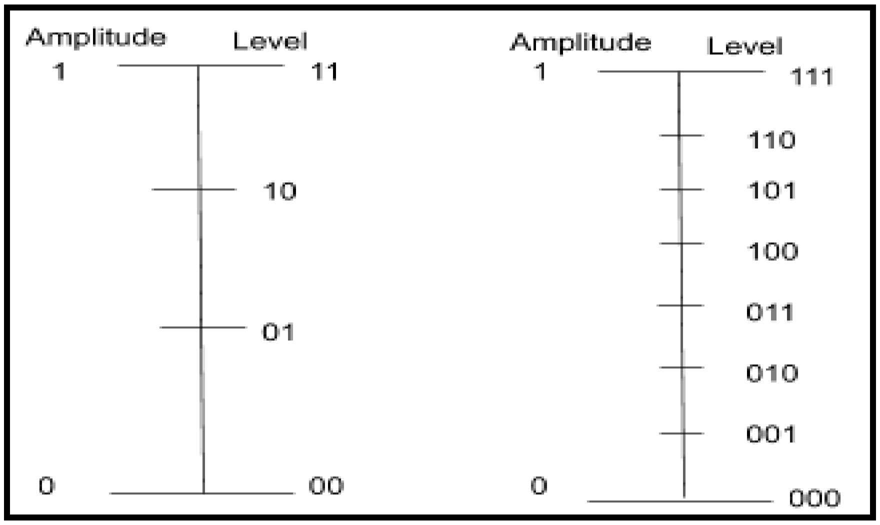
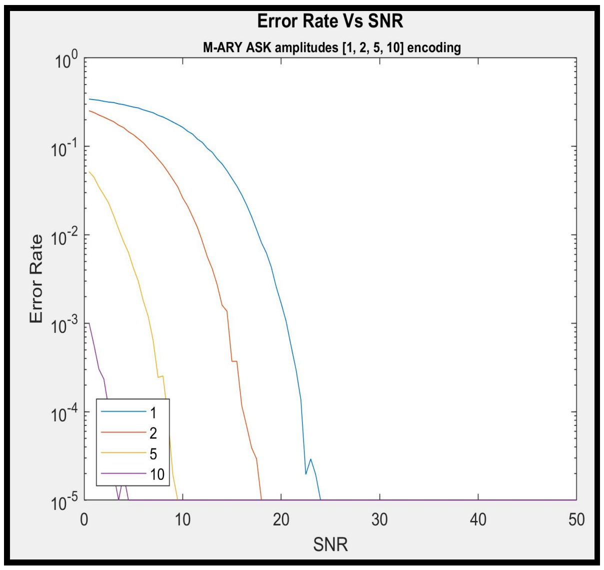
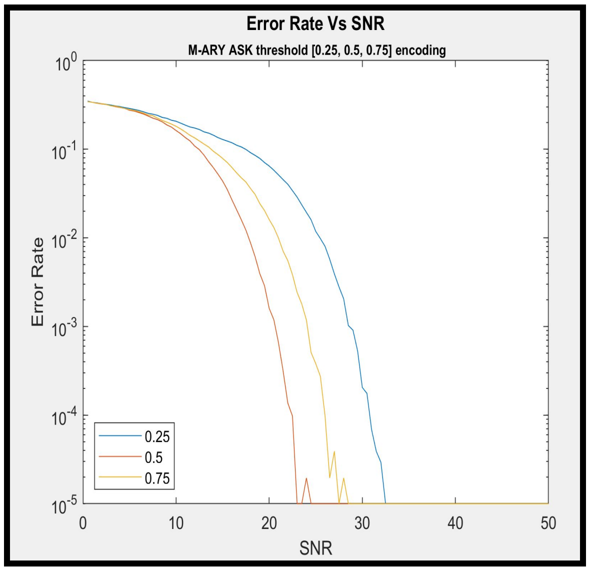


Fig1 **Fig2**

Let the amplitude be 1, in the first figure we have 4 levels, whereas in the second figure we have 8 levels. Therefore, the probability of an error occurrence in fig1 is less than that of fig 2.





- It can be observed that, the increase in amplitude, makes the system less prone to error for M-ary ASK (amplitudes).
- It can also be observed that, the error is the least when threshold is set to 0.5 for M-ary ASK (threshold).

Phase 3 - Error Detection & Correction

The performance of the proposed communication system can be improved further using error control codes. It will be experimented in this phase.

BACKGROUND

In the **Linear block codes**, the parity bits and message bits have linear combination. What this means is that, the resultant code word will be a linear combination of any two code words.

Let's take an example, where in which there are some blocks of data and each block contains k bits. These bits are then mapped to n bit blocks where n will be greater than k. The transmitter then adds redundant bits, (n-k). Code rate is known as the ratio between k and n ($r=k/n$). Parity bits are the last or first (n-k) and they aid us with error detection and correction.

Hamming codes are known as a type of linear error correcting codes. It can help detect upto two bit errors and help in correcting one bit errors without detection of uncorrected errors. In hamming codes, the source encodes the message by including redundant bits within the message. The redundant bits are just extra bits that are generated and inserted at certain positions in the message itself. This is done so to enable error detection and correction. When the receiver received this message, it performs some calculations to detect and find the bit position that contains the error.

Cyclic codes follow a certain cyclic property. The cyclic property of code words mentions that any cyclic-shift of a code word is also known as a code word. An (n,k) linear code is called cyclic if every cyclic shift of the code vector is also a code vector in C. Cyclic structure makes encoding and syndrome computation easy.

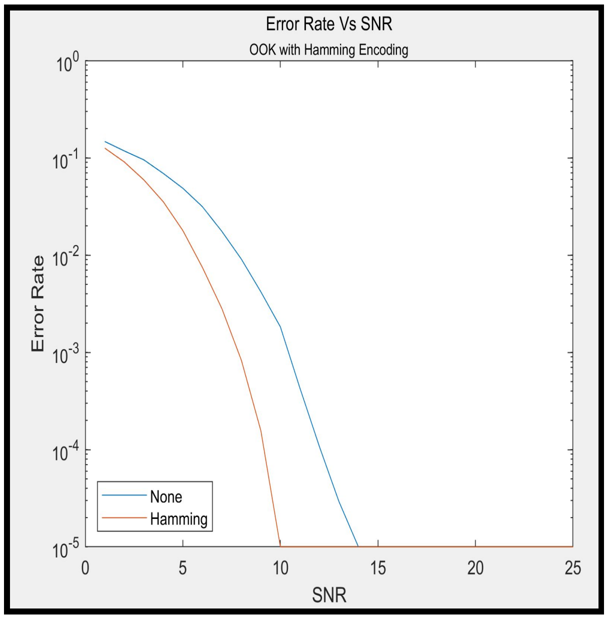
These are some cyclic properties:

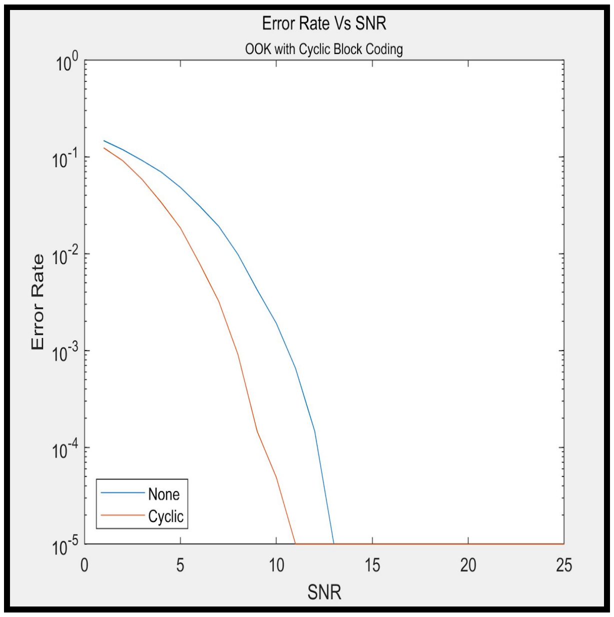
1. Within a set of code polynomials in C, there will be unique non-zero polynomial g(x) with minimal degree $r<n$.
2. Every code polynomial v(x) in an (n,k) cyclic code can be expressed uniquely as $v(x) = a(x)g(x)$.
3. An (n,k) linear cyclic code is completely specified by its nonzero code polynomial of minimum degree (g(x)).

Cyclic codes are used to help with error correction and correct double errors.

GRAPHS

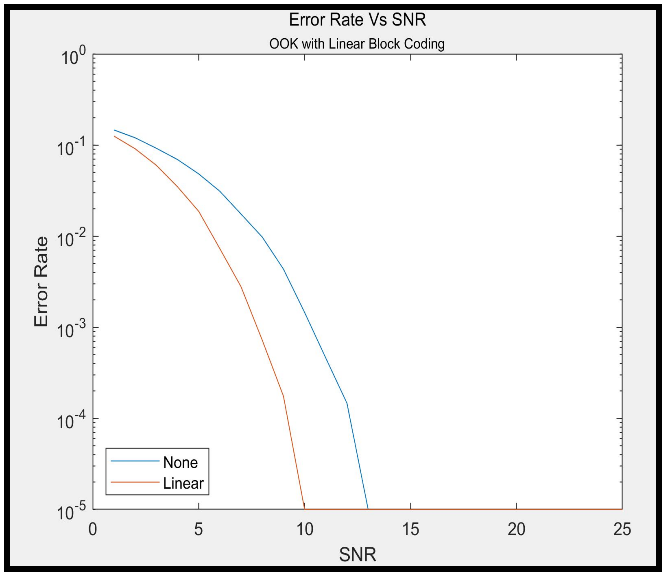
Given below are different the use of error control codes in OOK modulation and demodulation. OOK was chosen as it is observed to have the worst performance among all the binary modulation schemes.





-

Hamming **Cyclic**



Linear

It can be clearly observed from all the graphs above that the error control codes generally increase the performance by reducing the error rate.