Digital Circuits and Systems Lab

Laboratory report submitted for the partial fulfillment
of the requirements for the degree of

Bachelor of Technology
in
Electronics and Communication Engineering

by

Shrutika Bansal - 16UEC064

Course Coordinator
Dr. Divyang Rawal

Department of Electronics and Communication Engineering
The LNM Institute of Information Technology, Jaipur

August 2017

Contents

Chapter *1*

Theory

BER is a key property of the digital communication system. Various types of modulation methods are used in the digital information transmission system. BER can be demarcated as the number of received bits of a data stream over a communication channel that can be affected due to noise, interference and distortion or bit synchronization errors.
There are different modulation schemes that can be used to modulate the data:-namely BPSK,QPSK,FSK,etc. In space communication power is severely limited. FSK is not generally used as it would require very high bandwidth to modulate our data resulting in very low bandwidth efficiency. Higher constellation QAM also cant be used in our case as that would require very high C/N ratio. The choice is between QPSK and BPSK. The advantage of BPSK is that it requires the lowest C/N ratio. The drawback is that the data rate achieved using BPSK is very low. QPSK is basically two BPSK links operating on the same channel with their carriers in phase quadrature. Therefore the BER of a QPSK remains the same as BPSK. At the same time the data rate is doubled. The only penalty we pay is in terms of C/N ratio. QPSK requires 3 dB more C/N ratio than BPSK. This project demands high date rate without losing much on bandwith and power. Because of these tradeoffs is decided on using QPSK modulation scheme for the raw data received from the rover.

Chapter *2*

Experiment - 1

2.1 AIM

Analyze and Simulate BER performance of BPSK/QPSK signal over AWGN channel.

2.2 Theory

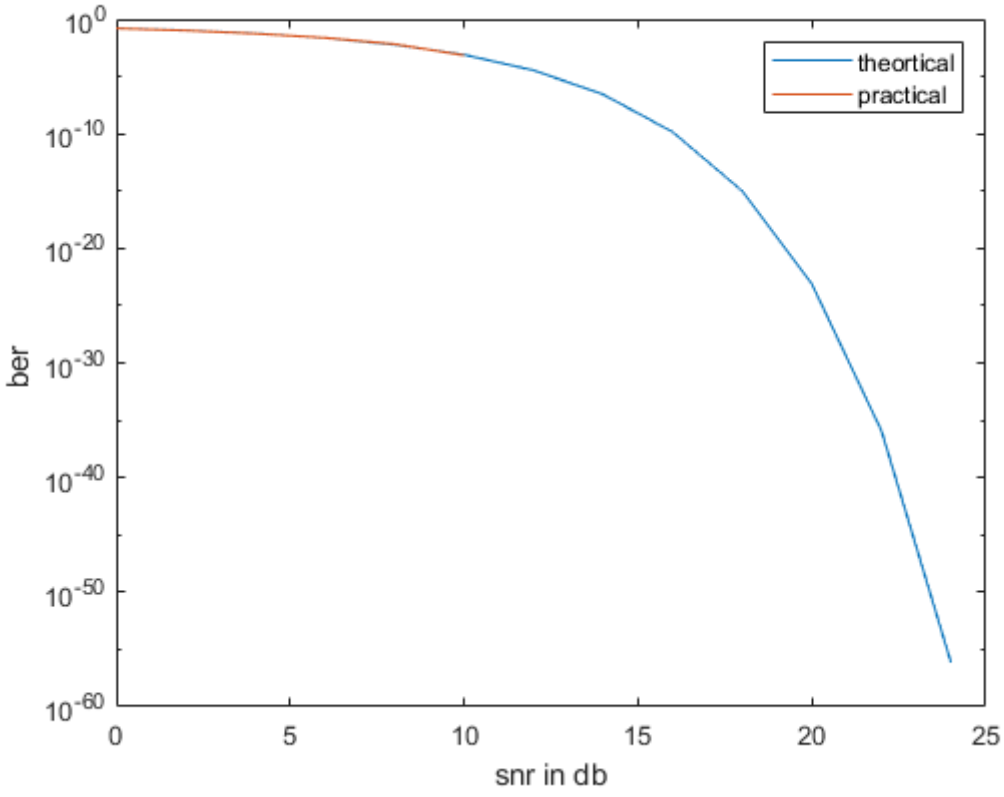
A BPSK modulated signal with power P = Eb is transmitted over (AWGN) Additive White Gaussian Channel is affected by various types of noise, like thermal noise. This noise is additive in nature, has flat spectrum(white - uncorrelated), has gaussian PDF(probability density function).

2.3 code and Results

2.3.1 BER Vs SNR for BSK over AWGN

```
clc;
clear all;
close all;
N=10000;

r = randi([0,1],N,1);

for i=1:N                       %converted to bpsk

        if(r(i)==0)
            r(i) = -1;
        else
             r(i) = 1;
        end


end

n = randn(N,1);                  %noise signal


snr_db = 0:2:24;
kk=1;% snr in db

for k = 1: length(snr_db)

snr_linear = 10.^(snr_db(k)/10);           % converted snr to linear
sigma = 1./(snr_linear).^(1/2);                % find sigma

y = r+ sigma.*n;                  % find y = bpsk_signal + sigma*noise


% convert y sequence into bpsk take threshold value = 0
% z is your constructed signal
% y is output signal with noise
for j=1:N
   if(y(j)<0)
            z(j,1) = -1;
        else
            z(j,1) = 1;
        end

end

% check bit by bit that r and z is same or not

count_error=0;
for jj=1:N
    if(r(jj)~=z(jj))
        count_error=count_error+1;
%     else
%         count_error;
```

```
    end
end

% calculate theortical ber  Q((snr_linear)^(1/2)


ber_th(k) = qfunc((snr_linear).^(1/2));
ber_prac(k) = count_error/N;
%k=k+1;
end
%snr_linear1 = 10.^(snr_db/10)  ;

semilogy(snr_db, ber_th);
xlabel("snr in db");
ylabel("ber ");

hold on
semilogy(snr_db, ber_prac);
xlabel("snr in db");
ylabel("ber ");

legend('theortical','practical')
```

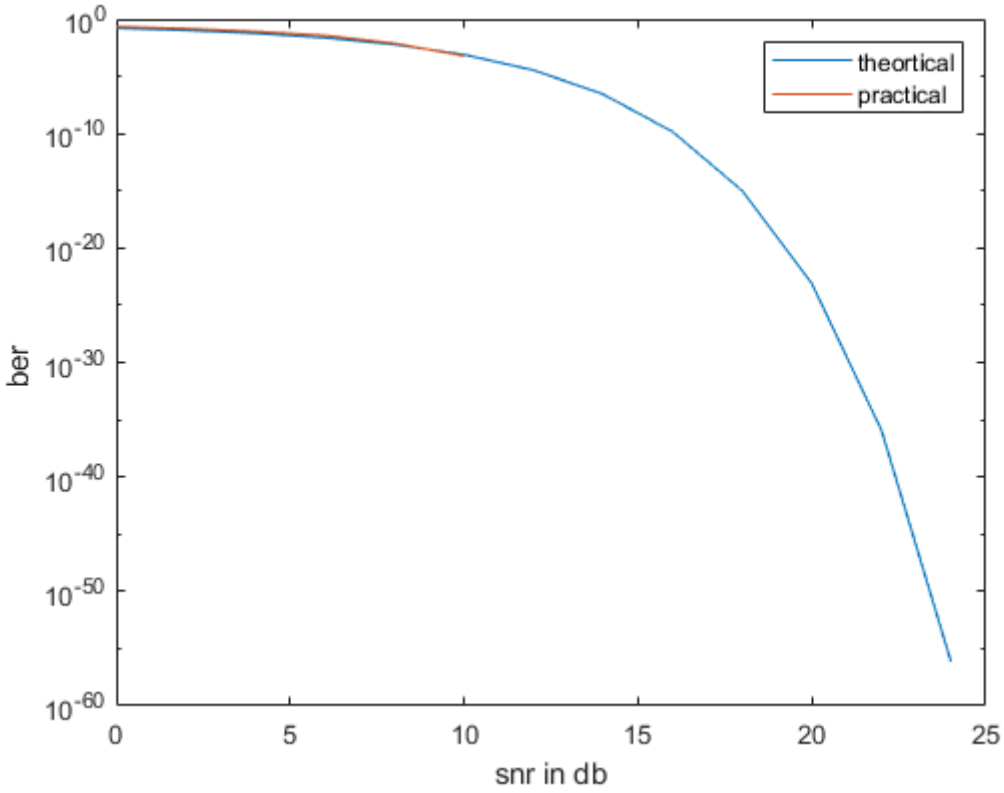


Published with MATLAB® R2018b

2.3.2 BER Vs SNR for QPSK over AWGN

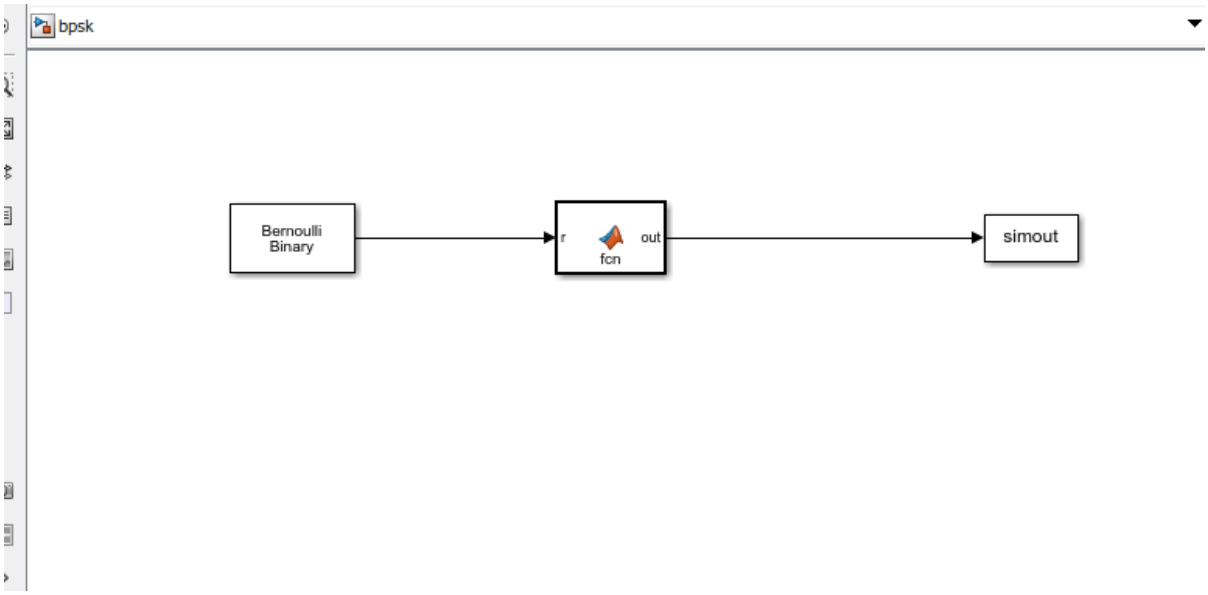
```
clc;
clear all;
close all;

N= 10000;

% r =[0 0 0 1 1 0 1 1];
% N = length(r);
r = randi([0,1],1,N);
k =1;
for i=1:2:N                     %converted to qpsk

        if((r(i)==0)&&(r(i+1)==0))
            r1(k) = 1+1j;

        elseif((r(i)==0)&&(r(i+1)==1))
            r1(k) = -1+1j;

        elseif((r(i)==1)&&(r(i+1)==0))
            r1(k) = -1-1j;

        elseif((r(i)==1)&&(r(i+1)==1))
            r1(k) = 1-1j;

        end
    k = k+1;

end

n = randn(1,N/2)+j*randn(1,N/2);


snr_db = 0:2:24;
kk=1;% snr in db

for k = 1: length(snr_db)

snr_linear = 10.^(snr_db(k)/10);          % converted snr to linear
sigma = 1./(snr_linear).^(1/2);               % find sigma

y = r1+ sigma.*n;                % find y = bpsk_signal + sigma*noise


% convert y sequence into bpsk take threshold value = 0
% z is your constructed signal
% y is output signal with noise
for j=1:N/2
    a=real(y(j));
    b = imag(y(j));
   if((a>=0)&&(b>=0))
            z(2*j-1)=0;
            z(2*j)=0;
```

```
   elseif((a<0)&&(b>=0))
            z(2*j-1)=0;
            z(2*j)=1;

   elseif((a<0)&&(b<0))
            z(2*j-1)=1;
            z(2*j)=0;

   elseif((a>=0)&&(b<0))
            z(2*j-1)=1;
            z(2*j)=1;
   end
end

% check bit by bit that r and z is same or not

count = sum(r ~= z);

% calculate theortical ber  Q((snr_linear)^(1/2)


ber_th(k) = qfunc((snr_linear).^(1/2));
ber_prac(k) = count/N;
%k=k+1;
end
%snr_linear1 = 10.^(snr_db/10)  ;

semilogy(snr_db, ber_th);
% xlabel("snr in db");
% ylabel("ber theortical");

hold on
semilogy(snr_db, ber_prac);
xlabel("snr in db");
ylabel("ber ");


legend('theortical','practical')
```



Published with MATLAB® R2018b

2.3.3 Simulink: BER BPSK-AWGN over Wired Channel



2.3.4 Simulink: BER QPSK-AWGN over Wired Channel

Chapter *3*

Experiment - 2

3.1 AIM

Analyze and Simulate BER performance of BPSK/QPSK signal over Wireless channel.

3.2 Theory

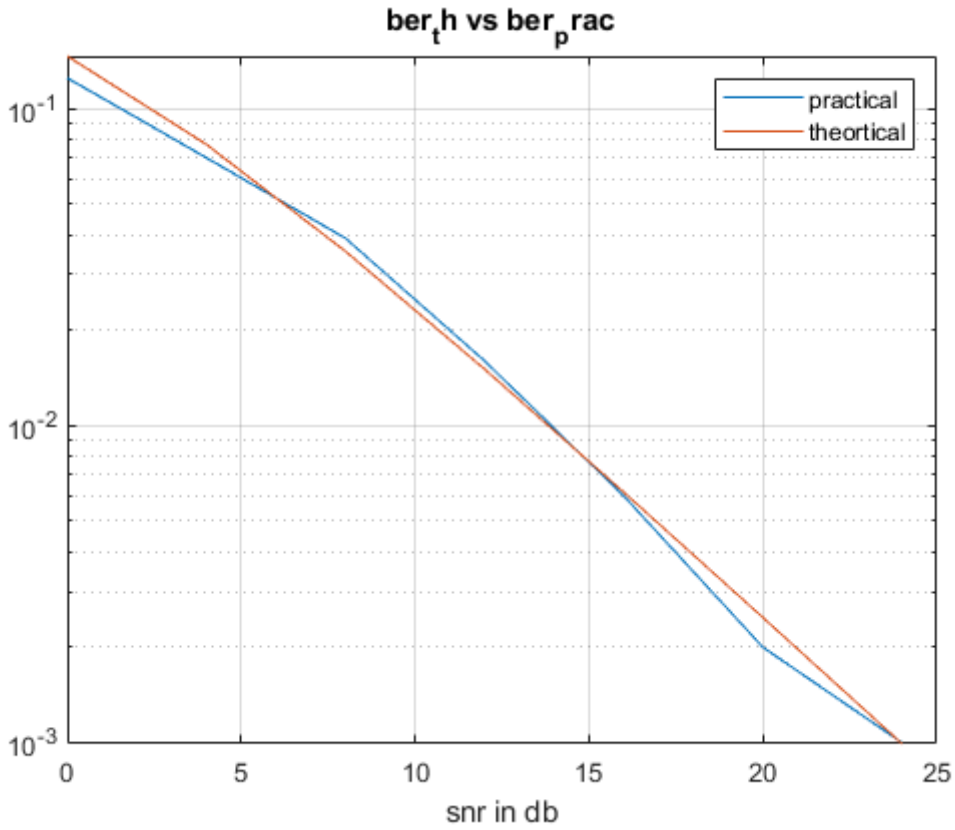
A BPSK/QPSK modulated signal with power P is transmitted over wireless channel accompanied by AWGN noise.

3.3 code and Results

3.3.1 BER Vs SNR for BPSK over Wireless channel

```
clc;
clear all;
close all;

N = 1000;
x = randi([0,1],1,N);

% convert to bpsk
for i = 1:N
    if(x(i)==0)
        t(i) = -1;
    else
        t(i)= x(i);
    end
end

h = (randn(1,N) + 1i*randn(1,N))*sqrt(1/2);  %h complex random ; ray
 light faded channel
n = randn(1,N);                     % noise genrated

snr_db = 0:4:24;
kk =0;



ber_prac = zeros(1, length(snr_db));

for k = 1: length(snr_db)

snr_linear(k) = 10.^(snr_db(k)/10);           % converted snr to
 linear
sigma(k) = 1./(snr_linear(k)).^(1/2);                % find sigma

y = h.*t + sigma(k).*n;                 % find y = channel*bpsk_signal
 + sigma*noise


% convert y sequence into bpsk take threshold value = 0
% z is your constructed signal
% y is output signal with noise
%bpsk wireless communication
z = y./h;


for i = 1:N
    if(z(i)>0)
        r(i)=1;
    else
        r(i)=-1;
    end
end
```

```
ber_th(k) = (1/2)*(1-(snr_linear(k)/(1+snr_linear(k))).^(1/2));

% check bit by bit that r and z is same or not


count_error(k)=0;
for jj=1:N
    if(t(jj)~=r(jj))
        count_error(k)=count_error(k)+1;
%     else
%         count_error;
    end
end
end

ber_prac  = count_error./N;
semilogy(snr_db,ber_prac);
grid on;
hold on;
semilogy(snr_db,ber_th);
title("ber_th vs ber_prac");
xlabel("snr in db");
legend('practical','theortical')
```

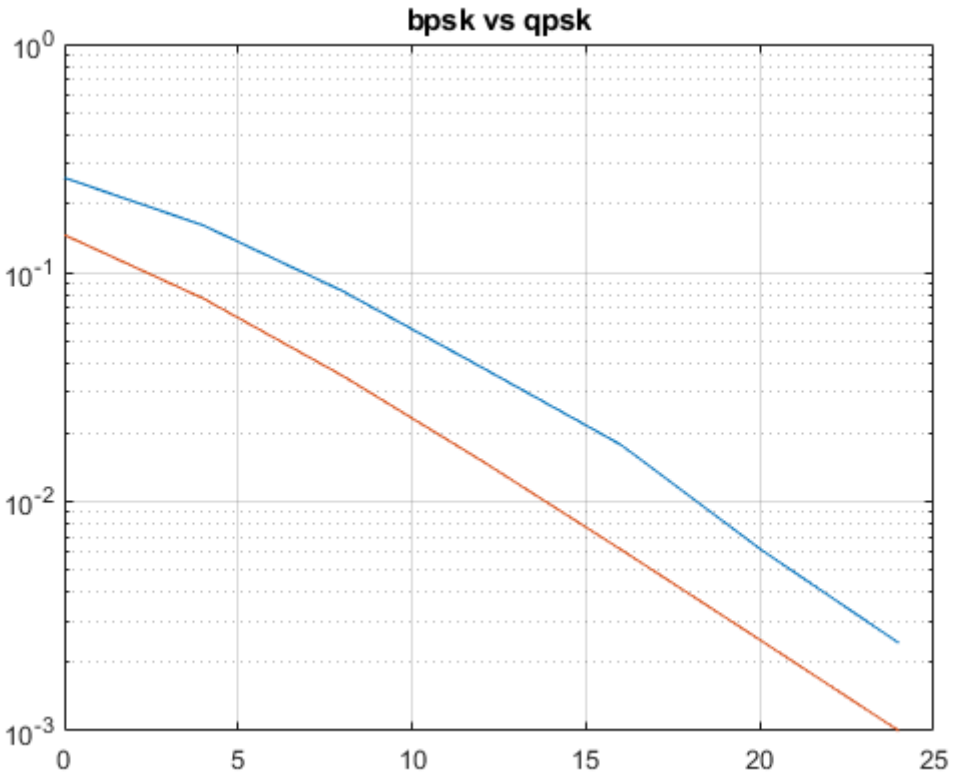


Published with MATLAB® R2018b

3.3.2 BER Vs SNR for QPSK over Wireless Channel

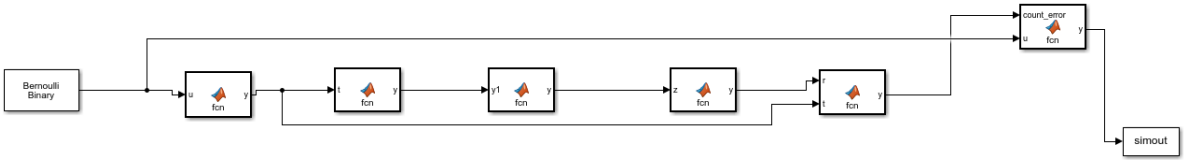
```
clc;
clear all;
close all;

N= 10000;

% r =[0 0 0 1 1 0 1 1];
% N = length(r);
r = randi([0,1],1,N);
k =1;
for i=1:2:N                     %converted to bpsk

        if((r(i)==0)&&(r(i+1)==0))
            r1(k) = 1+1j;

        elseif((r(i)==0)&&(r(i+1)==1))
            r1(k) = -1+1j;

        elseif((r(i)==1)&&(r(i+1)==0))
            r1(k) = -1-1j;

        elseif((r(i)==1)&&(r(i+1)==1))
            r1(k) = 1-1j;

        end
    k = k+1;

end
h = (randn(1,N/2) + 1i*randn(1,N/2))*sqrt(1/2);  %h complex random ;
 ray light faded channel
n = randn(1,N/2)+j*randn(1,N/2);


snr_db = 0:4:24;
kk=1;% snr in db

for k = 1: length(snr_db)

snr_linear(k) = 10.^(snr_db(k)/10);           % converted snr to
 linear
sigma(k) = 1./(snr_linear(k)).^(1/2);                % find sigma

y = h.*r1+ sigma(k).*n;                  % find y = bpsk_signal +
 sigma*noise
y1 = y./h;

% convert y sequence into bpsk take threshold value = 0
% z is your constructed signal
% y1 is output signal with noise
for j=1:N/2
    a=real(y1(j));
    b = imag(y1(j));
```

```
    if((a>=0)&&(b>=0))
             z(2*j-1)=0;
             z(2*j)=0;

    elseif((a<0)&&(b>=0))
             z(2*j-1)=0;
             z(2*j)=1;

    elseif((a<0)&&(b<0))
             z(2*j-1)=1;
             z(2*j)=0;

    elseif((a>=0)&&(b<0))
             z(2*j-1)=1;
             z(2*j)=1;
    end
end

% check bit by bit that r and z is same or not

ber_th(k) = (1/2)*(1-(snr_linear(k)/(1+snr_linear(k))).^(1/2));

% check bit by bit that r and z is same or not

count_error(k)=0;
for jj=1:N
    if(r(jj)~=z(jj))
        count_error(k)=count_error(k)+1;
%     else
%         count_error;
    end
end
end

ber_prac  = count_error./N;
semilogy(snr_db,ber_prac);
grid on;
hold on;
semilogy(snr_db,ber_th);
title("bpsk vs qpsk");
```

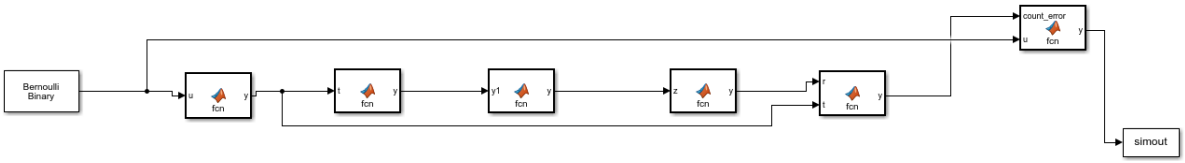


Published with MATLAB® R2018b

3.3.3 Simulink: BER BPSK-AWGN over Wireless Channel



3.3.4 Simulink: BER QPSK-AWGN over Wireless Channel



Chapter *4*

Experiment - 3

4.1 AIM

Analyze and Simulate BER performance of BPSK/QPSK signal over Multiple receiving antenna system.

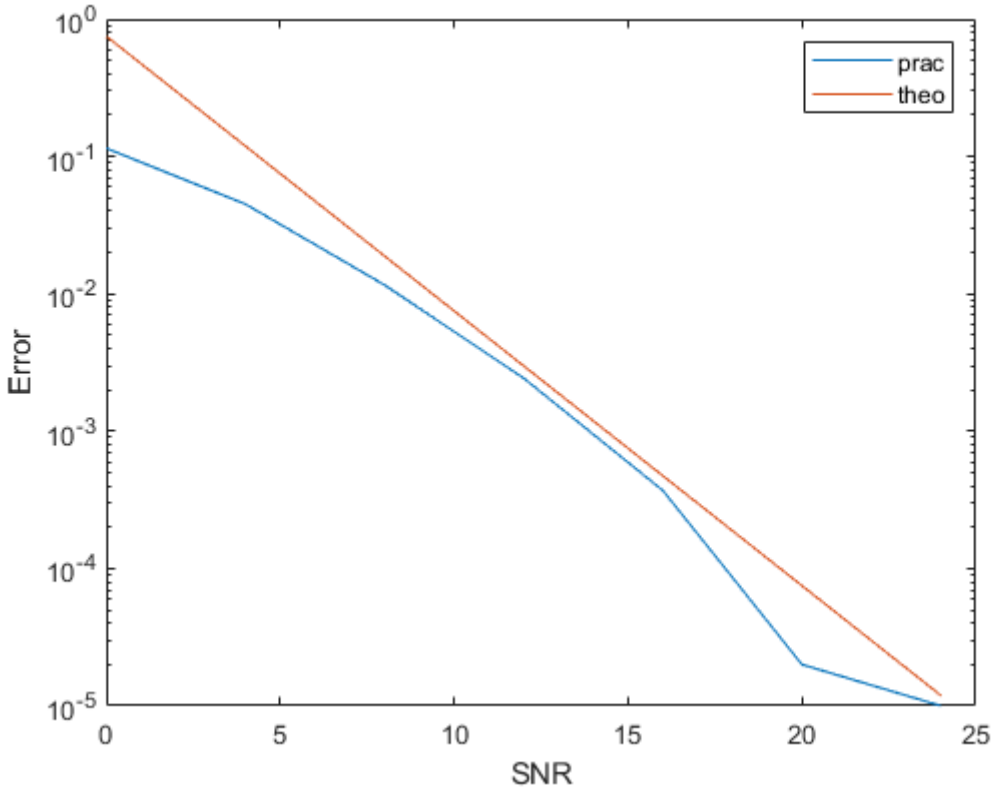
4.2 Theory

The advantage of using multiple receiving antenna is that probability of error in receiving signal decreases

4.3 code and Results

4.3.1 BER Vs SNR for BPSK over L = 2 receiving antenna system

```
clc;
clear all;
close all;

N = 100000;
X = randi([0,1],[1,N]);
L = 2;

for i = 1:N
    if X(i) == 0
        bpsk(i) = -1;
    else
        bpsk(i) = 1;
    end
end

h1 = (1/sqrt(2)).*complex(randn(1,N),randn(1,N));
h2 = (1/sqrt(2)).*complex(randn(1,N),randn(1,N));
n1 = complex(randn(1,N),randn(1,N));
n2 = complex(randn(1,N),randn(1,N));
normsq_h = h1.*conj(h1) + h2.*conj(h2);

SNR_dB = 0:4:24;
l = length(SNR_dB);

for i = 1:l
    SNR_lin(i) = 10^(SNR_dB(i)/10);
    sigma(i) = 1/sqrt(SNR_lin(i));
    y1 = h1.*bpsk + sigma(i)*n1;
    y2 = h2.*bpsk + sigma(i)*n2;
    Dec = ((conj(h1).*y1)+(conj(h2).*y2))./normsq_h;
    for j = 1:N
        if Dec(j) > 0
            X_rcvd(j) = 1;
        else
            X_rcvd(j) = 0;
        end
    end
    e(i) = sum(abs(X-X_rcvd))/N;
    P(i) = nchoosek(2*L-1,L)*(power(1/(2*SNR_lin(i)),L));
end

semilogy(SNR_dB,e);
hold on
semilogy(SNR_dB,P);
xlabel('SNR');ylabel('Error');
%title('SNR vs ERROR');
legend('prac','theo');
```

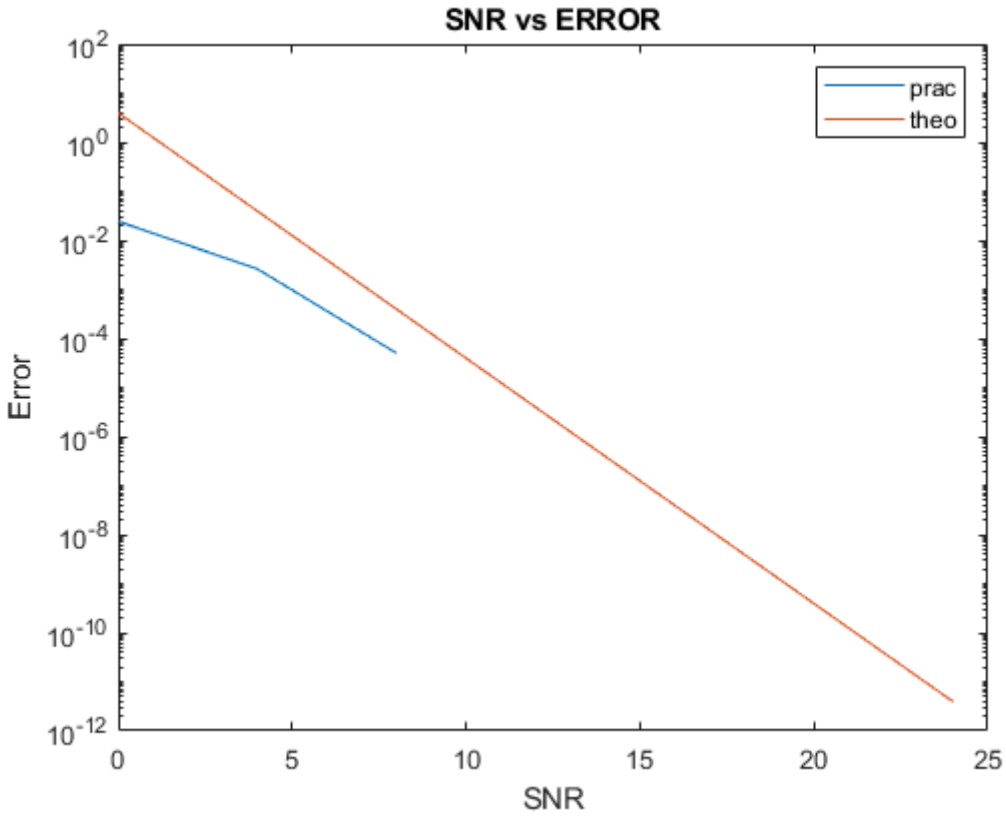


Published with MATLAB® R2018b

4.3.2 generalized code for any number of receiving antenna (BPSK)

```
clc;
clear all;
close all;

N = 100000;
X = randi([0,1],[1,N]);
L = 5;

for i = 1:N
    if X(i) == 0
        bpsk(i) = -1;
    else
        bpsk(i) = 1;
    end
end

for j = 1:L
    h(j,:) = (1/sqrt(2)).*(complex(randn(1,N),randn(1,N)));
    n(j,:) = complex(randn(1,N),randn(1,N));
end
normsq_h = 0;
for j = 1:L
    a = h(j,:).*conj(h(j,:));
    normsq_h = normsq_h + a;
end

SNR_dB = 0:4:24;
l = length(SNR_dB);

for i = 1:l
    SNR_lin(i) = 10^(SNR_dB(i)/10);
    sigma(i) = 1/sqrt(SNR_lin(i));
    for j = 1:L
        y(j,:) = h(j,:).*bpsk + sigma(i)*n(j,:);
    end
    Dec = 0;
    for k = 1:L
        a = conj(h(k,:)).*y(k,:);
        Dec = Dec + a;
    end
    Dec1 = Dec./normsq_h;
    for t = 1:N
        if Dec1(t) > 0
            X_rcvd(t) = 1;
        else
            X_rcvd(t) = 0;
        end
    end
    e(i) = sum(abs(X-X_rcvd))/N;
    P(i) = nchoosek(2*L-1,L)*(power(1/(2*SNR_lin(i)),L));
 end
```

```
semilogy(SNR_dB,e);
hold on
semilogy(SNR_dB,P);
xlabel('SNR');ylabel('Error');
title('SNR vs ERROR');
legend('prac','theo');
```

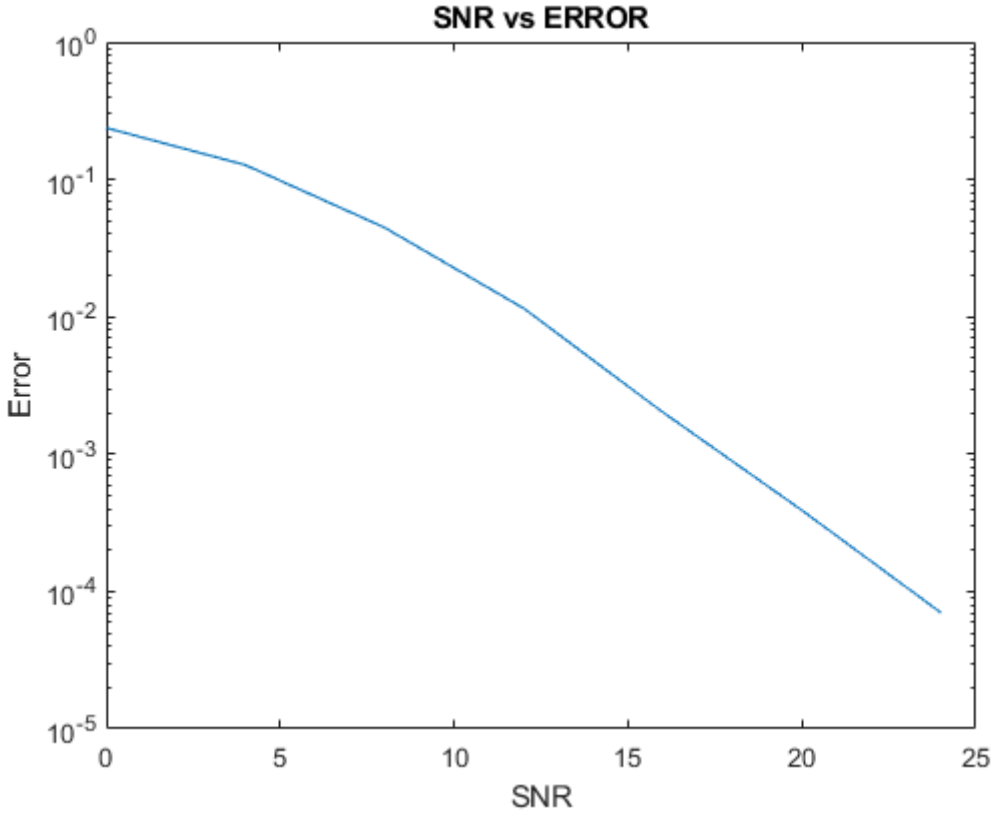


Published with MATLAB® R2018b

4.3.3 BER Vs SNR for QPSK over L = 2 receiving antenna system

```
clc;
clear all;
close all;

N = 100000;
X = randi([0,1],[1,N]);
L = 2;

n = 1;
for i = 1:2:N
    if (X(i) == 0) && (X(i+1) == 0)
        qpsk(n) = (1+1j)/sqrt(2);
    elseif (X(i) == 0) && (X(i+1) == 1)
        qpsk(n) = (-1+1j)/sqrt(2);
    elseif (X(i) == 1) && (X(i+1) == 0)
        qpsk(n) = (-1-1j)/sqrt(2);
    elseif (X(i) == 1) && (X(i+1) == 1)
        qpsk(n) = (1-1j)/sqrt(2);
    end
    n = n+1;
end

h1 = (1/sqrt(2)).*complex(randn(1,N/2),randn(1,N/2));
h2 = (1/sqrt(2)).*complex(randn(1,N/2),randn(1,N/2));
n1 = complex(randn(1,N/2),randn(1,N/2));
n2 = complex(randn(1,N/2),randn(1,N/2));
normsq_h = h1.*conj(h1) + h2.*conj(h2);

SNR_dB = 0:4:24;
l = length(SNR_dB);

for i = 1:l
    SNR_lin(i) = 10^(SNR_dB(i)/10);
    sigma(i) = 1/sqrt(SNR_lin(i));
    y1 = h1.*qpsk + sigma(i)*n1;
    y2 = h2.*qpsk + sigma(i)*n2;
    Dec = ((conj(h1).*y1)+(conj(h2).*y2))./normsq_h;
    n = 1;
    for j = 1:N/2
        if (real(Dec(j))>0) && (imag(Dec(j))>0)
            X_rcvd(n) = 0;
            X_rcvd(n+1) = 0;
        elseif (real(Dec(j))<0) && (imag(Dec(j))>0)
            X_rcvd(n) = 0;
            X_rcvd(n+1) = 1;
        elseif (real(Dec(j))<0) && (imag(Dec(j))<0)
            X_rcvd(n) = 1;
            X_rcvd(n+1) = 0;
        elseif (real(Dec(j))>0) && (imag(Dec(j))<0)
            X_rcvd(n) = 1;
            X_rcvd(n+1) = 1;
        end
```

```
        n = n+2;
    end
    e(i) = sum(abs(X-X_rcvd))/N;
end

semilogy(SNR_dB,e);
xlabel('SNR');ylabel('Error');
title('SNR vs ERROR');
```



Published with MATLAB® R2018b

Chapter *5*

Experiment - 4

5.1 AIM

Simulate M-QAM modulator and Demodulator.

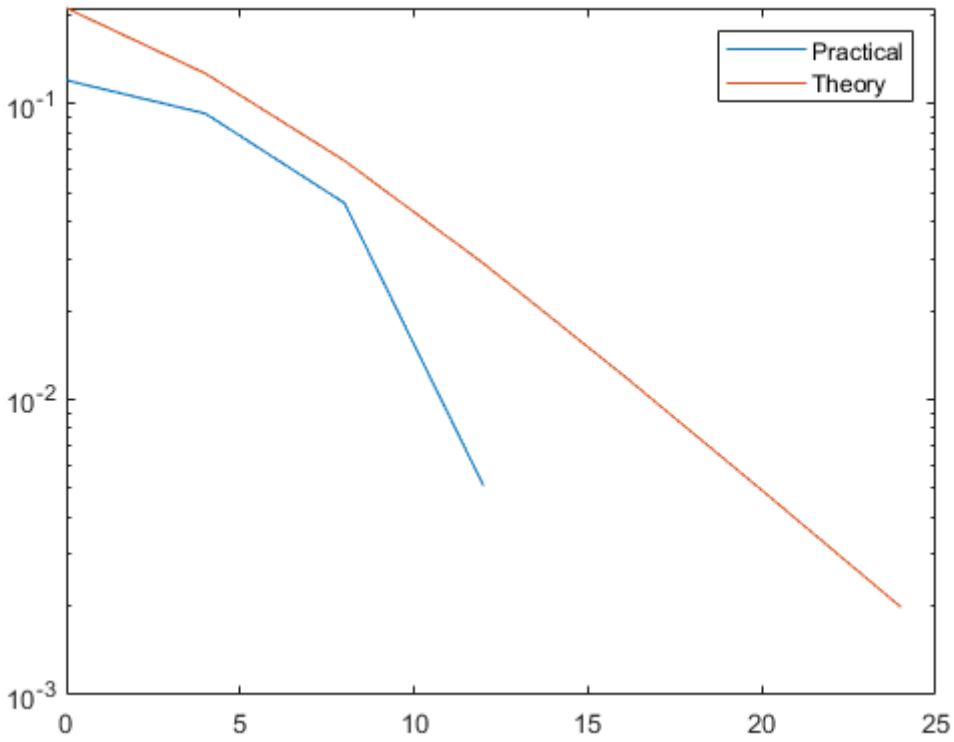
5.2 Theory

Advantage of using M-QAM is that robusnous and probability error in receiving signal decreases. We are implementing MATLAB as well as C++ code.

5.3 code and Results

5.3.1 Generate BPSK modulator, demodulator

```
clc;
clear all;
close all;

x = randi([0,1],1,10000);
bpsk = zeros(1,10000);
bpsk1 = zeros(1,10000);
for n=1:10000
    if(x(n)==0)
        bpsk(n) = -1;
    else
        bpsk(n) = 1;
    end
end

h = randn(1,10000) + 1j*randn(1,10000);
noise = randn(1,10000) + 1j*randn(1,10000);

snrd = zeros(1,7);
a = 0;
for r=1:7
    snrd(r) = a;
    a = a+4;
end

sigma = zeros(1,7);
snrl = zeros(1,7);
perror = zeros(1,7);
terror = zeros(1,7);

for r=1:7
    snrl(r) = 10^(snrd(r)/10);
    sigma(r) = sqrt(10.^(-snrl(r)/10));
    y = h.*bpsk + noise*sigma(r);
    y = y./h;
    for n=1:10000
        if(real(y(n))>0)
            bpsk1(n) = 1;
        else
            bpsk1(n) = 0;
        end
    end
    perror(r) = mean(abs(x-bpsk1));
    %perror(r) = sum(abs((x-bpsk1)/2))/10000;
    terror(r) = 0.5*(1-sqrt(snrl(r)/(2+snrl(r))));
end

figure;
semilogy(snrd,perror);
hold on
semilogy(snrd,terror);
legend('Practical','Theory');
```



Published with MATLAB® R2018b

5.3.2 Generate C + + code for wireless QAM modulator. use zero noise case

```
#include <iostream>
using namespace std;
#include <cmath>
//#include "wired.h"
#include <fstream>
#include"tmwtypes.h"
#include"stdlib.h"
/* run this program using the console pauser or add your own getch, system("pause") or input
loop */

int main() {

        int i,j,k;
        int n =4;
        int lenout=4*n;
        int index;
        int leninp1 = 4*n;
        creal_T h[leninp1];
        creal_T x[leninp1];
        creal_T  y[leninp1/4];
        creal_T t[leninp1];

        cout << " values of x: ";
         for(i = 0; i < leninp1; i++)
  {

        h[i].re=(rand()%2);
        h[i].im=(rand()%2);
        x[i].re=(rand()%2);
        //x[i].im=2*(rand()%2)-1;


        cout <<x[i].re << " ";


  }

        cout << "\n "<< "\n" ;

        for(i = 0; i < leninp1/4; i++)
        {
                if(x[4*i].re==0&&x[4*i+1].re==0&&x[4*i+2].re==0&&x[4*i+3].re==1)
```

```
        y[i].re=1;
        y[i].im = 3;

if(x[4*i].re==0&&x[4*i+1].re==1&&x[4*i+2].re==0&&x[4*i+3].re==1)
        y[i].re=3;
        y[i].im = 3;

if(x[4*i].re==0&&x[4*i+1].re==1&&x[4*i+2].re==0&&x[4*i+3].re==0)
        y[i].re=3;
        y[i].im = 1;

if(x[4*i].re==0&&x[4*i+1].re==1&&x[4*i+2].re==1&&x[4*i+3].re==0)
        y[i].re=3;
        y[i].im = -1;

if(x[4*i].re==0&&x[4*i+1].re==1&&x[4*i+2].re==1&&x[4*i+3].re==1)
        y[i].re=3;
        y[i].im = 3;

if(x[4*i].re==0&&x[4*i+1].re==0&&x[4*i+2].re==0&&x[4*i+3].re==0)
        y[i].re=1;
        y[i].im = 1;

if(x[4*i].re==0&&x[4*i+1].re==0&&x[4*i+2].re==1&&x[4*i+3].re==1)
        y[i].re=1;
        y[i].im = -1;
```

```
if(x[4*i].re==0&&x[4*i+1].re==0&&x[4*i+2].re==1&&x[4*i+3].re==1)
        y[i].re=1;
        y[i].im = 3;


if(x[4*i].re==1&&x[4*i+1].re==0&&x[4*i+2].re==0&&x[4*i+3].re==1)
        y[i].re=-1;
        y[i].im = 3;


if(x[4*i].re==1&&x[4*i+1].re==0&&x[4*i+2].re==0&&x[4*i+3].re==0)
        y[i].re=-1;
        y[i].im = 1;

if(x[4*i].re==1&&x[4*i+1].re==0&&x[4*i+2].re==1&&x[4*i+3].re==0)
        y[i].re=-1;
        y[i].im = -1;


if(x[4*i].re==1&&x[4*i+1].re==0&&x[4*i+2].re==1&&x[4*i+3].re==1)
        y[i].re=-1;
        y[i].im = 3;

if(x[4*i].re==1&&x[4*i+1].re==1&&x[4*i+2].re==0&&x[4*i+3].re==1)
        y[i].re=-3;
        y[i].im = 3;


if(x[4*i].re==1&&x[4*i+1].re==1&&x[4*i+2].re==0&&x[4*i+3].re==0)
        y[i].re=-3;
        y[i].im = 1;
```

```
            if(x[4*i].re==1&&x[4*i+1].re==1&&x[4*i+2].re==1&&x[4*i+3].re==0)
                y[i].re=3;
                y[i].im = 3;



            if(x[4*i].re==1&&x[4*i+1].re==1&&x[4*i+2].re==1&&x[4*i+3].re==1)
                y[i].re=-3;
                y[i].im = 3;



    }
    cout<< "values of y: ";

            for(i = 0; i < leninp1/4; i++)
                    cout << y[i].re<< " "<< y[i].im << "\n" ;




    return 0;
}
```

Chapter *6*

Experiment - 5

6.1 AIM

Gaussian noise Generation (Central limit theorem) and performance evaluation of various systems.

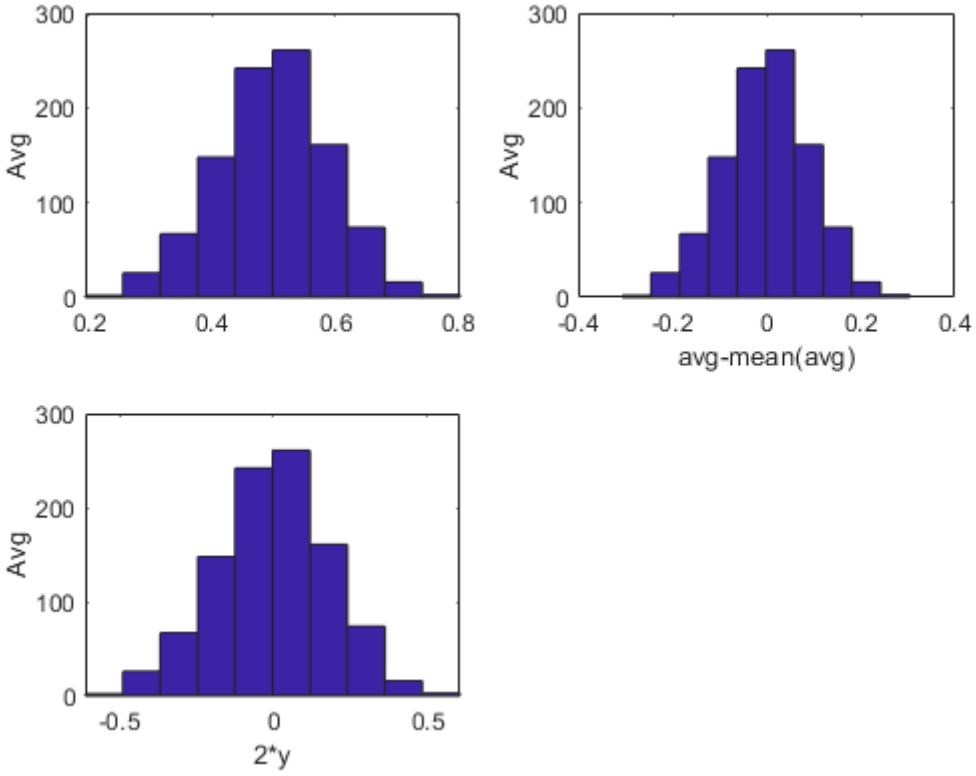
6.2 Theory

Gaussian noise is generated in MATLAB using central limit theorem and in C++ too.

6.3 code and Results

6.3.1 Generate Gaussian noise using central limit theorem in matlab

```
clc;
clear all;
 close all;


 sum = 0;
 n = 1000;
 a = rand(1,n);
  b = rand(1,n);
   c = rand(1,n);
    d = rand(1,n);
     e = rand(1,n);
      f = rand(1,n);
       g = rand(1,n);
        h = rand(1,n);
         x = rand(1,n);
          y = rand(1,n);

sum=a+b+c+d+e+f+g+h+x+y;

avg = sum/10;
subplot(221);
hist(avg);
%xlabel('SNR');
ylabel('Avg');


y = avg-mean(avg);
subplot(222)
hist(y);
xlabel('avg-mean(avg)');ylabel('Avg');


subplot(223)
y1=2*y;
hist(y1);
xlabel('2*y');ylabel('Avg');
```



Published with MATLAB® R2018b

6.3.2 Generate Gaussian noise in DevC++

```
#include <iostream>
using namespace std;
#include <cmath>
#include <fstream>
#include"tmwtypes.h"
#include"stdlib.h"

int main() {
	int i, j;
	int n = 10, N = 5, max = 32767;
	double x[5][1000], y[1000] = {0}, z[1000];
	ofstream fout;
	fout.open("noise.txt");
	for (j = 0; j<N; j++)
	{
		srand(j);
		for (i = 0;i<n;i++)
		{
			x[j][i] = (double)rand()/max;
			cout <<x[j][i]<<endl;
		}
		cout<<"xxxxxx"<<endl;
	}
	cout<<"------"<<endl;
	for (i = 0; i<n; i++)
	{
		for (j = 0; j<N; j++)
		{
			y[i] = y[i]+x[j][i];
		}
		cout<<y[i]<<endl;
	}
	cout<<"======="<<endl;
	for (j = 0; j<n; j++)
	{
		z[j] = y[j]/N;
		cout << z[j] <<endl;
		fout << z[j] <<"\t";
	}
	fout.close();
	return 0;
}
```

Chapter *7*

Experiment - 6

7.1 AIM

Image transmission over Wireless channel using Zedboard.

7.2 Theory

Image is transmitted using c++ code in QAM transmission after that image is transmitted by zedboard using PC's.

7.3 code and Results

7.3.1 C++ code to read the Cameraman1.bmp file, store it in out.bmp.

```
#include <bits/stdc++.h>
#include<iostream>
#include<string>
#include <cstdio>
#include <fstream>
#include"tmwtypes.h"
#include"stdlib.h"
using namespace std;
int x[1024*8];
int incr=0;
int z[1024];
 int decToBinary(int n)
{
    // array to store binary number
    int binaryNum[1000];
            for(int i=0; i<8; i++)
            binaryNum[i]=0;
    // counter for binary array
    int i = 0;
    while (n > 0) {

        // storing remainder in binary array
        binaryNum[i] = n % 2;
        n = n / 2;
        i++;
    }

    // printing binary array in reverse order
    for (int j = 7; j >= 0; j--)
        x[incr++]=binaryNum[j];
}


  int binaryToDecimal(long int n)
{
    long int num = n;
    long int dec_value = 0;

    // Initializing base value to 1, i.e 2^0
    int base = 1;

    long int temp = num;
    while (temp)
```

```
    {
        long int last_digit = temp % 10;
        temp = temp/10;

        dec_value += last_digit*base;

        base = base*2;
    }

    return dec_value;
}




int main()
{
    FILE* file1;
    FILE* out;
    //FILE *shrutika;

    char buffer[1024];
    int buffer1[1024];
    file1 = fopen("cameraman1.bmp","r");
    int count=0;
    while (!feof(file1))
    {
        fread(buffer, sizeof(buffer), 1, file1);
    }
    for(int i=0; i<1024; i++)
    {
            buffer1[i]=buffer[i]+128;
            decToBinary(buffer1[i]);

            }
```

```
int n=1024*8;;
int norm;
creal_T h[n];
creal_T y[n];
creal_T conj[n];
creal_T det[n];
creal_T xnew[n/2];
int final[n];
int j=0;
for(int i=0; i<n;)
{
        if(x[i]==0&&x[i+1]==0)
                {
                        xnew[j].re=1;
                        xnew[j].im=1;
                }
        else if(x[i]==0&&x[i+1]==1)
                {
                        xnew[j].re=-1;
                        xnew[j].im=-1;
                }
        else if(x[i]==1&&x[i+1]==0)
                {
                        xnew[j].re=-1;
                        xnew[j].im=1;
                }
        else if(x[i]==1&&x[i+1]==1)
                {
                        xnew[j].re=1;
                        xnew[j].im=-1;
                }
        i=i+2;
        j=j+1;
}
for(int i=0; i<n/2;i++)
{
        h[i].re=rand();
        h[i].im=rand();
}


for(int i=0;i<n/2;i++)
{
```

```
        y[i].re=xnew[i].re*h[i].re-(xnew[i].im*h[i].im);
        y[i].im=xnew[i].im*h[i].re+(h[i].im*xnew[i].re);
        h[i].im=-h[i].im;
        det[i].re=(y[i].re*h[i].re)-(y[i].im*h[i].im);
        det[i].im=(y[i].re*h[i].im)+(h[i].re*y[i].im);
        norm=h[i].re*h[i].re+(h[i].im*h[i].im);
        det[i].re=det[i].re/norm;
        det[i].im=det[i].im/norm;
        }


        j=0;
        for(int i=0; i<n/2; i++)
        {
                if(det[i].re==1&&det[i].im==1)
                {
                        final[j++]=0;
                        final[j++]=0;
                }
                else if(det[i].re==1&&det[i].im==-1)
                {
                        final[j++]=1;
                        final[j++]=1;
                }
                else if(det[i].re==-1&&det[i].im==1)
                {
                        final[j++]=1;
                        final[j++]=0;
                }
                else if(det[i].re==-1&&det[i].im==-1)
                {
                        final[j++]=0;
                        final[j++]=1;
                }
        }
//      for(int i=0; i<n; i++)
//      cout<<x[i];
//      cout<<"\n";
        //for(int i=0; i<n; i++)
//      cout<<final[i];

        int reconst[1024];
        int z=0;
```

```
	long int shruti=0;
	for(int i=0; i<1024*8; )
	{
		for(int k=0; k<8; k++)
		{
			int p=final[i+k];
			shruti=p*pow(10,7-k)+shruti;

		}
			int rep=binaryToDecimal(shruti);
			reconst[z++]=rep;
		shruti=0;
		i=i+8;
	}

	char reconstim[1024];
	for(int i=0; i<1024;i++)
	{
		reconstim[i]=char(reconst[i]-128);
		cout<<(reconst[i])<<" "<<(buffer1[i])<<"\n";
	}




	out = fopen("out.bmp","wb");

	for(int i=0; i<1024; i++)
	{
		fputc((reconstim[i]),out);

	}
//	shrutika = fopen("out.txt","wb");

  return 0;
}
```

Chapter *8*

Experiment - 7(Pre Lab)

8.1 AIM

Design/Simulate Fixed point Detector for single link Wireless system.

8.2 Theory

Here we used fixed point so that we can convert all our floating data into integer.

8.3 code and Results

8.3.1 fixed point simulation

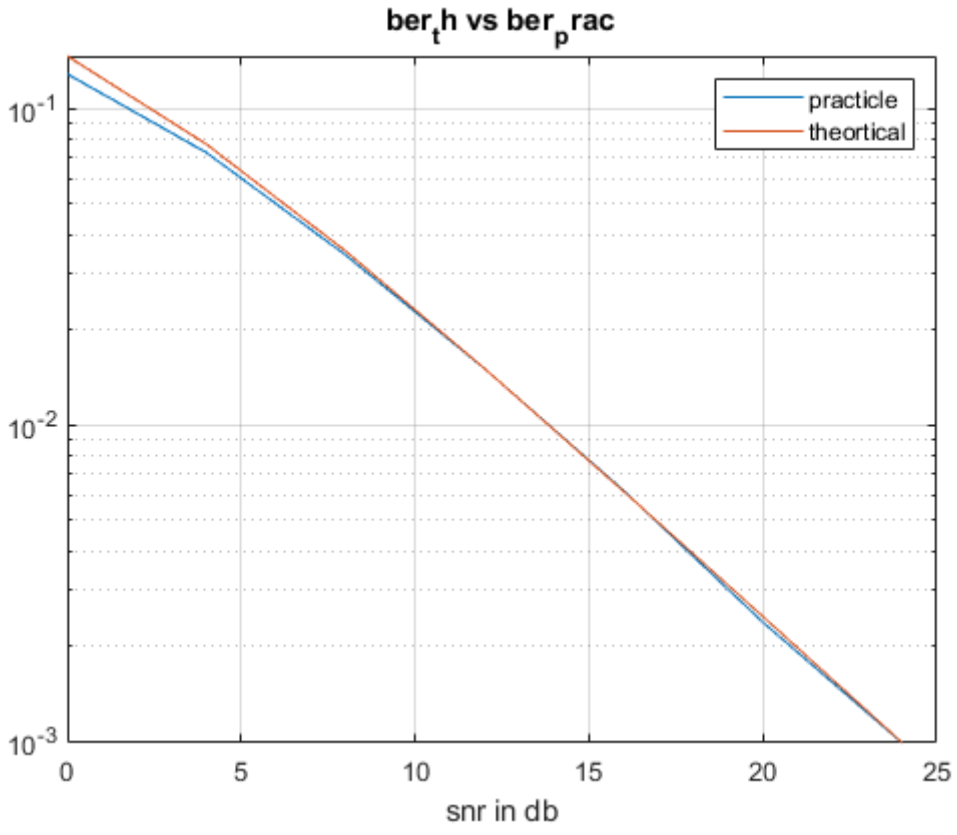
```
clc;
clear all;
close all;

x=1;
h = round(0.3*(2^8));
n = round(0.01*0.2*(2^8));
y = (h*x+n)*(2^8);
dec = ceil((y/h)*(2^8));
if((dec>0))
    dec=1;
else
    dec=0;

end

error = dec-x


error =

     0
```

Published with MATLAB® R2018b

8.3.2 BPSK fixed point 16 bit

```
clc;
clear all;
close all;

N = 100000;
x = randi([0,1],1,N);

% convert to bpsk
for i = 1:N
    if(x(i)==0)
        t(i) = -1;
    else
        t(i)= x(i);
    end
end

h = round(((randn(1,N) + 1i*randn(1,N))*sqrt(1/2))*(2^16));  %h
 complex random ; ray light faded channel
n = round((randn(1,N))*(2^16));                    % noise genrated

snr_db = 0:4:24;
kk =0;



ber_prac = zeros(1, length(snr_db));

for k = 1: length(snr_db)

snr_linear(k) = 10.^(snr_db(k)/10);           % converted snr to
 linear
sigma(k) = 1./(snr_linear(k)).^(1/2);                % find sigma

y = round((h.*t + sigma(k).*n)*(2^16));                 % find y =
 channel*bpsk_signal + sigma*noise


% convert y sequence into bpsk take threshold value = 0
% z is your constructed signal
% y is output signal with noise
%bpsk wireless communication
z = ceil((y./h)*(2^16));


for i = 1:N
    if(z(i)>0)
        r(i)=1;
    else
        r(i)=-1;
    end
end
```

```
ber_th(k) = (1/2)*(1-(snr_linear(k)/(1+snr_linear(k))).^(1/2));

% check bit by bit that r and z is same or not


count_error(k)=0;
for jj=1:N
    if(t(jj)~=r(jj))
        count_error(k)=count_error(k)+1;
%     else
%         count_error;
    end
end
end

ber_prac  = count_error./N;
semilogy(snr_db,ber_prac);
grid on;
hold on;
semilogy(snr_db,ber_th);
title("ber_th vs ber_prac");
xlabel("snr in db");


legend('practicle','theortical');
```



Published with MATLAB® R2018b

Chapter *9*

Conclusion

Through this lab we learn signal transmission in a perfect sequence. Firstly we learn how to write a MATLAB code of BPSK and QPSK in wired channel and compare the snr values.
After that we learn how to write a MATLAB code of BPSK and QPSK in wireless channel and compare the snr values.
After wireless system we try to minimize receiving error using multiple antennas.
We aded noise and modulate and demodulate signal in C code and MATLAB code and transmitted our c code on zed board.
Through this lab we are able to understand how transmission technique works in real life. We are able to implement modulation, demodulation and image transmission using MATLAB and C code through BPSK, QPSk and M-QAM.
We are also able to merge noisse and multiple antenna in one project as well as image transmission too.