Exp 1

By : shahd Ahmed Mohamed

Introduction to Probability of error calculation using Matlab.

❖ The Objective :

The objective of this experiment is to investigate the systematic procedure of the BER in communication systems by a simulation called the "Monte Carlo" simulation, where the idea behind a Monte Carlo simulation is simple, based on simulating the system repeatedly, where for each simulation count the number of transmitted symbols and symbol errors, then estimate the symbol error rate as the ratio of the total number of observed errors and the total number of transmitted bits.

In our experiment we performed the following procedures:

- Generate an array of random bits by using "randi" built in function
  From the code: randi([0 1],1,num_bit).

- Add noise (based on SNR) using a built in function
  From the code: N=sqrt(1/SNR(count))*randn(1,num_bit).

- Estimate the probability of error by detecting the noisy received signal in error (simple detector).
  From the code: (Y(k)> 0.5 && data(k)==0)||(Y(k)<0.5 && data(k)==1).

- Count the number of errors for each SNR.
  From the code: BER_sim(count)=Error.

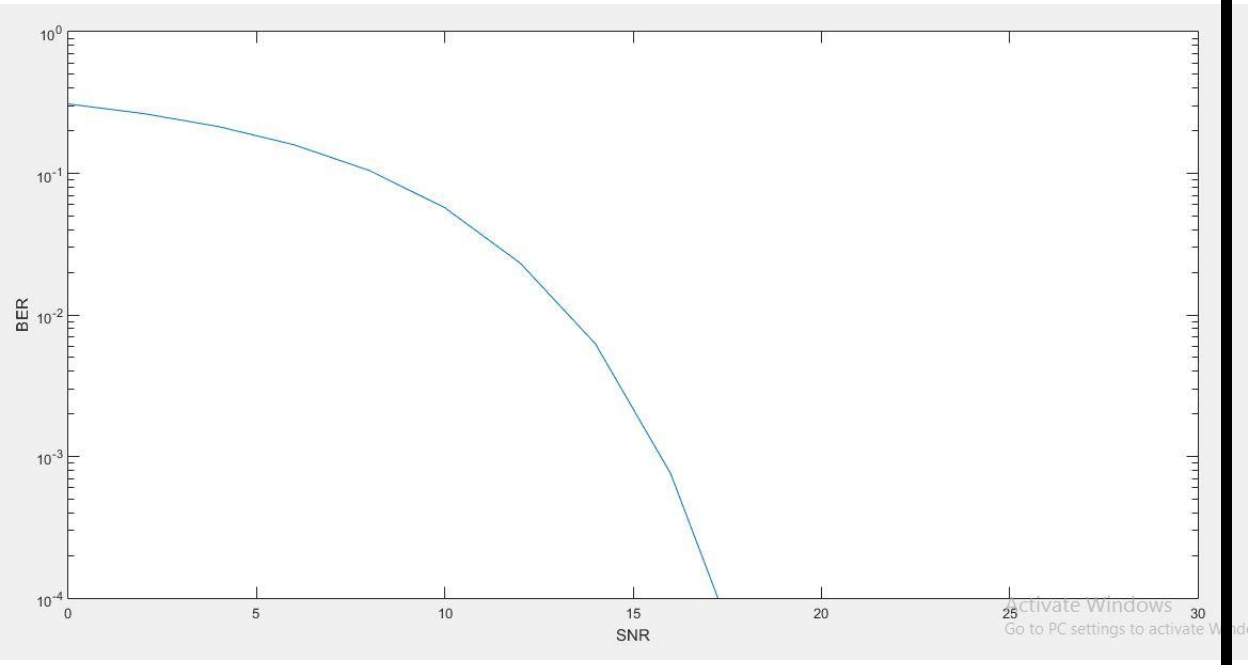
- By repeating the previous for large number of iterations to get the BER by averaging inside a for loop.

- All the previous steps are repeated per SNR.

The second objective is to investigate the performance of digital communication system with modeling the AWGN by its baseband equivalent like:

- Generate a gaussian distributed signal with total power equals to signal power/SNR.
- For simplicity, we always normalize the signal to unity so you can model the AWGN channel.

The result we got from our simulation is:



The Requirements :

- The previous figure shows the relation between the BER and SNR.
- Calculation of the transmitted signal Power:
  - Pt = norm(signal,2)2/length(signal) = 0.4994 watt = −3.0155 db
- Identify the meaning of 'measured' field :
  - It specifies that it measures the power of the signal before adding noise.
- At which value of SNR the system is nearly without error :
  - At SNR = 20 the BER is zero 2.

Experiment 2 Performance of Matched Filters and Correlators

By: Marwan Nabil Mohammed Basiouny Alfiel

No. 189

1 Introduction

The matched filter is the maximum-likelihood receiver in the presence of additive white Gaussian noise. Thus, for equal prior symbol probabilities, it will yield optimum bit-error performance. This is equivalent on the AWGN channel to maximizing the signal-to-noise ratio.

2 Modeling Matched filter receiver

We will study the decision-making process in the digital communications receiver which was modeled as shown below in Fig. 1

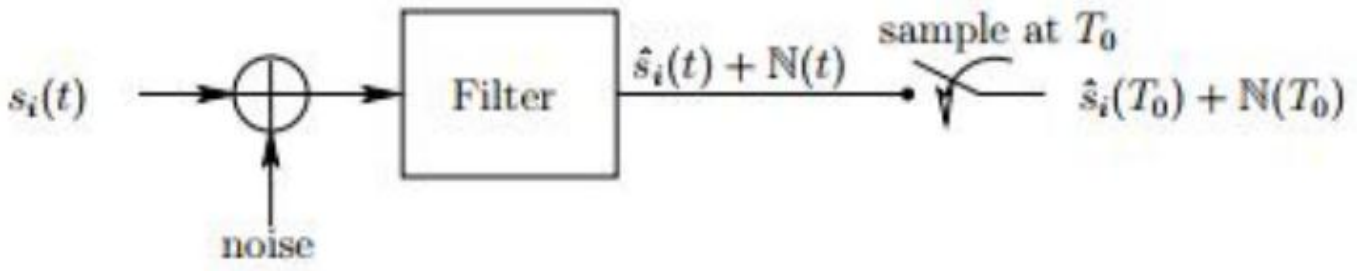


Figure 1: The receiver Model

1. It is well known, that the optimum receiver for an AWGN channel is the matched lter receiver. When the noise is absent, matched filter output is just signal energy i.e.

    - It matches the source impulse and maximize the SNR.
    - For non Gaussian noise, matched filter is not optimal.
    - Matched filter maximizes the SNR at the output of an FIR (also IIR) filter (even if the noise is non Gaussian).
    - Its operation depends on the symbols being used and the apriori

- Assuming that the threshold voltage: $V_t = \frac{S_1+S_2}{2}$
- Its BER vs SNR curve looks practically as shown in Fig. 3

3 Comparison between Simple detectors and Matched Filter and Correlators

For the Simple detector:

- At high SNR the BER is totally zero
- It's an ideal detector where we assume the probability of error is $P_e = \frac{1}{2} \times P(x \leq v_{th}) + P(y > v_{th})$
- Assuming that the threshold voltage: $V_t = \frac{S_1+S_2}{2}$
- Its BER vs SNR curve looks practically as shown in Fig. 2

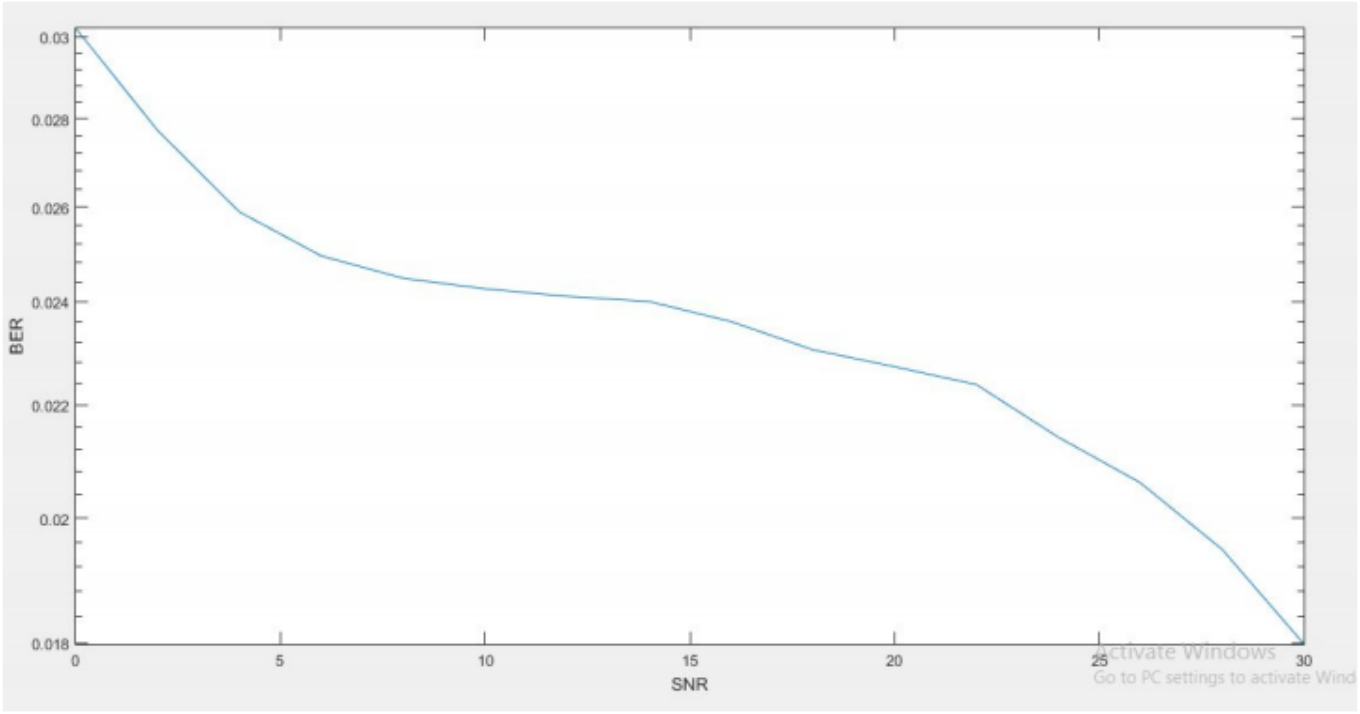


Figure 3: Matched Filter

4 Requirements

1. Calculation of the transmitted signal Power:

$$P_t = norm(signal, 2)^2 / length(signal) = 0.5012 \texttt{ watt} = -3 \texttt{ db}$$

2. At which value of SNR the system is nearly without error : At SNR = 30 the BER is 0.0180 which is near zero as shown in Fig. 4

```
BER =

  Columns 1 through 12

    0.0302    0.0277    0.0259    0.0249    0.0245    0.0243    0.0241    0.0240    0.0236    0.0231    0.0227    0.0224

  Columns 13 through 16

    0.0214    0.0206    0.0195    0.0180
```


Figure 4: BER values for different SNR

3. Since the code should run for reasonable time (few minutes).We've tried 2 cases at which max-run=20 and 2 as shown in Figures 5 and 6

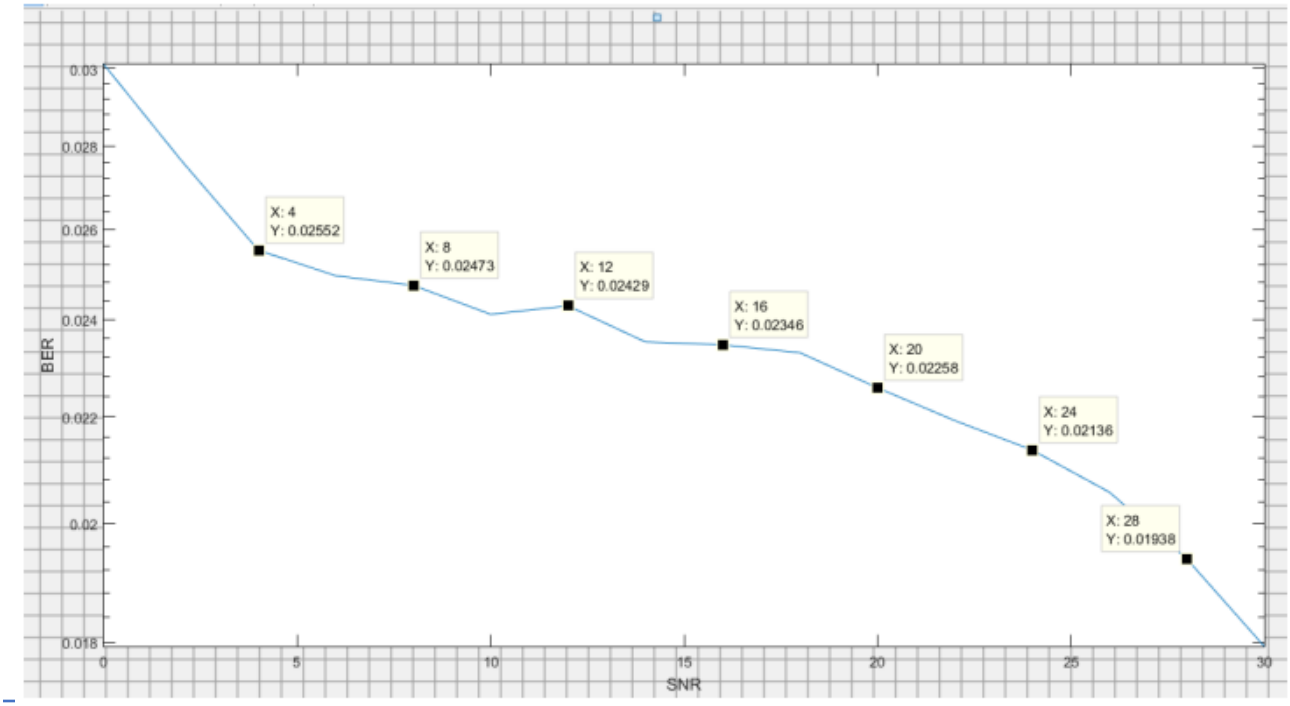


Figure 5: for max run= 20

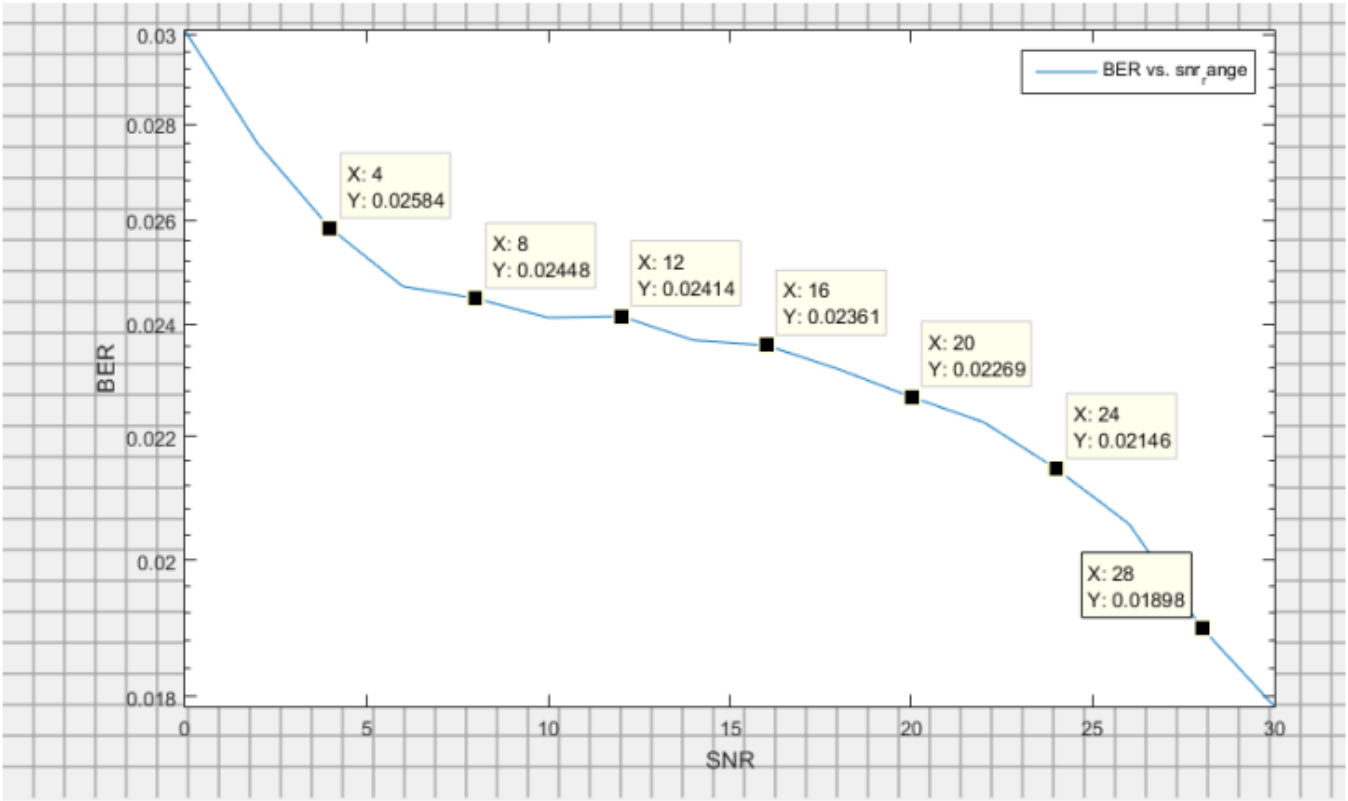


Figure 6: for max run= 2

Second_Report_Experiment (3) With Details

By : Moustafa Raafat Moustafa

Performance of different modulation types

The objective is to compare the performance of different modulation types (ASK-FSK-PSK).

Transmitter:

First, I want to generate a sequence of random 0's and 1's bits, which indeed is the OOK representation. I did it using this command line

```
TX = randi([0,1],1,10000);
TX_ASK = TX;
```

After that, I generate PRK bits, and how can we do that using, what we already have from OOK bits? Using this simple command line

```
TX_PSK = (2*TX-1);
```

Now I represent it as 1's and -1's instead of 1's and 0's, which is the PRK representation.

Last scheme is FSK, which is a little trickier. The idea is to send 1 if the bit to send = 0, otherwise send i, where i is the complex number. I did that using the following command lines

```
TX_FSK = zeros(1,length(TX)); %I need to use for loop to do it, so we prepare the array of 0's here first


%%FSK generation
for i= 1:length(TX)
    if TX(i) == 0
        TX_FSK(i) = 1;
    else
        TX_FSK(i) = j; %where j is the complex number sqrt(-1)
    end
end
```

Also, I need to create initial arrays to save BER of each modulation technique in. I did that using the following command.

```
ber_ASK=[];ber_PSK=[];ber_FSK=[]; %prepare the ber arrays for every mod. tech.
```

Channel:

Now, I want to add AWGN to the bits, or symbols in case of FSK, I did that, using the following command lines

```
%%here we add awgn and we also use measured to calc the power beacuae
    %%it's not unity
        RX_ASK = awgn (TX_ASK,snr,'measured');
        RX_PSK = awgn (TX_PSK,snr,'measured');
        RX_FSK = awgn (TX_FSK,snr,'measured'); %NOTE :  If TX_FSK is complex, awgn adds
        %complex noise which is the case here
```

Receiver:

Here, I want to decide whether the sequence w '1' or '0', I did that, using the following command lines for OOK and PRK techniques

```
for i= 1:length(TX)
        %for ASK the threshold is 0.5
            if  RX_ASK(i) <= 0.5
                RX_ASK(i) = 0;
            else
                RX_ASK(i) = 1;
            end

        %for BPSK the threshold is 0
            if RX_PSK(i) <=0
                RX_PSK(i)=0;
            else
                RX_PSK(i)=1;
            end
```

Again, FSK is a little trickier,

```
%for FSK the threshold is real 0.5 and imag 0.5 because we
deal
          %here with complex numbers and both parts are
affected as  mentioned
             if imag(RX_FSK(i)) <= 0.5 && real(RX_FSK(i)) >=
0.5
                 RX_FSK(i)=0;
             else
                 RX_FSK(i)=1;
             end
        end
```

At last, I compare the sent and detected bits using biterr function to find BER which we're interested in all cases. I did that using the following command lines,

```
[n1,BER1] = biterr (TX,RX_ASK);
[n2,BER2] = biterr (TX,RX_PSK);
[n3,BER3] = biterr (TX,RX_FSK);
%% here we save them in a matrix for each snr
ber_ASK = [ber_ASK BER1];
ber_PSK = [ber_PSK BER2];
ber_FSK = [ber_FSK BER3];
```

Plotting:

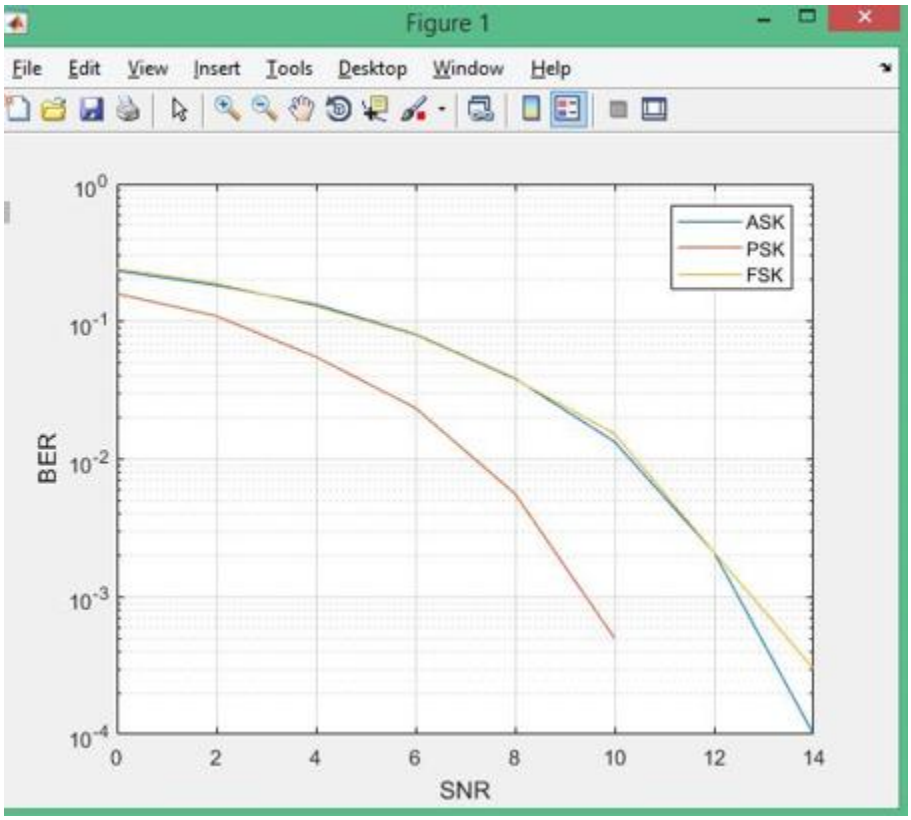
Here we plot snr vs each BER together on the same figure to be able to compare. I did that, using the following command lines

```
snr = (0:2:30)';

semilogy(snr,[(ber_ASK)' (ber_PSK)' (ber_FSK)'])

xlabel ('SNR'); ylabel ('BER'); legend('ASK','PSK','FSK');
grid
```

This was the result,



As we see BPSK(PRK) has the performance as it has the least BER among the 3 types, while OOK and orthogonal-FSK almost have the same performance.

Note: the other requirements are presented in the other report.

Bonus#1 of experiment (3)

The objective is to evaluate the same curves using the MATLAB built-in functions.

Transmitter:

Ask: I generate it using the following command lines

```
TX = randi([0,1],1,10000); %random sequence of zeros and
ones
TX_ASK = TX;                %OOK is the same sequence
```

PRK: I generate it using the following command line

```
TX_PSK = pskmod(TX,M,pi); %mod of psk using the built in fn.
```

FSK: I generate it using the following command line

```
TX_FSK = fskmod(TX,M,1,2,4); %mod of fsk using the built in
fn.
```

Also, I need to create initial arrays to save BER of each modulation technique in. I did that using the following command.

```
ber_ASK=[];ber_PSK=[];ber_FSK=[]; %prepare the ber
arrays for every mod. tech.
```

Channel:

Now, I want to add AWGN to the bits, or symbols in case of FSK, I did that, using the following command lines

```
%%here we add awgn and we also use measured to calc
the power beacuae
    %%it's not unity
        RX_ASK = awgn (TX_ASK,snr,'measured');
        RX_PSK = awgn (TX_PSK,snr,'measured');
        RX_FSK = awgn
(TX_FSK,snr,'measured'); %NOTE :  If TX_FSK is
complex, awgn adds
        %complex noise which is the case here
```

Receiver:

Here, I want to decide whether the sequence w '1' or '0', I did that, using the following command lines for OOK technique

```
for i= 1:length(TX)
        %for ASK the threshold is 0.5
            if  RX_ASK(i) <= 0.5
                RX_ASK(i) = 0;
            else
                RX_ASK(i) = 1;
            end
end
```

FOR PRK and FSK, I use built-in function to detect them, using the following command lines

```
RX_FSK = fskdemod(RX_FSK,M,1,2,4);  %Demod of fsk using the
built in fn.

RX_PSK = pskdemod(RX_PSK,M,pi);     %Demod of psk using the
built in fn.
```

At last, I compare the sent and detected bits using biterr function to find BER which we're interested in all cases. I did that using the following command lines,

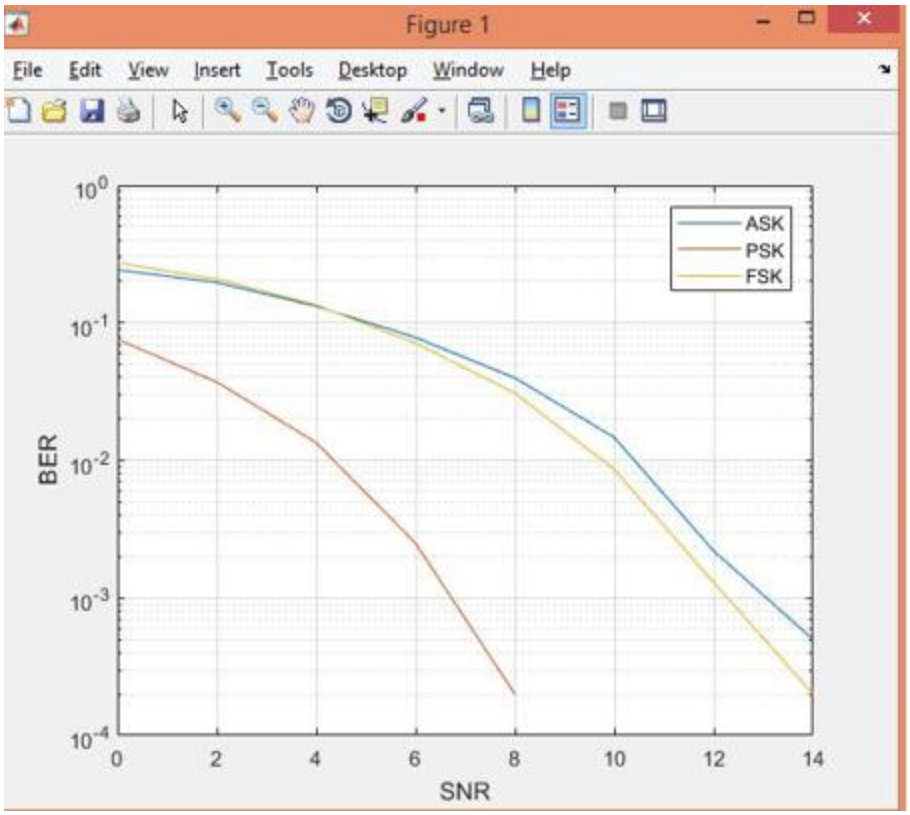
```
[n1,BER1] = biterr (TX,RX_ASK);
[n2,BER2] = biterr (TX,RX_PSK);
[n3,BER3] = biterr (TX,RX_FSK);
%% here we save them in a matrix for each snr
ber_ASK = [ber_ASK BER1];
ber_PSK = [ber_PSK BER2];
ber_FSK = [ber_FSK BER3];
```

Plotting:

Here we plot snr vs each BER together on the same figure to be able to compare. I did that, using the following command lines

```
snr = (0:2:30)';
semilogy(snr,[(ber_ASK)' (ber_PSK)' (ber_FSK)'])
xlabel ('SNR'); ylabel ('BER'); legend('ASK','PSK','FSK');
grid
```

This was the result,



It’s also obvious here that BPSK has the best performance as it has the lowest BER among the 3 types, while OOK and orthogonal-FSK almost have the same performance.

Bonus#2 of experiment (3)

The objective is to evaluate the probability of error of 16QAM modulation.

Transmitter:

First, I want to generate sequence of symbols of a 16QAM technique. I did that, using the following command lines

```
c = [-3-3i -3-1i -3+3i -3+1i -1-3i -1-1i -1+3i -1+1i 3-3i
3-1i 3+3i 3+1i 1-3i 1-1i 1+3i 1+1i ];

M = length(c); %which is 16

data = randi([0 M-1],10000,1); %random sequence of symbols
fro 0 to 15

TX_QAM = genqammod(data,c);  %mod of 16QAM using the built
in fn.
```

Also, I need to create initial arrays to save BER of each modulation technique in. I did that using the following command line

```
ber_QAM=[]; %prepare the ber arrays
```

Channel:

Now, I want to add AWGN to the symbols, I did that, using the following command lines

```
for snr = 0:2:30

%%here we add awgn and we also use measured to calc the
power beacuae

    %%it's not unity

        RX_QAM = awgn(TX_QAM,snr,'measured');
```

Receiver:

Here, I want to decide what the symbol was, I did that, using the following command lines

```
RX_QAM = genqamdemod(RX_QAM,c);  %Demod of 16QAM using the
built in fn.
```

At last, I compare the sent and detected bits using biterr function to find BER which we're interested in. I did that using the following command lines,

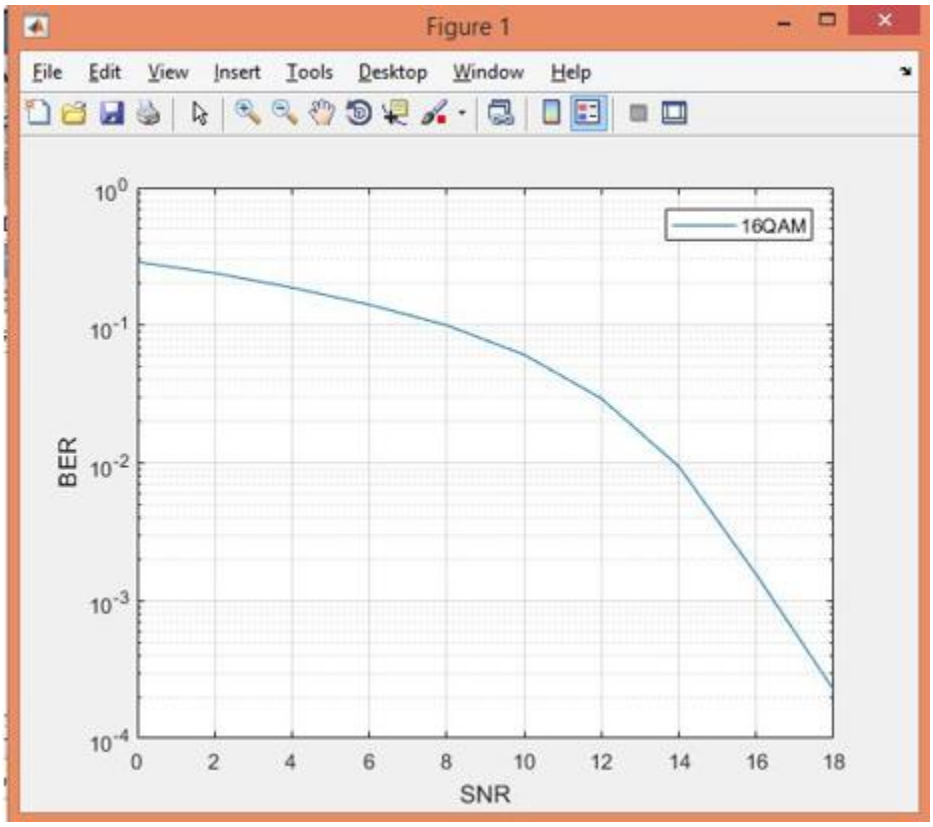
```
        [n1,BER1] = biterr (data,RX_QAM);
        %% here we save them in a matrix for each snr
        ber_QAM = [ber_QAM BER1];
end
```

Plotting:

Here we plot snr vs each BER together on the same figure to be able to compare. I did that, using the following command lines

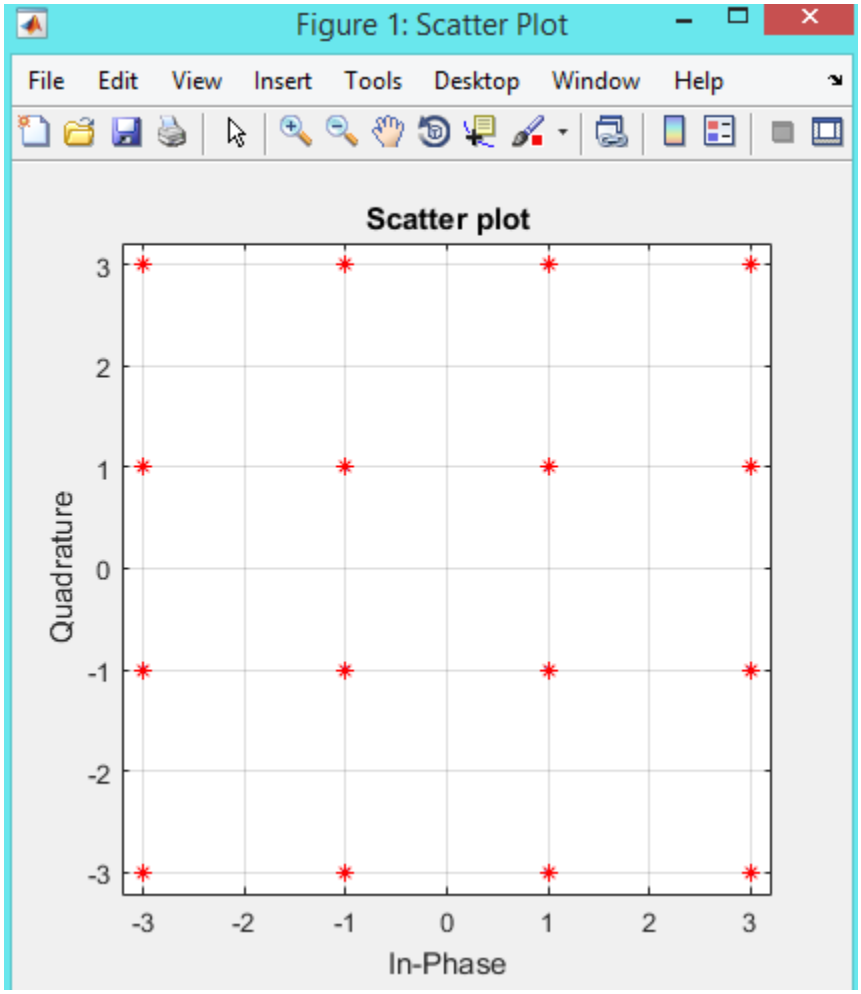
```
%%here we plot snr vs BER
snr = (0:2:30)';
semilogy(snr,ber_QAM)
xlabel ('SNR') ; ylabel ('BER') ; legend('16QAM') ; grid
```

This was the result,



Also, I wrote a simple command lines if we're interested in plotting the constellation of 16QAM, as follows

```
h = scatterplot(TX_QAM);
hold on
scatterplot(c,[],[],'r*',h)
grid
```



Brief explanation for EXP_4 code

By: Asmaa Mohamed Salah Eldien

The code divided into five main blocks :
1- Generating the sine wave with amplitude 1 V and frequency 2 Hz
2- Sampling this sine wave using sampling frequency 4000 Hz
3- Quantizing the sampled signal using "FI" command and varying n from (3,4,5,10)
4- Converting the quantized signal to binary using ".bin" command
5- Calculating Mean Square Error using a "for loop" and the equation given in PDF

Reconstruction from over sampling :
putting Fs= Hz, as in under sampling , Fs must be larger than 'Nyquist Rate= 2fm'

Reconstruction from minimum sampling :
putting Fs=10 Hz ,as in minimum sampling , Fs must be equals 'Nyquist Rate= 2fm'

Reconstruction from under sampling :
putting Fs=5 Hz, as in under sampling , Fs must be smaller than 'Nyquist Rate= 2fm'

BONUS:

1-Change m ,n values to change resolution
2-Quantize the sampled signal using **'quantizenumeric' command**
3- ***Quantize the sampled signal using*** **'command' command and produce compressed signal**

EXP 5

By : Asmaa Moustafa Saad

- The Delta Modulation Project comprises 4 codes files; code1, Modulation, Demodulation, and Output.
- The ***code1*** file has 4 parts. The first three part are about the normal Delta Modulation. The differences between them are the sampling time and the step size according to the project's requirements. The last part is about the "Variable Slope Delta Modulation" required in the project's bonus.
- *For Part 1, Part 2, and Part 3, the sequence goes as follows:*
    1. A suitable *Sampling Time* and *Step Size* are chosen fitting the signal period.
    2. The time interval for all signals is generated with a *Start= 0* and *End= 6* (Representing 6ms).
    3. One of the required three signals is generated; a sine wave, a DC voltage, or a square wave.
    4. The generated signal whatever it is gets plotted.
    5. The *Modulation* function found in the folder with the same name gets called.
    6. *(Modulation Function)*: First, we create a variable to hold Delta Modulated Signal's values called *Delta_Mod* along with another variable to hold Output Signal's values, *Out_Sign.* Next, for each sample of the signal, the delta modulated signal value is compared to the original signal value. if Act_Signal(i-1) > Delta_Mod (i-1), meaning the actual signal is increasing, a step is added to the delta modulated signal. On the other hand, if Act_Signal(i-1) < Delta_Mod (i-1), meaning the actual signal is decreasing, , a step is subtracted from the delta modulated signal.
    7. The output delta modulated signal plot, generated by the *Output* function, is added to the original signal plot.
    8. *(Output Function)*: This function is used to generate an accurate plot for the Delta Modulated signal. This is achieved by generating a large number of sample between each two signal samples, adjusting the scale to fit with the signal scale, reading the output signal, then finally drawing a square wave if this value equals to 1.
    9. The Delta Modulated signal is then demodulated using the Demodulation function.
    10. *(Demodulation Function)*: This part is about filtering the delta modulated signal using a low pass filter, to smoothen its edges and output a signal that resembles the original signal.
    11. Finally, the mean square error is calculated using the given formulas.

- *For the Variable Slope Delta Modulation part (Bonus Part), the sequence goes as follows:*
    1. The slope is calculated using this formula (Act_Signal(i) - Act_Signal(i-1))/Ts.
    2. Next, for each sample of the signal, the delta modulated signal value is compared to the original signal value. if Act_Signal(i-1) > Delta_Mod (i-1), meaning the actual signal is increasing, and the slope ranges between 0 and 0.7, a step is added to the delta modulated signal, and if the slope is greater than 0.7, two times the step is added to the delta modulated signal. On the other hand, if Act_Signal(i-1) < Delta_Mod (i-1), meaning the actual signal is decreasing, a step is subtracted from the delta modulated signal.
    3. The signal is then plotted.
- *Some Notes:*
    - The Delta Modulated signal is plotted using the *stairs (t,Delta_Mod)* function.
    - hold on is used to add multiple plots on the same figure.

Report 2 on Experiment 6: Codes Description

By: Mohamed Wagdy Ali Nomeir

This report will describe each code's idea of operation individually; we entered random bits in the code and generated the waveform for different line codes. We also provided the PSD of each line code to understand the required bandwidth of each.

We generated random binary data using “Randi” built-in function and then modulated it for each line code. We generated continuous waveform from these binary data.

NRZ: In this line code we compared random data, if they are equal to one, output will be equal to one volt if not then it is equal to -1 volt.

NRZ-I: In this modulation bit 1 reverses the waveform while zero keeps it the same. It needs a memory with the previous. We done the code by using the remainder given the data is zero or one, if the current bit is one and the sum is divisible by 2 then -1 volt is generated else 1 volt is generated, and vice versa for zero bit.

RZ: is the same as NRZ only the last half of each bit is guarded by a zero volt, we added 2 variables to know the beginning and the half of each duration.

AMI: it generated a zero level for a zero bit, and a one or negative one when one bit is generated. The code sum of the previous data, if the sum is even then a positive one volt is generated else negative one volt is generated.

Manchester coding: This line code, it generates the first half of the bit high and the second half is negative one if one bit is generated, and vice versa for zero. It also has 2 variables to check for the beginning and half of bit duration.

MLT-3: In this line code if bit zero is transmitted keep the previous level, if bit 1 then there are 4 transitions, high to neutral, neutral to low, low to neutral, neutral to high. We divided this code in two parts; the first part generates 5 values, 4 for each transition and one for keeping the previous transition. Then the waveform is generated corresponding to these values.

Experiment 7

By : Marwan Helmy Zaki

Repetition code:

Transmitted sequence is generated using random generation function, i.e. randi(), which generates a row matrix with 10^6 columns. The result is a random sequence of zeros and ones. Bit error rate (Ber) is an empty matrix (zeros) of size 16x3 because there is bit error rate for each SNR value which is 16 values and there are 3 repetition numbers (3,5 & 11). SNR is range of decibels from 0dB up to 30dB with 2dB step. Sent sequence is a repeated version of the generated row sequence. repmat() function is used to repeat the generated row matrix multiple times which result in having repeated rows below each other. To convert this matrix to column matrix we iterate through it using colon operator (:) then taking the transpose to convert the column matrix to row. Output of the receiver is saved in output variable which has the same size as the transmitted sequence. For loop is iterated on the values of the SNR to be added in the noise function. AWGN noise is added to the transmitted sequence with SNR value in each iteration. Threshold voltage is calculated through the function $\frac{\sqrt{\frac{1}{RepNum}}}{2}$. For loop is used to iterate on every single bit in the received sequence to be compared to the threshold voltage. If the value of the bit is above the threshold then 1 was transmitted and if the value is below the threshold then zero was transmitted. The result is iterated on using for loop and we see how many zeros and ones in a repetition range and put the result in a counter for zeros and counter for ones. For example, if the repetition number is 3, then we take every 3 bits in the sequence and see the number of zeros and ones in these 3 bits. If the number of zeros is greater than the number of ones, then the result of the decision device is 0. On the other hand, if the number of ones is greater than the number of zeros, then the result of the decision device is 1. At the end, biterr() function is used to detect the bit rate error ratio. Then SNR is drawn against the bit error rate using semilogy function.

Linear Block Code:

(7,4) linear code is used which means that the length of the data sequence is 4 bits and the length of the codeword resulted is 7 bits. Bit error rate (ber) is initialized to be an empty array to store the result of the bit error rate output. Cyclpoly() function is used as a generator to the polynomial that will be used to generate the parity check matrix. Cyclgen() function is used to generate the parity check matrix, i.e. denoted by H. This

matrix has the form [I | p'] where I is the identity matrix and p' is the transpose of the check parity matrix. Gen2par() function is used to generate the generating matrix G using the matrix H as a parameter. This matrix has the form [p | I]. Syndtable() function is used to generate decoding table for an error correcting binary code with parity check matrix as a parameter. Encode() function is used to encode the sequence as linear block code. Pskmod() function modulates the encoded data using phase shift keying modulation. For loop is used to iterate through each value of the SNR which is a range from 0dB to 30dB with 2dB step. AWGN noise is added to the transmitted sequence. The received data is demodulated using pskdemod() function. Decode() function is used to decode the received data. We then take the first 10^6 bits from the sequence. Biterr() function is used to calculate the bit error rate. The SNR is drawn against bit error rate using semiolgy() function.

Convolutional code:

Number of constrain length is 7 which means that number of shift registers equals 7 stages. Poly2trellis() function is used to generate trellis form of the convolutional code. Convenc() function is used to encode the data bits using convolutional format. Then the sequence is modulated as phase shift keying using pskmod() function. Bit error rate is initialized as an empty array. For loop is used to iterate through each value of the SNR in each iteration. AWGN noise is added to the sequence using awgn() function. The sequence is demodulated using pskdemod(). The received sequence is decoded using vitdec() function. Then the bit rate error is calculated using biterr() function. SNR is drawn against bit error rate using semiolgy() function.