CODE:

function LEACH()

n=200;

%Initial Energy

Eo=0.1;

xm=100;

ym=100;

%x and y Coordinates of the Sink

sink.x=1.5\*xm;

sink.y=0.5\*ym;

%Optimal Election Probability of a node

%to become cluster head

p=0.2;

%Eelec=Etx=Erx

ETX=50\*0.000000001;

ERX=50\*0.000000001;

%Transmit Amplifier types

Efs=10\*0.000000000001;

Emp=0.0013\*0.000000000001;

%Data Aggregation Energy

EDA=5\*0.000000001;

%Values for Hetereogeneity

%Percentage of nodes than are advanced

m=0.5;

%\alpha

a=1;

%maximum number of rounds

%rmax=input('enter the number of iterations you want to run : ');

rmax=50;

%%%%%%%%%%%%%%%%%%%%%%%%% END OF PARAMETERS %%%%%%%%%%%%%%%%%%%%%%%%

%Computation of do

do=sqrt(Efs/Emp);

%Creation of the random Sensor Network

figure,

hold off;

for i=1:1:n

S(i).xd=rand(1,1)\*xm;

XR(i)=S(i).xd;

S(i).yd=rand(1,1)\*ym;

YR(i)=S(i).yd;

S(i).G=0;

%initially there are no cluster heads only nodes

S(i).type='N';

temp\_rnd0=i;

%Random Election of Normal Nodes

if (temp\_rnd0>=m\*n+1)

S(i).E=Eo;

S(i).ENERGY=0;

plot(S(i).xd,S(i).yd,'o-r');

hold on;

end

%Random Election of Advanced Nodes

if (temp\_rnd0<m\*n+1)

S(i).E=Eo\*(1+a);

S(i).ENERGY=1;

plot(S(i).xd,S(i).yd,'+');

hold on;

end

end

S(n+1).xd=sink.x;

S(n+1).yd=sink.y;

plot(S(n+1).xd,S(n+1).yd,'o', 'MarkerSize', 12, 'MarkerFaceColor', 'r');

figure(1);

% figure(1)

% plot(o1,o2,'^','LineWidth',1, 'MarkerEdgeColor','k', 'MarkerFaceColor','y', 'MarkerSize',12);

% hold on

%First Iteration

%counter for CHs

countCHs=0;

%counter for CHs per round

rcountCHs=0;

cluster=1;

countCHs;

rcountCHs=rcountCHs+countCHs;

flag\_first\_dead=0;

for r=0:1:rmax

r;

%Operation for epoch

if(mod(r, round(1/p) )==0)

for i=1:1:n

S(i).G=0;

S(i).cl=0;

end

end

hold off;

%Number of dead nodes

dead=0;

%Number of dead Advanced Nodes

dead\_a=0;

%Number of dead Normal Nodes

dead\_n=0;

%counter for bit transmitted to Bases Station and to Cluster Heads

packets\_TO\_BS=0;

packets\_TO\_CH=0;

%counter for bit transmitted to Bases Station and to Cluster Heads

%per round

PACKETS\_TO\_CH(r+1)=0;

PACKETS\_TO\_BS(r+1)=0;

figure(1);

for i=1:1:n

%checking if there is a dead node

if (S(i).E<=0)

plot(S(i).xd,S(i).yd,'^','LineWidth',1, 'MarkerEdgeColor','k', 'MarkerFaceColor','y', 'MarkerSize',8);

dead=dead+1;

if(S(i).ENERGY==1)

dead\_a=dead\_a+1;

end

if(S(i).ENERGY==0)

dead\_n=dead\_n+1;

end

hold on;

end

if S(i).E>0

S(i).type='N';

if (S(i).ENERGY==0)

plot(S(i).xd,S(i).yd,'o','LineWidth',1, 'MarkerEdgeColor','k', 'MarkerFaceColor','g', 'MarkerSize',8);

end

if (S(i).ENERGY==1)

plot(S(i).xd,S(i).yd,'+','LineWidth',3, 'MarkerEdgeColor','k', 'MarkerFaceColor','r', 'MarkerSize',8);

end

hold on;

end

end

plot(S(n+1).xd,S(n+1).yd,'x','LineWidth',1, 'MarkerEdgeColor','k', 'MarkerFaceColor','r', 'MarkerSize',8);

STATISTICS(r+1).DEAD=dead;

DEAD(r+1)=dead;

DEAD\_N(r+1)=dead\_n;

DEAD\_A(r+1)=dead\_a;

% plot(S(n+1).xd,S(n+1).yd,'o', 'MarkerSize', 12, 'MarkerFaceColor', 'r');

% plot(S(n+1).xd,S(n+1).yd,'x','LineWidth',1, 'MarkerEdgeColor','k', 'MarkerFaceColor','r', 'MarkerSize',8);

%When the first node dies

if (dead==1)

if(flag\_first\_dead==0)

first\_dead=r;

flag\_first\_dead=1;

end

end

countCHs=0;

cluster=1;

for i=1:1:n

if(S(i).E>0)

temp\_rand=rand;

if ( (S(i).G)<=0)

%Election of Cluster Heads

if(temp\_rand<= (p/(1-p\*mod(r,round(1/p)))))

countCHs=countCHs+1;

packets\_TO\_BS=packets\_TO\_BS+1;

PACKETS\_TO\_BS(r+1)=packets\_TO\_BS;

S(i).type='C';

S(i).G=round(1/p)-1;

C(cluster).xd=S(i).xd;

C(cluster).yd=S(i).yd;

plot(S(i).xd,S(i).yd,'k\*');

distance=sqrt( (S(i).xd-(S(n+1).xd) )^2 + (S(i).yd-(S(n+1).yd) )^2 );

C(cluster).distance=distance;

C(cluster).id=i;

X(cluster)=S(i).xd;

Y(cluster)=S(i).yd;

cluster=cluster+1;

%Calculation of Energy dissipated

distance;

if (distance>do)

S(i).E=S(i).E- ( (ETX+EDA)\*(4000) + Emp\*4000\*( distance\*distance\*distance\*distance ));

%S(i).E=S(i).E- ( (ETX+EDA)\*(4000) + Emp\*4000\*( distance\*distance\*distance\*distance ));

end

if (distance<=do)

S(i).E=S(i).E- ( (ETX+EDA)\*(4000) + Efs\*4000\*( distance \* distance ));

%S(i).E=S(i).E- ( (ETX+EDA)\*(4000) + Efs\*4000\*( distance \* distance ));

end

Energy\_disp(r+1) = S(i).E;

end

end

end

end

STATISTICS(r+1).CLUSTERHEADS=cluster-1;

CLUSTERHS(r+1)=cluster-1;

%Election of Associated Cluster Head for Normal Nodes

for i=1:1:n

if ( S(i).type=='N' && S(i).E>0 )

if(cluster-1>=1)

min\_dis=sqrt( (S(i).xd-S(n+1).xd)^2 + (S(i).yd-S(n+1).yd)^2 );

min\_dis\_cluster=1;

for c=1:1:cluster-1

temp=min(min\_dis,sqrt( (S(i).xd-C(c).xd)^2 + (S(i).yd-C(c).yd)^2 ) );

if ( temp<min\_dis )

min\_dis=temp;

min\_dis\_cluster=c;

end

end

%Energy dissipated by associated Cluster Head

min\_dis;

if (min\_dis>do)

S(i).E=S(i).E- ( ETX\*(4000) + Emp\*4000\*( min\_dis \* min\_dis \* min\_dis \* min\_dis));

end

if (min\_dis<=do)

S(i).E=S(i).E- ( ETX\*(4000) + Efs\*4000\*( min\_dis \* min\_dis));

end

%Energy dissipated

if(min\_dis>0)

distance=sqrt( (S(C(min\_dis\_cluster).id).xd-(S(n+1).xd) )^2 + (S(C(min\_dis\_cluster).id).yd-(S(n+1).yd) )^2 );

S(C(min\_dis\_cluster).id).E = S(C(min\_dis\_cluster).id).E- ( (ERX + EDA)\*4000 );

if (distance>do)

S(C(min\_dis\_cluster).id).E=S(C(min\_dis\_cluster).id).E- ( (ETX+EDA)\*(4000) + Emp\*4000\*( distance\*distance\*distance\*distance ));

end

if (distance<=do)

S(C(min\_dis\_cluster).id).E=S(C(min\_dis\_cluster).id).E- ( (ETX+EDA)\*(4000) + Efs\*4000\*( distance \* distance ));

end

PACKETS\_TO\_CH(r+1)=n-dead-cluster+1;

end

S(i).min\_dis=min\_dis;

S(i).min\_dis\_cluster=min\_dis\_cluster;

end

end

end

hold on;

countCHs;

rcountCHs=rcountCHs+countCHs;

sum=0;

for i=1:1:n

if(S(i).E>0)

sum=sum+S(i).E;

end

end

avg=sum/n;

STATISTICS(r+1).AVG=avg;

sum;

%Code for Voronoi Cells

%Unfortynately if there is a small

%number of cells, Matlab's voronoi

%procedure has some problems

warning('OFF');

[vx,vy]=voronoi(X(:),Y(:));

plot(X,Y,'g+',vx,vy,'m-');

hold on;

voronoi(X,Y);

axis([10 xm 0 ym]);

end

msgbox('Leach Simulation completted....');

end

Code 2:

tic

clc;

clear all;

close all;

global refDevices blindDevices totalDevices linearRefLocs dhat funcEvals dfuncEvals;

% Basic simulation parameters

roomSize = [100,100]; % Room size, meters

gridSize = 5; % How many sensors per side

refDevices = 10; % How many references (must be same length as actualRefLocs)

trials = 20; % How many indep trials to run

measMethod = 'R';

totalDevices = 31;

blindDevices = totalDevices - refDevices;

blindCoords = 2\*blindDevices;

actualRefLocs = [0,0; 0,25;0,50;0,75;0,100; 100,0; 100,25;100,50;100,75;100,100];

linearRefLocs = [actualRefLocs(:,1)', actualRefLocs(:,2)'];

func = 'calcError';

dfunc = 'calcDError';

% Optimization parameters

ftol = 0.00001;

%| 1. Set up the blindfolded device locations

delta = 1/(gridSize-1);

coords = 0:delta:1;

xMatrix = ones(gridSize,1)\*coords;

yMatrix = xMatrix';

xBlind=rand(1,21)\*100;

yBlind=rand(1,21)\*100;

%%%%%%%%%%%%%%%%%%%%%%%

prompt = {'Enter Totla No of Nodes::'};

dlgtitle = 'No of Nodes';

definput = {'30'};

opts.Interpreter = 'tex';

answer = inputdlg(prompt,dlgtitle,[1 40],definput,opts);

%%%%%%%%%%%%%%%%%%%%%%%%

%noOfNodes =21;% normal nodes

global noOfNodes;

noOfNodes=str2num(answer{1})

%%%%%%%%%%%%%%%%%%%

rand('state', 0);

figure(1);

clf;

hold on;

L = 100;% area aLimit

R = 25; % maximum range;

netXloc = rand(1,noOfNodes)\*L;

netYloc = rand(1,noOfNodes)\*L;

ANx = rand(1,5)\*L;

ANy = rand(1,5)\*L;

MNx = rand(1,1)\*L;

MNy = rand(1,1)\*L;

for gy=1:noOfNodes

plot(netXloc(gy), netYloc(gy), 'ro','linewidth',3);

text(netXloc(gy)+1, netYloc(gy), num2str(gy));

ylabel('Vertical Area');

xlabel('Horizontal Area');

pause(0.1);

end

plot(ANx, ANy, 'bs','linewidth',3);hold on;

text(ANx+1, ANy, 'BS');

plot(MNx, MNy, 'gs','linewidth',3);

text(MNx+1, MNy, 'MN');

title('Network Nodes Positioning');

hold off;

figure(2);

clf;

hold on;

for i = 1:noOfNodes

plot(netXloc(i), netYloc(i), 'r.','linewidth',2);

text(netXloc(i), netYloc(i), num2str(i));

for j = 1:noOfNodes

distance(i,j) = sqrt((netXloc(i) - netXloc(j))^2 + (netYloc(i) - netYloc(j))^2);

if distance(i,j) <= R

matrix(i, j) = 1; % there is a link;

line([netXloc(i) netXloc(j)], [netYloc(i) netYloc(j)], 'LineStyle', ':');

else

matrix(i, j) = inf;

end;

end;

end;

title('Link of each node to another in network');

figure(3);

clf;

hold on;

plot(netXloc, netYloc, 'ro','linewidth',3);

ylabel('Vertical Area');

xlabel('Horizontal Area');

plot(ANx, ANy, 'bs','linewidth',3);hold on;

text(ANx+1, ANy, 'BS');

plot(MNx, MNy, 'g<','linewidth',3);

%text(MNx+1, MNy, 'MN init');

title('Network Nodes Positioning');

hold on

% Random Way point based Mobility Generation for MN

s\_input = struct('V\_POSITION\_X\_INTERVAL',[0 100],...%(m)

'V\_POSITION\_Y\_INTERVAL',[0 100],...%(m)

'V\_SPEED\_INTERVAL',[0 5],...%(m/s)

'V\_PAUSE\_INTERVAL',[0 0.1],...%pause time (s)

'V\_WALK\_INTERVAL',[0 20],...%walk time (s)

'V\_DIRECTION\_INTERVAL',[-180 180],...%(degrees)

'SIMULATION\_TIME',100,...%(s)

'NB\_NODES',1);

s\_mobility = Generate\_Mobility(s\_input);

timeStep = 0.1;%(s)

test\_Animate(s\_mobility,s\_input,timeStep);

v\_t = 0:timeStep:s\_input.SIMULATION\_TIME;

for nodeIndex = 1:s\_mobility.NB\_NODES

%Simple interpolation (linear) to get the position, anytime.

%Remember that "interp1" is the matlab function to use in order to

%get nodes' position at any continuous time.

vs\_node(nodeIndex).v\_x = interp1(s\_mobility.VS\_NODE(nodeIndex).V\_TIME,s\_mobility.VS\_NODE(nodeIndex).V\_POSITION\_X,v\_t);

vs\_node(nodeIndex).v\_y = interp1(s\_mobility.VS\_NODE(nodeIndex).V\_TIME,s\_mobility.VS\_NODE(nodeIndex).V\_POSITION\_Y,v\_t);

end

for i = 1:length(vs\_node.v\_x)

MN1x=vs\_node.v\_x(i);

MN1y=vs\_node.v\_y(i);

for j = 1:noOfNodes

distanceMN(i,j) = sqrt((MN1x - netXloc(j))^2 + (MN1y - netYloc(j))^2);

end;

end;

actualBlindLocs = [xBlind', yBlind'];

actualAllLocs = [actualRefLocs; actualBlindLocs];

xActual = actualAllLocs(:,1)';

yActual = actualAllLocs(:,2)';

actualDist = L2\_distance(actualAllLocs', actualAllLocs',0);

sigmaOverN = 1.7;

% % If C==1, then this simulation runs the \_true\_ MLE.

% If C==exp( 0.5\* (log(10)/10 \*sigmaOverN)^2), then this runs a

% bias-corrected (pseudo-) MLE.

% C = exp( 0.5\* (log(10)/10 \*sigmaOverN)^2);

C = 1;

for trial = 1:trials,

dhat = actualDist.\*10.^(sigmaOverN/10 .\*symrandn(totalDevices))./C;

blindLocs0 = [xBlind, yBlind]; % Use the true coordinates (unrealistic but best case)

funcEvals = 0; dfuncEvals = 0;

p=blindLocs0;

ITMIN = 10; % the minimum number of iterations before exit

ITMAX = 300; % ITMAX is the maximum allowed number of iterations

EPS = 1.0e-10; % EPS is a small number to rectify the special case

disp('iter mean xi mini p xi maxi p xi fret');

disp('---- ---------- --------------------- --------------------- --------');

fp = feval(func, p);

xi = feval(dfunc, p);

exitCondition = 0;

g = -xi;

h = g;

xi = g;

extraLoops = 0;

maxExtraLoops = 10;

for its=1:ITMAX,

iter = its;

[p, xi, fret] = TriMinDist(p, xi, func, dfunc);

absxi = abs(xi);

[minv, mini] = min(absxi);

[maxv, maxi] = max(absxi);

meanv = mean(absxi);

outStr = sprintf('%4d %10.4g %2d %7.4g %10.4g %2d %7.4g %10.4g %8.4g', ...

iter, meanv, mini, p(mini), minv, maxi, p(maxi), maxv, fret);

disp(outStr);

if ( 2.0\*abs(fret-fp) <= ftol\*( abs(fret) + abs(fp) + EPS )),

if (its > ITMIN),

exitCondition = 1;

break;

end

end

fp = fret;

xi = feval(dfunc, p);

gg = sum(g.^2);

dgg = sum( (xi + g).\*xi );

if gg == 0,

exitCondition = 2;

break;

end

gam = dgg/gg;

g = -xi;

h = g + gam.\*h;

xi = h;

end

coordsMLE=p;

errorMin=fret;

figure(4)

clf

for i=1:blindDevices,

plot(blindLocs0(:,i), blindLocs0(:,blindDevices+i),'o','linewidth',10)

hold on

plot(coordsMLE(:,i), coordsMLE(:,blindDevices+i),'rp','linewidth',4)

hold on

plot(ANx, ANy, 'gs','linewidth',3);hold on;

text(ANx+1, ANy, 'BS');

% plot(vs\_node(i).v\_x ,vs\_node(i).v\_y ,'mp','linewidth',4);hold on

% text(vs\_node(i).v\_x+1 ,vs\_node(i).v\_y, 'MN');

end

legend('Unknown Node','Head','Anchor Node');

%| 6. Save the resulting estimated coords

coordEsts(trial, 1:blindCoords) = coordsMLE;

end % for trial

figure(4)

clf

for i=1:blindDevices,

plot(blindLocs0(:,i), blindLocs0(:,blindDevices+i),'o','linewidth',10)

hold on

plot(coordsMLE(:,i), coordsMLE(:,blindDevices+i),'rp','linewidth',4)

hold on

plot(ANx, ANy, 'gs','linewidth',4);hold on;

text(ANx+1, ANy, 'BS');

% plot(vs\_node(i).v\_x ,vs\_node(i).v\_y ,'mp','linewidth',4);hold on

% text(vs\_node(i).v\_x+1 ,vs\_node(i).v\_y, 'MN');

end

legend('Unknown Node','Head','Anchor node');

estMean = mean(coordEsts);

estCov = cov(coordEsts);

estVars = diag(estCov);

estStds = sqrt(estVars);

locVars = estVars(1:blindDevices) + estVars((blindDevices+1):(2\*blindDevices));

locStd = sqrt(locVars);

toc % show time of execution

figure; clf;

[locstdCRB, coordCRB] = FinalEst('R', [xBlind, actualRefLocs(:,1)'], ...

[yBlind, actualRefLocs(:,2)'], blindDevices, totalDevices, sigmaOverN);

for i=1:blindDevices,

hold on

R = cov(coordEsts(:,i), coordEsts(:,blindDevices+i));

drawOval(estMean(i), estMean(blindDevices+i), R, 'k-','v', 8, 0, 1);

R\_CRB = coordCRB([i, i+blindDevices],[i, i+blindDevices]);

drawOval(xBlind(i), yBlind(i), R\_CRB, 'r--','.',20, 0, 1);

end

% set(gca,'xlim',[-0.2 1.2])

% set(gca,'ylim',[-0.2 1.2])

set(gca,'FontSize',18)

xlabel('X Position (m)')

ylabel('Y Position (m)')

title('Final Position');

% Use for comparison

RMS\_est\_Std = sqrt(mean(locStd.^2))

RMS\_crb\_Std = sqrt(mean(locstdCRB.^2))

figure;

plot(linspace(1,5,length(locStd)),locStd\*0.87/max(locStd),'k-<','linewidth',2);hold on

plot(linspace(1,5,length(locStd)),locStd/max(locStd),'r-o','linewidth',2);hold on

plot(linspace(1,5,length(locStd)),locStd\*1.2/max(locStd),'b-s','linewidth',2);hold on

plot(linspace(1,5,length(locStd)),locStd\*1.5/max(locStd),'g-p','linewidth',2);hold on

plot(linspace(1,5,length(locStd)),locStd\*1.7/max(locStd),'m-d','linewidth',2);

xlabel('The number of location');

ylabel('Percentage Error(100%)');

%legend('Mobile aided','Proposed','DT','HF&KDE','NT');

title('Percentage error');

figure;

plot(linspace(-110,0,length(dhat)),sort(mean(dhat,1)),'r-<','linewidth',2);hold on

plot(linspace(-110,0,length(dhat)),sort(mean(dhat,1))\*0.5,'b-<','linewidth',2)

ylabel('RSSI(dB)');

xlabel('Received Power(dBm)');

title('Existing versus Hetrogenous signal power');

legend('Existing Models','GFDMA');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%GFDMA%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%

K=noOfNodes\*8;%number of subcarriers

M=noOfNodes;%number of subsymbols

r=0;%length of the cyclic prefix (CP) in multiples of 'subsymbols'

CP=r\*K;

a=1;%roll-off

% Symbol source

s = 1/sqrt(2)\*(sign(randn(K,M))+1i\*sign(randn(K,M)));%basic example

s(:,1) = 0;%1st symbol null

if r>0 s(:,M-r+1) = 0;

end% M-r symbol null

s([(K/8:3\*K/8)+K/2],:) = 0;%some null subcarriers

d=reshape(s,[K\*M 1]); % Split into real and imag

di = real(d);

dq = imag(d); % Meyer RRC (defined in time)

R=((0:(K-1))'-K/2-eps)/(a\*K)+1/2;

R(R<0)=0;

R(R>1)=1;

F=1-R;% Ramp rise/fall

R=R.^4.\*(35 - 84\*R+70\*R.^2-20\*R.^3);

F=1-R;% Meyer auxiliary function

R=1/2\*(cos(F\*pi)+1);

F=1-R;% Meyer RC rise/fall

R=sqrt(R);F=sqrt(F);%Meyer RRC

g=[F;zeros((M-2)\*K,1);R];

g=g/sqrt(sum(g.^2));%normalization

% Frequency shift oqam

gi = g;

gq = ifft(circshift(fft(gi), M/2));

plot(real([gi gq]))

% Ai matrix

Ai = zeros(M\*K, M\*K);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

n = 0:M\*K-1; n=n';

w = exp(1j\*2\*pi/K);

for k=0:K-1

for m=0:M-1

Ai(:,m\*K+k+1) = 1i^(mod(m,2))\*circshift(gi, m\*K) .\* w.^(k\*n);

end

end

%surf(abs(Ai))

%Aq matrix

Aq = zeros(M\*K, M\*K);

for k=0:K-1

for m=0:M-1

Aq(:,m\*K+k+1) = 1i^(mod(m,2)+1)\*circshift(gq, m\*K) .\* w.^(k\*n);

end

end %surf(abs(Aq))

x=Ai\*di+Aq\*dq; % Add CP

xcp = [x([end-CP+(1:CP)],:);x];

subplot(2,1,1);plot(abs(xcp))

Xcp=fft(xcp,4\*M\*K);

Xcp=Xcp/std(Xcp);%normalize to ~0dB

subplot(2,1,2);

plot(mag2db(abs(Xcp)));

ylim([-80 10]);

grid on

%add channel, sync, remove cp, equalize

dhat=real(Ai'\*x)+1i\*real(Aq'\*x)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%BER%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%clear all

%close all

%clc

%nbits = 208000;

%modlevel = 2 ;

%nbitpersym = 52; % number of bits per qam OFDM symbol (same as the number of subcarriers for 16-qam)

%nsym = 10^4; % number of symbols

%len\_fft = 64; % fft size

%sub\_car = 52; % number of data subcarriers

%EbNo = 0:2:15;

%EsNo= EbNo+10\*log10(52/64)+ 10\*log10(64/80) +10\*log10(4);

%snr=EsNo - 10\*log10((64/80));

%%M = modem.qammod('M',16); % modulation object

%%M = qammod('M',16);

%x = randi([0, 1], log2(16), 1);

%M = qammod(x, 16, 'InputType', 'bit', 'UnitAveragePower', true, 'PlotConstellation', true);

% Generating data

%t\_data=randi(nbitpersym\*nsym\*4,1,2);

%qamdata=bi2de(reshape(t\_data,4,520000).','left-msb');

%maping = bin2gray(qamdata,'qam',16);

% modulating data

%mod\_data =1/sqrt(10)\* modulate(M,maping);

% serial to parallel conversion

%par\_data = reshape(mod\_data,nbitpersym,nsym).';

% pilot insertion

%pilot\_ins\_data=[zeros(nsym,6) par\_data(:,[1:nbitpersym/2]) zeros(nsym,1) par\_data(:,[nbitpersym/2+1:nbitpersym]) zeros(nsym,5)] ;

% fourier transform time doamain data

%IFFT\_data =ifft(fftshift(pilot\_ins\_data.')).';

%a=max(max(abs(IFFT\_data)));

%IFFT\_data=IFFT\_data./a; % normalization

% addition cyclic prefix

%cylic\_add\_data = [IFFT\_data(:,[49:64]) IFFT\_data].';

% parallel to serial coversion

%ser\_data = reshape(cylic\_add\_data,80\*nsym,1);

% passing thru channel

%no\_of\_error=[];

%ratio=[];

%for ii=1:length(snr)

%chan\_awgn = awgn(ser\_data,snr(ii),'measured'); % awgn addition

%ser\_to\_para = reshape(chan\_awgn,80,nsym).'; % serial to parallel coversion

%cyclic\_pre\_rem = ser\_to\_para(:,[17:80]); %cyclic prefix removal

%FFT\_recdata =a\*fftshift(fft(cyclic\_pre\_rem.')).'; % freq domain transform

%rem\_pilot = FFT\_recdata (:,[6+[1:nbitpersym/2] 7+[nbitpersym/2+1:nbitpersym] ]); %pilot removal

%ser\_data\_1 =sqrt(10)\* reshape(rem\_pilot.',nbitpersym\*nsym,1); % serial coversion

%z = qammod('M',16);

%x = randi([0, 1], log2(16), 1);

%z = qammod(x, 16, 'InputType', 'bit', 'UnitAveragePower', true, 'PlotConstellation', true);

%z=modem.qamdemod('M',16);

%demod\_Data = demodulate(z,ser\_data\_1); %demodulatin the data

%demaping = gray2bin(demod\_Data,'qam',16);

%data1 = de2bi(demaping,'left-msb');

%data2 = reshape(data1.',nbitpersym\*nsym\*4,1);

%[no\_of\_error(ii),ratio(ii)]=biterr(t\_data , data2) ; % error rate calculation

%end

% plotting the result

%semilogy(EbNo,ratio,'--\*r','linewidth',2);

%hold on;

%theoryBer = (1/4)\*3/2\*erfc(sqrt(4\*0.1\*(10.^(EbNo/10))));

%semilogy(EbNo,theoryBer ,'--b','linewidth',2);

%axis([0 15 10^-5 1])

%legend('simulated','theoritical')

%grid on

%xlabel('EbNo');

%ylabel('BER')

%title('Bit error probability curve for qam using OFDM');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%55

s=1024;

w=365;

b=0:0.1:100;

d=w.\*b;

e=s./d;

c=b.\*log2(1+e);

plot(b,c);

xlabel('Bandwidth --->');

ylabel('Capicaty--->');

grid;