SIR C.R.REDDY COLLEGE OF ENGINEERING
ELURU-534007

Department of Electronics and Communications

DIGITAL SIGNAL PROCESSING Lab Manual (MTCS - 15)
For I / II M.Tech (Communication Systems), II - Semester

SIR C.R.REDDY COLLEGE OF ENGINEERING
ELURU-534007

Digital Signal Processing Lab

LIST OF EXPERIMENTS

1. SAMPLING & RECONSTRUCTION OF ANALOG SIGNALS

2. FILTERING OF SIGNALS

3. SPECTRUM ANALYSIS USING FFT

4. PULSE CODE MODULATION

5. DELTA MODULATION

6. CDMA SIGNAL DETECTION IN GAUSSIAN NOISE USING MATCHED FILTER RECEIVER AND DECORRELATOR

7. **ADAPTIVE CHANNEL EQUALIZATION USING LMS ALGORTHIM**

8. CONVOLUTION CODING AND VITERBI DECODING

9. DIGITAL MODULATION AND DEMODULATION

10. HUFFMAN SOURCE CODING

11. OVERALL COMMUNICATION SYSTEM

12. FADING CHANNEL SIMULATION

1.SAMPLING &RECONSTRUCTION OF ANALOG SIGNALS

AIM:

Write a MATLAB program for Sampling &reconstruction of analog signals.

PROGRAM:

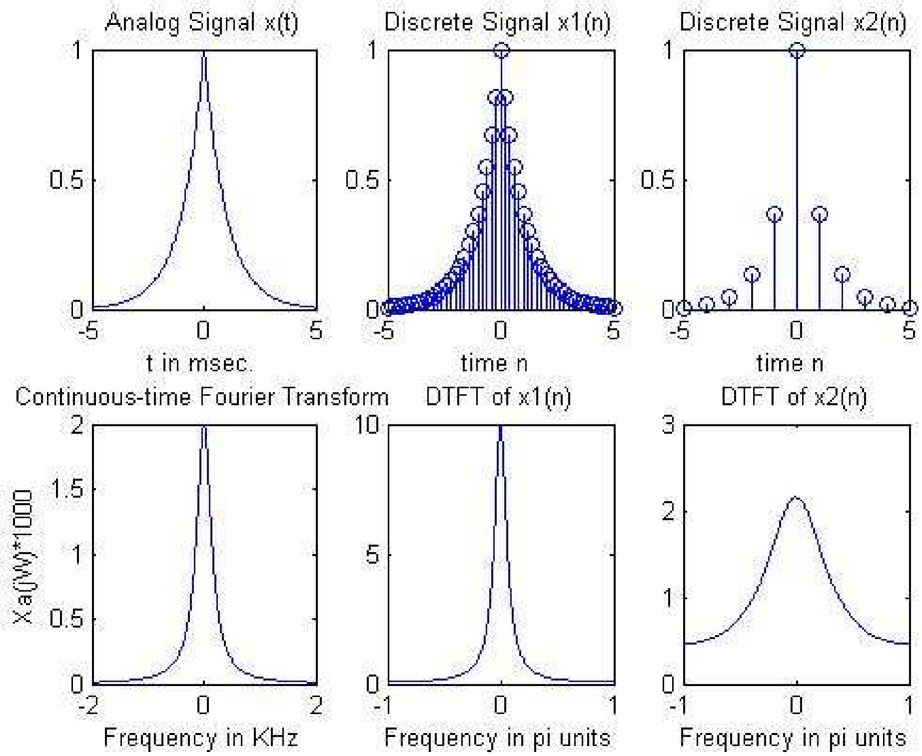
```
%%      Sampling & Reconstruction of Analog Signals
% Analog Signal: x(t)=exp(-a*abs(t)), X(jw)=2a/(a^2+w^2)
clear;
Dt = 0.00005;Fs=1/Dt;
t = -0.005:Dt:0.005;
xa = exp(-1000*abs(t));

% Discrete-time Signal x1(n): sampled above Nyquist rate
Ts1 = 0.0002; Fs1 = 1/Ts1; n1 = -25:1:25;
 nTs1 = n1*Ts1;
xn1 = exp(-1000*abs(nTs1));

% Discrete-time Signal x2(n): sampled below Nyquist rate
Ts2 = 0.001; Fs2 = 1/Ts2; n2 = -5:1:5;
nTs2 = n2*Ts2;
xn2 = exp(-1000*abs(nTs2));

% Continuous-time Fourier Transform
Wmax = 2*pi*2000;
K = 500; k = 0:1:K;
W = k*Wmax/K;

%Xa1 = xa * exp(-j*t'*W) * Dt; Xa1 = real(Xa1); %numerical approx of FT
Xa=2e3./(1e6+W.^2); %exact
W = [-fliplr(W), W(2:501)]; % Omega from -Wmax to Wmax
Xa = [fliplr(Xa), Xa(2:501)]; %CTFT

% Discrete-time Fourier transform(DTFT) of x1(n) & x2(n)
K = 500; k = 0:1:K; %compute DTFT at K points
w = pi*k/K; %freq points
X1 = xn1 * exp(-j*n1'*w);X1 = real(X1);
X2 = xn2 * exp(-j*n2'*w);X2= real(X2);
w = [-fliplr(w), w(2:K+1)]; %freq axis at 2K+1 points
X1 = [fliplr(X1), X1(2:K+1)];X2 = [fliplr(X2), X2(2:K+1)]; %DTFT


% plots
subplot(2,3,1);plot(t*1000,xa);xlabel('t in msec.');title('Analog Signal x(t)')
subplot(2,3,2);stem(n1*Ts1*1000,xn1);xlabel('time n '); title('Discrete Signal x1(n)');
```

```
subplot(2,3,3);stem(n2*Ts2*1000,xn2);xlabel('time n '); title('Discrete Signal x2(n)');
subplot(2,3,4);plot(W/(2*pi*1000),Xa*1000);
xlabel('Frequency in KHz'); ylabel('Xa(jW)*1000');title('Continuous-time Fourier
Transform')
subplot(2,3,5);plot(w/pi,X1);xlabel('Frequency in pi units');title('DTFT of x1(n)');
subplot(2,3,6);plot(w/pi,X2);xlabel('Frequency in pi units');title('DTFT of x2(n)');

% Analog Signal Reconstruction with Low Pass Filter

% Interpolation using sinc function
xa1 = xn1 * sinc(Fs1*(ones(length(nTs1),1)*t-nTs1'*ones(1,length(t))));
error1 = max(abs(xa1 - exp(-1000*abs(t)))) %print error
xa2 = xn2 * sinc(Fs2*(ones(length(nTs2),1)*t-nTs2'*ones(1,length(t))));
error2 = max(abs(xa2 - exp(-1000*abs(t))))

% Plots
figure;
subplot(2,2,1);plot(t*1000,xa);xlabel('t in msec.');title('Original Signal x(t)')
subplot(2,2,2);plot(t*1000,xa1);xlabel('t in msec.'); ylabel('x1(t)')
title('Reconstructed Signal from x1(n) using sinc function'); hold on
subplot(2,2,3);plot(t*1000,xa2);xlabel('t in msec.'); ylabel('x2(t)')
title('Reconstructed Signal from x2(n) using sinc function'); hold on

% Practical Reconstruction with Zero Order Hold/First Order Hold
% using MATLAB stairs(ZOH) and plot(FOH) functions

% Analog Signal reconstruction from x1(n)
figure;
subplot(2,2,1); stairs(nTs1*1000,xn1);xlabel('t in msec.'); ylabel('xa1(t)')
title('Reconstructed Signal from x1(n) using zero-order-hold'); hold on
subplot(2,2,2); plot(nTs1*1000,xn1);xlabel('t in msec.'); ylabel('xa1(t)')
title('Reconstructed Signal from x1(n) using first-order-hold'); hold on

% Analog Signal reconstruction from x2(n)
subplot(2,2,3); stairs(nTs2*1000,xn2);xlabel('t in msec.'); ylabel('xa2(t)')
title('Reconstructed Signal from x2(n) using zero-order-hold'); hold on
subplot(2,2,4); plot(nTs2*1000,xn2);xlabel('t in msec.'); ylabel('xa2(t)')
title('Reconstructed Signal from x2(n) using first-order-hold'); hold on
```

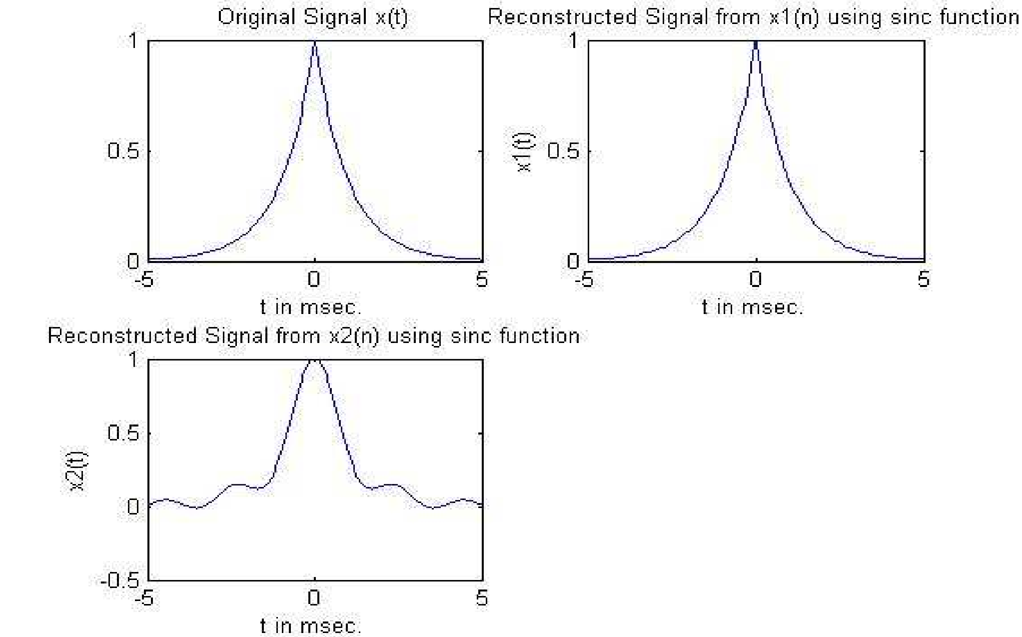
RESULT:

Sampling & reconstruction of analog signals are generated and plotted using MATLAB.

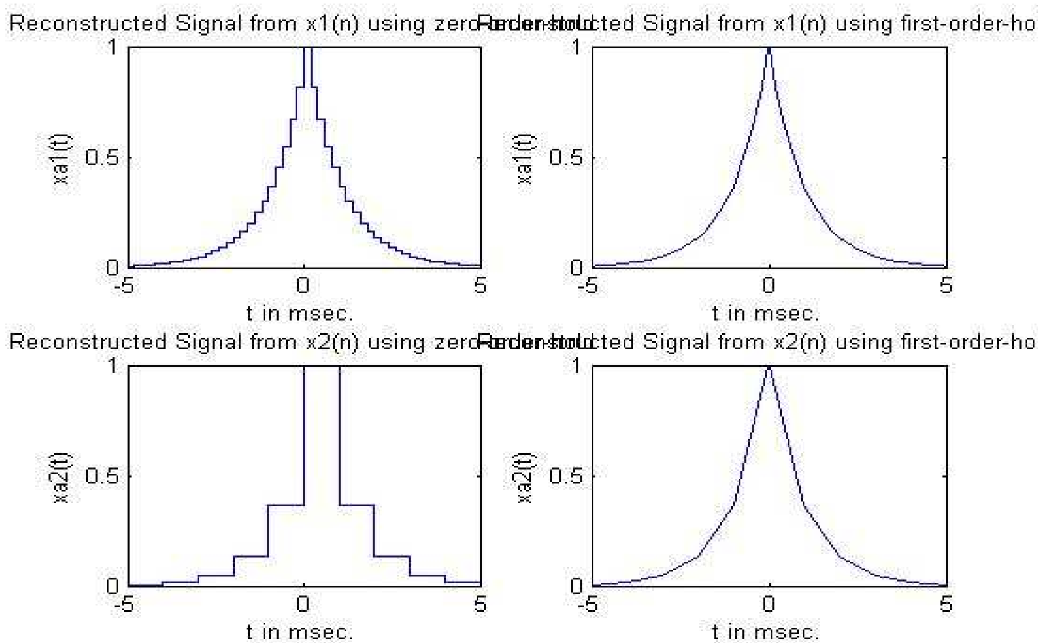
Wave Forms:



Analog Signal x(t)
Discrete Signal x1(n)
Discrete Signal x2(n)
1
0.5
0
-5
0
5
t in msec.
time n
time n
Continuous-time Fourier Transform
DTFT of x1(n)
DTFT of x2(n)
Xa(jW)*1000
2
1.5
1
0.5
10
3
-2
-1
Frequency in KHz
Frequency in pi units
Frequency in pi units



Original Signal x(t)
t in msec.
Reconstructed Signal from x1(n) using sinc function
x1(t)
t in msec.
Reconstructed Signal from x2(n) using sinc function
x2(t)
t in msec.



Reconstructed Signal from x1(n) using zer
xa1(t)
t in msec.
ted Signal from x1(n) using first-order-hol
xa1(t)
t in msec.
Reconstructed Signal from x2(n) using zer
xa2(t)
t in msec.
ted Signal from x2(n) using first-order-hol
xa2(t)
t in msec.

2. FILTERING OF SIGNALS

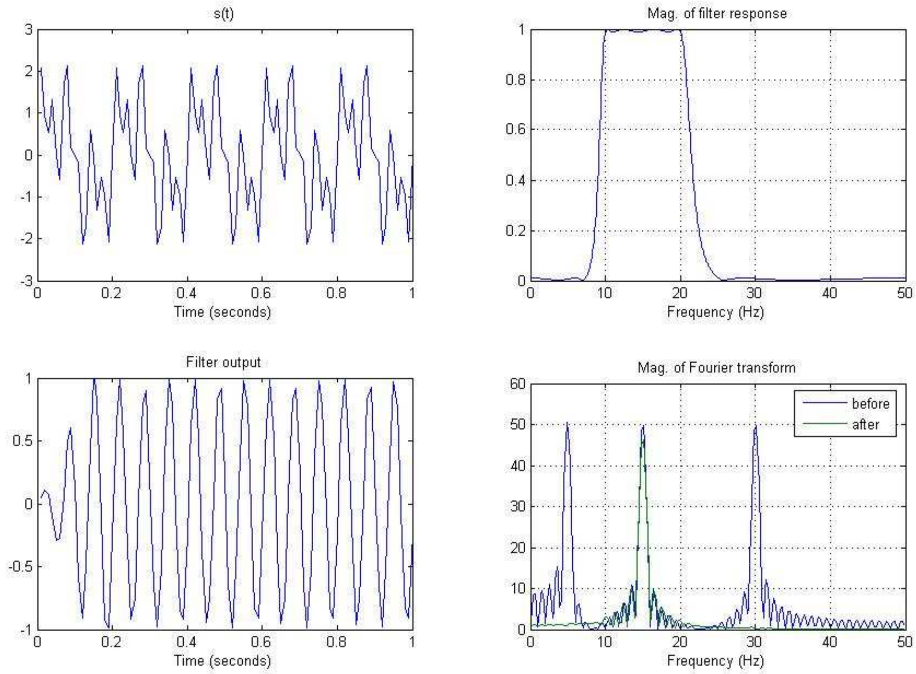
AIM:
Write a MATLAB program for filtering of signals.

PROGRAM:

```
%%      Filtering of Signals

Fs = 100; % sampling freq
t = (1:100)/Fs; % time axis
s1 = sin(2*pi*t*5); s2=sin(2*pi*t*15); s3=sin(2*pi*t*30); % 3 freq are present
s = s1+s2+s3; % signal
subplot(2,2,1),plot(t,s);xlabel('Time (seconds)');title('s(t)');%ylabel('Time waveform');

% design a digital filter(BP) to select 15Hz component
[b,a] = ellip(4,0.1,40,[10 20]*2/Fs); % Elliptic filter
[H,w] = freqz(b,a,512); % compute frequency response
subplot(2,2,2),plot(w*Fs/(2*pi),abs(H));title('Mag. of filter response');
xlabel('Frequency (Hz)');
grid;

% send signal through filter
sf = filter(b,a,s);
subplot(2,2,3), plot(t,sf); xlabel('Time (seconds)'); title('Filter output');
axis([0 1 -1 1]);
S = fft(s,512); SF = fft(sf,512); w = (0:255)/256*(Fs/2); % compute spectrum
subplot(2,2,4),plot(w,abs([S(1:256)' SF(1:256)']));
title('Mag. of Fourier transform');
xlabel('Frequency (Hz)');
grid;
legend({'before','after'})
```

RESULT:
Filtering of signals is performed using MATLAB.

Wave Forms:


s(t)
Time (seconds)
Mag. of filter response
Frequency (Hz)
Filter output
Time (seconds)
Mag. of Fourier transform
before
after
Frequency (Hz)

3. SPECTRUM ANALYSIS USING FFT

AIM:

Write a MATLAB program for spectrum analysis using fft.

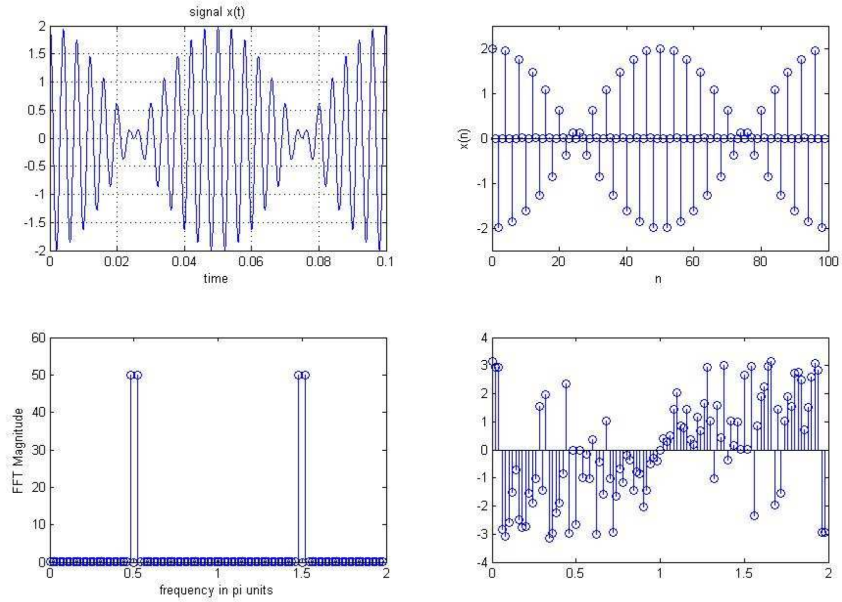
PROGRAM:

```
%%      Spectrum Analysis using FFT

% Nyquist freq=2*Fh, Fs>=2*Fh
% normalized digital freq=(2*pi*f)/Fs
% FFT resolution=Fs/N
% consider zero-padding for better display of FFT
% x(t)=cos(w1*t)+cos(w2*t)

clc;clear all; close all;
f1=240;f2=260;Fs=1000;Ts=1/Fs;Np=100;

% analog signal
t=0:0.0001:Ts*Np;xa=cos(2*pi*f1*t)+cos(2*pi*f2*t);

% discrete signal
n=[0:Np-1];
x=cos(2*pi*f1*n*Ts)+cos(2*pi*f2*n*Ts);
X=fft(x,length(x));magX=abs(X);
w=2*pi/Np*n;
figure;subplot(2,2,1);plot(t,xa);title('signal x(t)');xlabel('time');grid
subplot(2,2,2);stem(n,x);;xlabel('n');ylabel('x(n)')
axis([0,Np,-2.5,2.5])
subplot(2,2,3);stem(w/pi,magX);;xlabel('frequency in pi units');
ylabel('FFT Magnitude');
subplot(2,2,4);stem(w/pi,angle(X));
xlabel('frequency in pi units');
ylabel('FFT Phase');
```

RESULT:

Spectrum analysis using fft is performed using MATLAB.

Wave Forms:



4. PULSE CODE MODULATION

AIM:

Write a MATLAB program to generate pulse code modulation

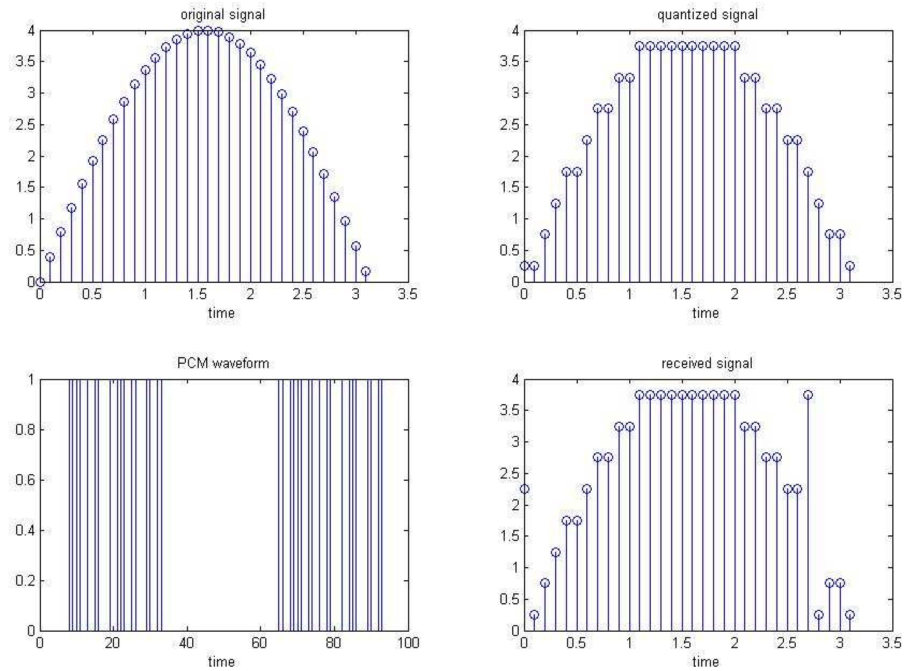
PROGRAM:

```
%%      Pulse Code Modulation (PCM)

P=5; %percentage of errors in transmission

% sampling
t = [0:0.1:1*pi]; % Times at which to sample the sine function
sig = 4*sin(t); % Original signal, a sine wave
%sig=exp(-1/3*t);

% quantization
Vh=max(sig);Vl=min(sig);
N=3;M=2^N;S=(Vh-Vl)/M; %design N-bit uniform quantizer with stepsize=S
partition = [Vl+S:S:Vh-S]; % Length M-1, to represent M intervals
codebook = [Vl+S/2:S:Vh-S/2]; % Length M, one entry for each interval

% partition = [-1:.2:1]; % Length 11, to represent 12 intervals
% codebook = [-1.2:.2:1]; % Length 12, one entry for each interval
[index,quantized_sig,distor] = quantiz(sig,partition,codebook); % Quantize.

% binary encoding
codedsig=de2bi(index,'left-msb');codedsig=codedsig';
txbits=codedsig(:); %serial transmit
errvec=randsrc(length(txbits),1,[0 1;(1-P/100) P/100]); %error vector

%rxbits=xor(txbits,errvec);
rxbits=rem(txbits+errvec,2); %bits received

rxbits=reshape(rxbits,N,length(sig));rxbits=rxbits';
index1=bi2de(rxbits,'left-msb'); %decode
reconstructedsig=codebook(index1+1); %re-quantize

%plot(t,sig,'x',t,quantized_sig,'.')
%plot(t,sig,'x-',t,quantized_sig,'.--',t,reconstructedsig,'d-')
%figure,stem(t,sig,'x-');hold;stem(t,quantized_sig,'.--');stem(t,reconstructedsig,'d-');
figure,subplot(2,2,1); stem(t,sig); xlabel('time'); title('original signal');
subplot(2,2,2); stem(t,quantized_sig); xlabel('time'); title('quantized signal');
tt=[0:N*length(t)-1];
subplot(2,2,3);stairs(tt,txbits); xlabel('time'); title('PCM waveform');
subplot(2,2,4);stem(t,reconstructedsig); xlabel('time'); title('received signal');
```

RESULT:

Pulse code modulation is performed using MATLAB.

Wave Forms:



original signal
quantized signal
PCM waveform
received signal
time
time
time
time

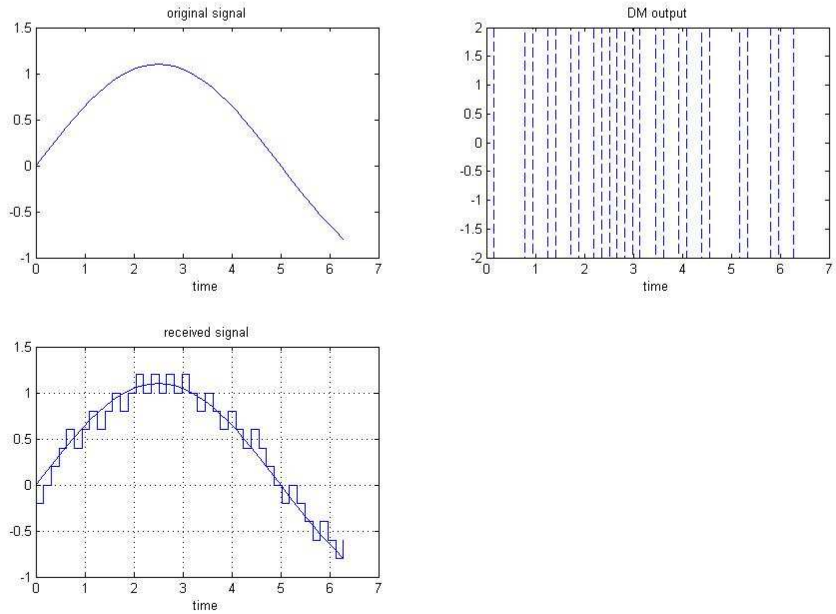
5. DELTA MODULATION

AIM:
Write a MATLAB program to perform delta modulation

PROGRAM:
```
%%      Delta Modulation (DM)

%delta modulation = 1-bit differential pulse code modulation (DPCM)
predictor = [0 1]; % y(k)=x(k-1)
%partition = [-1:.1:.9];codebook = [-1:.1:1];
step=0.2; %SFs>=2pifA
partition = [0];codebook = [-1*step step]; %DM quantizer

t = [0:pi/20:2*pi];
x = 1.1*sin(2*pi*0.1*t); % Original signal, a sine wave
%t = [0:0.1:2*pi];x = 4*sin(t);
%x=exp(-1/3*t);
%x = sawtooth(3*t); % Original signal

% Quantize x(t) using DPCM.
encodedx = dpcmenco(x,codebook,partition,predictor);

% Try to recover x from the modulated signal.
decodedx = dpcmdeco(encodedx,codebook,predictor);
distor = sum((x-decodedx).^2)/length(x) % Mean square error

% plots

figure,subplot(2,2,1);plot(t,x);xlabel('time');title('original signal');
subplot(2,2,2);stairs(t,10*codebook(encodedx+1),'--');xlabel('time');title('DM output');
subplot(2,2,3);plot(t,x);hold;stairs(t,decodedx);grid;xlabel('time');title('received signal');
```

RESULT:
Delta modulation is performed using MATLAB.

Wave Forms:



6. CDMA SIGNAL DETECTION IN GAUSSIAN NOISE USING MATCHED FILTER RECEIVER AND DECORRELATOR

AIM:

Write a MATLAB program to CDMA signal detection in Gaussian noise using matched filter receiver and decorrelator

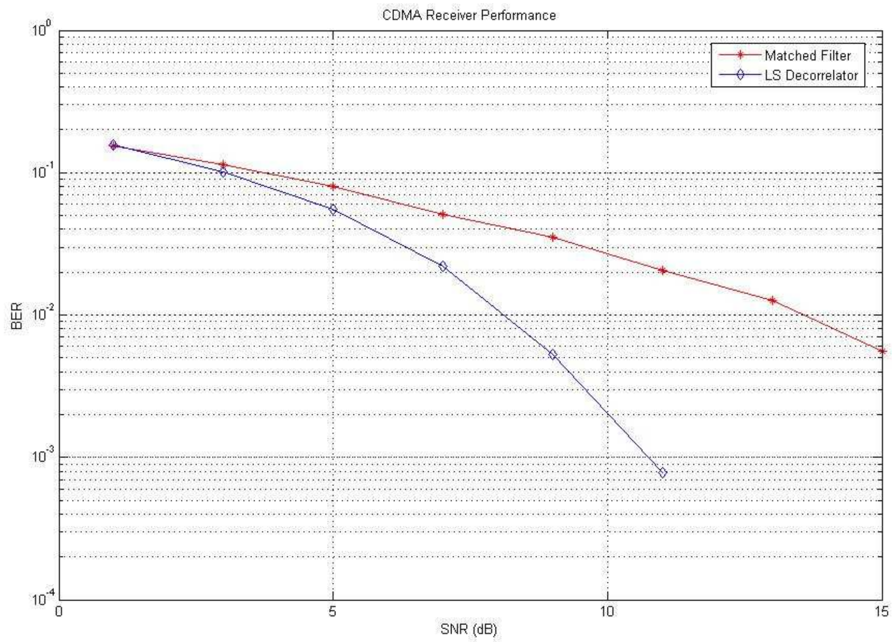
PROGRAM:

```
%% CDMA Signal Detection in Gaussian Noise using MF receiver and
%% De-correlator(faster code)

clear; tic,
K=8; N=31; M=10000; % no of bits
bsize=1000;nblocks=50; Nb=M/bsize;
snr_db=[1:2:15];
S2=randsrc(N,K); S=(1/sqrt(N))*S2;

R1=S'*S;s1=S(:,1); % user-1 sequence
LS=inv(R1)*S'; % Least-Squares detector

snr=10.^(0.1*snr_db); % linear snr
ber1=zeros(1,length(snr)); ber2=zeros(1,length(snr));
B=[];T=[];R=[]; %preallocate

A1=1;           % user 1 amplitude=fixed,vary noise var
A2=A1;          % user 2 relative amplitude
A3=A1; A4=A1; A5=A1; A6=A1; A=diag([A1,A2,A3,A4,A5,A6]);
A=eye(K);
H=S*A; % channel matrix
   for i=1:length(snr)
  var=1/snr(i);err1=0;err2=0; % initialize
  for k=1:nblocks %block processing
W=sqrt(var)*randn(N,bsize);
B=randsrc(K,bsize);    % user data matrix
T=H*B;                 % transmitted CDMA signal block
R=T+W;                 % received signal  block

z1=s1'*R;b1emf=sign(A1*z1); err1=err1+sum(b1emf ~= B(1,:)); %matched filter
z2=LS*R;b1ed=sign(A1*z2); err2=err2+sum(b1ed(1,:) ~= B(1,:)); %Decorrelator

end; % k loop
ber1(i)=err1/(bsize*nblocks); ber2(i)=err2/(bsize*nblocks);
end;toc
% plot
figure,semilogy(snr_db,ber1,'-r*'); hold on;
semilogy(snr_db,ber2,'-bd'); grid
xlabel('SNR (dB)'); ylabel('BER'); title('CDMA Receiver Performance')
legend('Matched Filter','LS Decorrelator');
```

RESULT:

CDMA signal detection in Gaussian noise using matched filter receiver and decorrelator is performed using MATLAB.

Wave Forms:



7. ADAPTIVE CHANNEL EQUALIZATION USING LMS ALGORTHIM

AIM:

Write a MATLAB program to Adaptive channel equalization using LMS algorthim.

PROGRAM:

```
%%      Channel Equalization using Adaptive Filters
%%      1. Basic Procedure for Equalizing a Signal

% Build a set of test data
x = pskmod(randint(1000,1),2); % BPSK symbols
rxsig = conv(x,[1 0.8 0.3]);  % Received signal with ISI
% Create an equalizer object.
eqlms = lineareq(8,lms(0.03));
% Change the reference tap index in the equalizer.
eqlms.RefTap = 4;
% Apply the equalizer object to a signal.
y = equalize(eqlms,rxsig,x(1:200));

%%      2. Equalizing Using a Training Sequence

% Set up parameters and signals.
M = 4; % Alphabet size for modulation
msg = randint(1500,1,M); % Random message
modmsg = pskmod(msg,M); % Modulate using QPSK.
trainlen = 500; % Length of training sequence
chan = [.986; .845; .237; .123+.31i]; % Channel coefficients
chan = [1 0.8 0.3];

filtmsg = filter(chan,1,modmsg); % Introduce channel distortion.

% Equalize the received signal.
eq1 = lineareq(8, lms(0.01)); % Create an equalizer object.
eq1.SigConst = pskmod([0:M-1],M); % Set signal constellation.
% Change the reference tap index in the equalizer.
eq1.RefTap = 4;

[symbolest,yd,e] = equalize(eq1,filtmsg,modmsg(1:trainlen)); % Equalize.
plot(20*log10(abs(e(1:250))));

% Plot signals.
h = scatterplot(filtmsg,1,trainlen,'bx'); hold on;
scatterplot(symbolest,1,trainlen,'g.',h);
scatterplot(eq1.SigConst,1,0,'k*',h);
legend('Filtered signal','Equalized signal',...
   'Ideal signal constellation');
```

```
hold off;
% Compute error rates with and without equalization.
demodmsg_noeq = pskdemod(filtmsg,M); % Demodulate unequalized signal.
demodmsg = pskdemod(yd,M); % Demodulate detected signal from equalizer.
[nnoeq,rnoeq] = symerr(demodmsg_noeq(trainlen+1:end),...
   msg(trainlen+1:end));
[neq,req] = symerr(demodmsg(trainlen+1:end),...
   msg(trainlen+1:end));
disp('Symbol error rates with and without equalizer:')
disp([req rnoeq])

%%      3. Equalizing in Decision-Directed Mode

M = 4; % Alphabet size for modulation
msg = randint(1500,1,M); % Random message
modmsg = pskmod(msg,M); % Modulate using QPSK.
trainlen = 500; % Length of training sequence
chan = [.986; .845; .237; .123+.31i]; % Channel coefficients
filtmsg = filter(chan,1,modmsg); % Introduce channel distortion.

% Set up equalizer.
eqlms = lineareq(8, lms(0.01)); % Create an equalizer object.
eqlms.SigConst = pskmod([0:M-1],M); % Set signal constellation.
% Maintain continuity between calls to equalize.
eqlms.ResetBeforeFiltering = 0;

% Equalize the received signal, in pieces.
% 1. Process the training sequence.
s1 = equalize(eqlms,filtmsg(1:trainlen),modmsg(1:trainlen));
% 2. Process some of the data in decision-directed mode.
s2 = equalize(eqlms,filtmsg(trainlen+1:800));
% 3. Process the rest of the data in decision-directed mode.
s3 = equalize(eqlms,filtmsg(801:end));
s = [s1; s2; s3]; % Full output of equalizer

%%      4. Delays from Equalization

M = 2; % Use BPSK modulation for this example.
msg = randint(1000,1,M); % Random data
modmsg = pskmod(msg,M); % Modulate.
trainlen = 100; % Length of training sequence
trainsig = modmsg(1:trainlen); % Training sequence

% Define an equalizer and equalize the received signal.
eqlin = lineareq(3,normlms(.0005,.0001),pskmod(0:M-1,M));
eqlin.RefTap = 2; % Set reference tap of equalizer.
[eqsig,detsym] = equalize(eqlin,modmsg,trainsig); % Equalize.

detmsg = pskdemod(detsym,M); % Demodulate the detected signal.
```

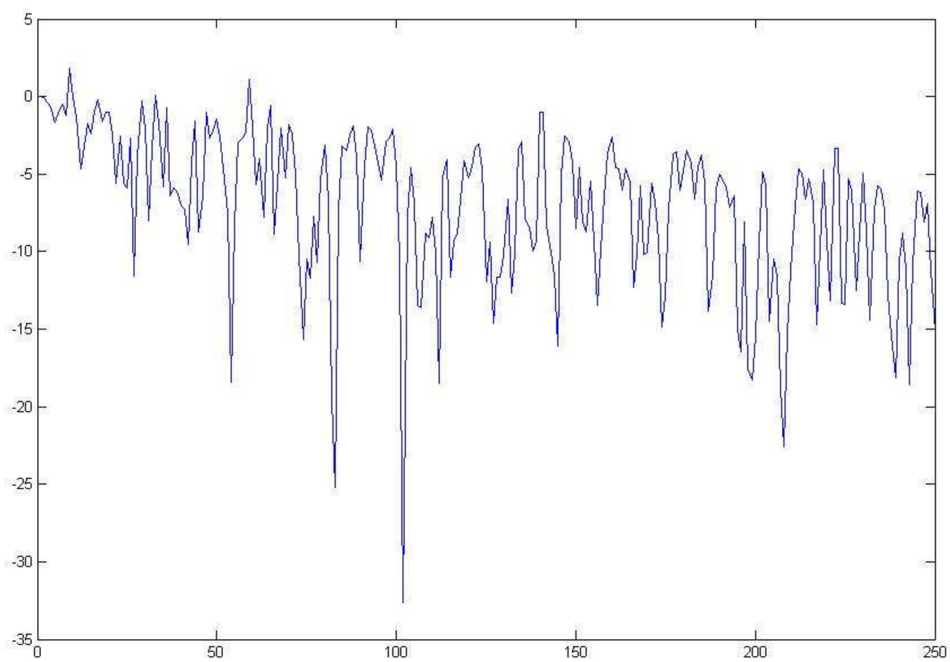
```
% Compensate for delay introduced by RefTap.
D = (eqlin.RefTap -1)/eqlin.nSampPerSym;
trunc_detmsg = detmsg(D+1:end); % Omit first D symbols of equalized data.
trunc_msg = msg(1:end-D); % Omit last D symbols.

% Compute bit error rate, ignoring training sequence.
[numerrs,ber] = biterr(trunc_msg(trainlen+1:end),...
  trunc_detmsg(trainlen+1:end))

%%      5. Equalizing Using a Loop

% Set up parameters.
M = 16; % Alphabet size for modulation
sigconst = qammod(0:M-1,M); % Signal constellation for 16-QAM
chan = [1 0.45 0.3+0.2i]; % Channel coefficients

% Set up equalizers.
eqrls = lineareq(6, rls(0.99,0.1)); % Create an RLS equalizer object.
eqrls.SigConst = sigconst; % Set signal constellation.
eqrls.ResetBeforeFiltering = 0; % Maintain continuity between iterations.

eqlms = lineareq(6, lms(0.003)); % Create an LMS equalizer object.
eqlms.SigConst = sigconst; % Set signal constellation.
eqlms.ResetBeforeFiltering = 0; % Maintain continuity between iterations.

eq_current = eqrls; % Point to RLS for first iteration.

% Main loop
for jj = 1:4
  msg = randint(500,1,M); % Random message
  modmsg = qammod(msg,M); % Modulate using 8-QAM.

% Set up training sequence for first iteration.
  if jj == 1
    ltr = 200; trainsig = modmsg(1:ltr);
  else
    % Use decision-directed mode after first iteration.
    ltr = 0; trainsig = [];
  end

  % Introduce channel distortion.
  filtmsg = filter(chan,1,modmsg);

  % Equalize the received signal.
  [s,sd,e] = equalize(eq_current,filtmsg,trainsig);

  % Plot signals.
  h = scatterplot(filtmsg(ltr+1:end),1,0,'bx'); hold on;
  scatterplot(s(ltr+1:end),1,0,'g.',h);
  scatterplot(sigconst,1,0,'k*',h);
```

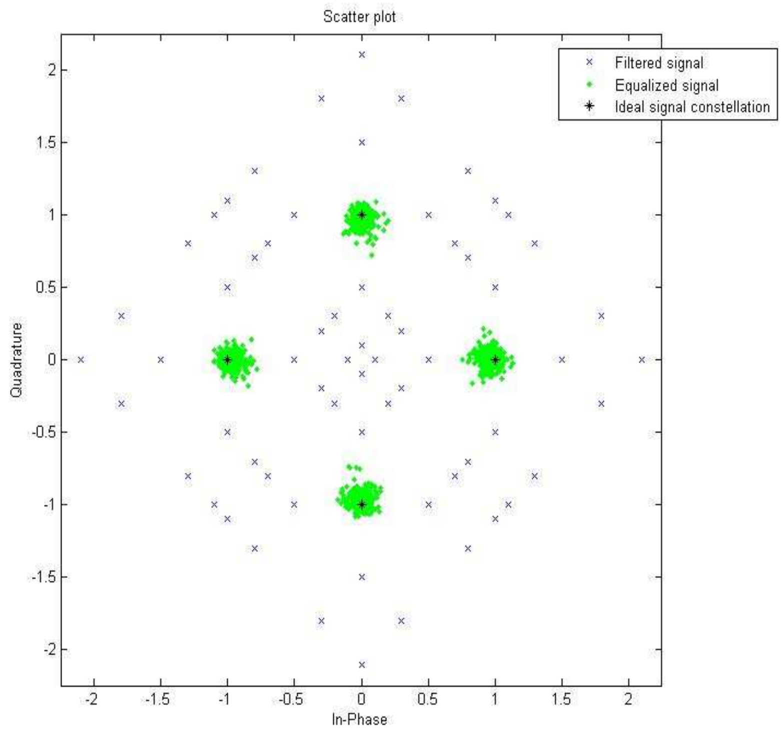
```
   legend('Received signal','Equalized signal','Signal constellation');
   title(['Iteration #' num2str(jj) ' (' eq_current.AlgType ')']);
   hold off;

   % Switch from RLS to LMS after second iteration.
   if jj == 2
      eqlms.WeightInputs = eq_current.WeightInputs; % Copy final inputs.
      eqlms.Weights = eq_current.Weights; % Copy final weights.
      eq_current = eqlms; % Make eq_current point to eqlms.
   end
end
figure,plot(20*log10(abs(e(1:500))));

%%
```

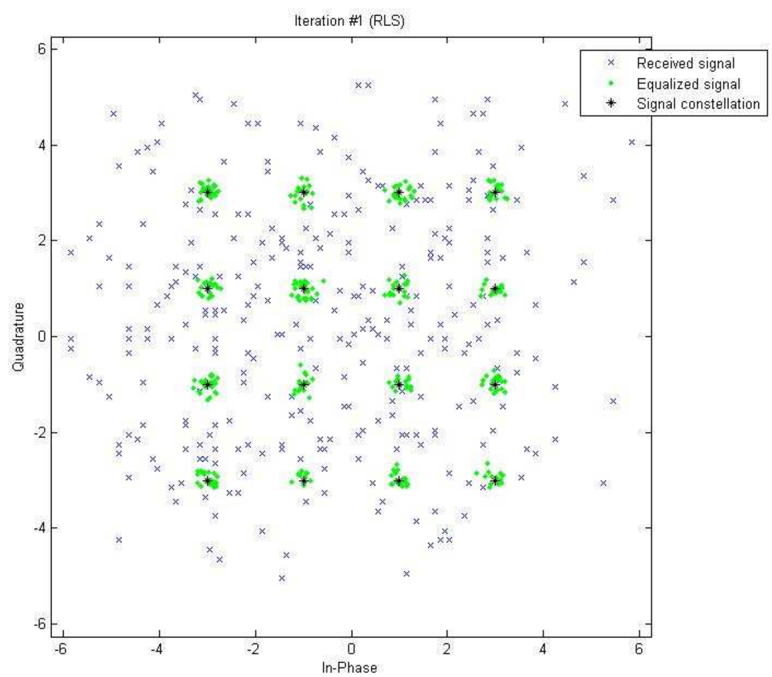
RESULT:

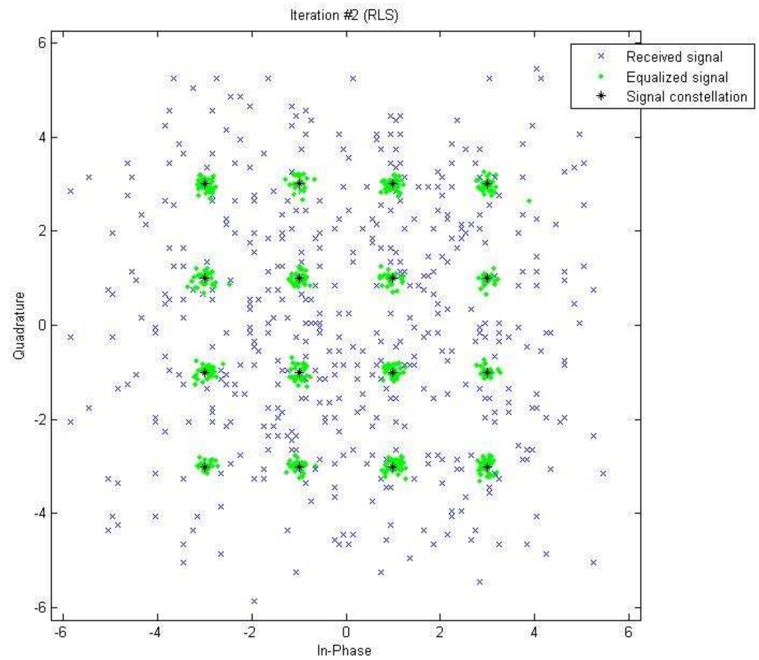
Adaptive channel equalization using LMS algorithm is performed using MATLAB.

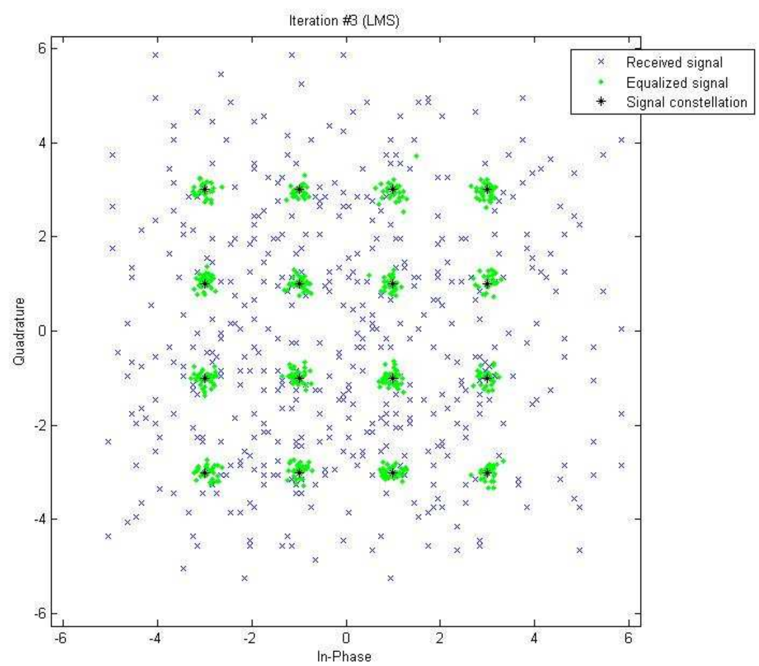
Wave Forms:

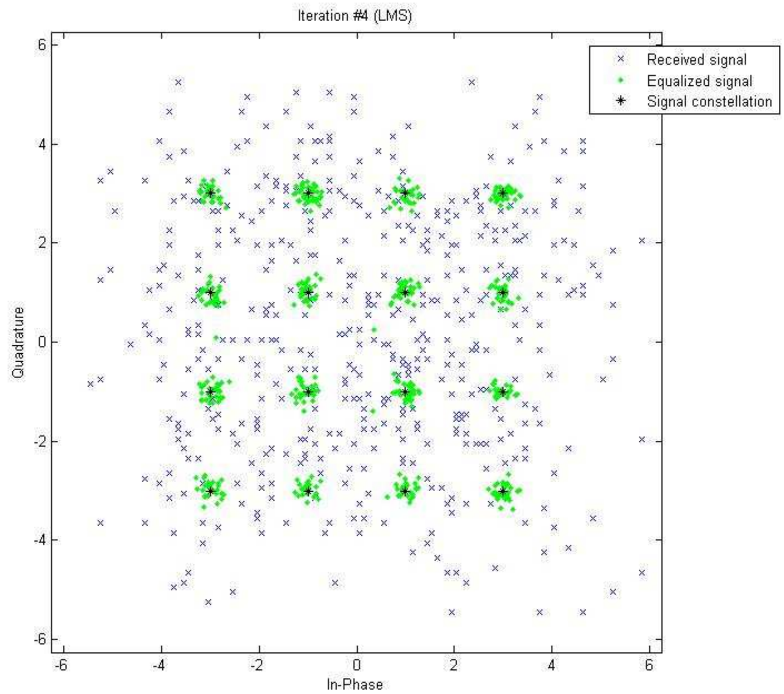


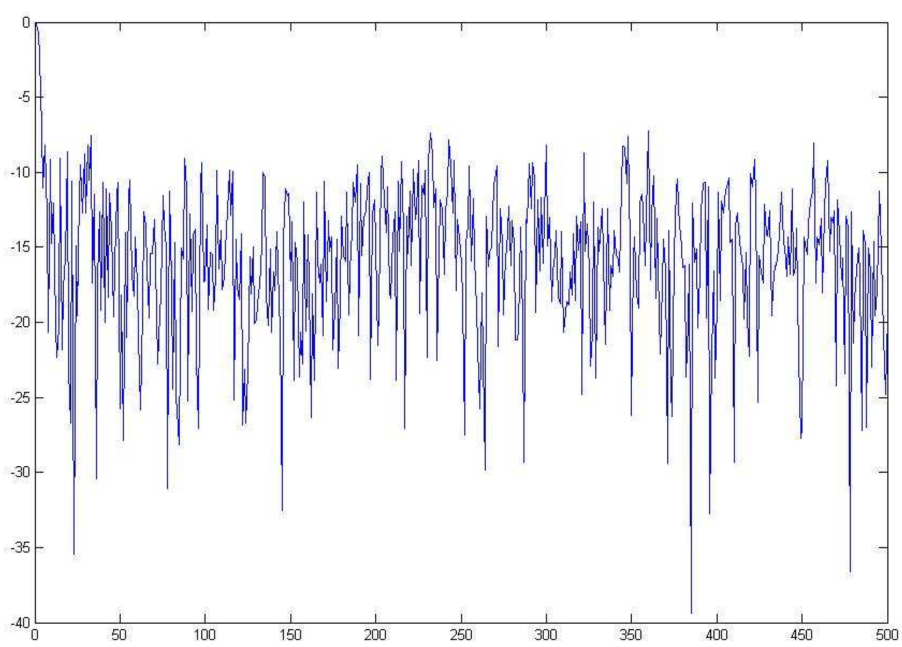

Scatter plot
Filtered signal
Equalized signal
Ideal signal constellation
Quadrature
In-Phase


Iteration #1 (RLS)
Received signal
Equalized signal
Signal constellation
Quadrature
In-Phase


Iteration #2 (RLS)
Received signal
Equalized signal
Signal constellation
Quadrature
In-Phase
6
4
2
0
-2
-4
-6


Iteration #3 (LMS)
Received signal
Equalized signal
Signal constellation
Quadrature
In-Phase
6
4
2
0
-2
-4
-6


Iteration #4 (LMS)
Received signal
Equalized signal
Signal constellation
Quadrature
In-Phase



8. CONVOLUTION CODING AND VITERBI DECODING

AIM:
Write a MATLAB program to Perform Turbo coding and decoding.

PROGRAM:

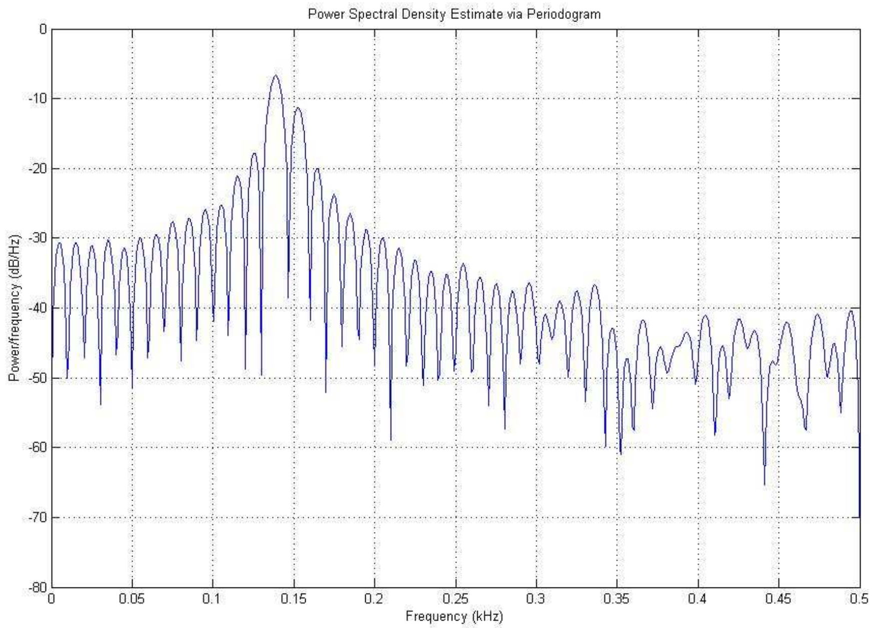
```
%%      Convolutional Coding and Viterbi Decoding Examples

% BSC channel
t = poly2trellis([4 3],[4 5 17;7 4 2]); % Trellis
msg = ones(10000,1); % Data to encode
code = convenc(ones(10000,1),t); % Encode using convolutional code.
[ncode,err] = bsc(code,.01); % Introduce errors in code.
numchanerrs = sum(sum(err)) % Number of channel errors
dcode = vitdec(ncode,t,2,'trunc','hard'); % Decode.
[numsyserrs,ber] = biterr(dcode,msg) % Errors after decoding

%%

t = poly2trellis([4 3],[4 5 17;7 4 2]); % Define trellis.
code = convenc(ones(100,1),t); % Encode a string of ones.
tb = 2; % Traceback length for decoding
decoded = vitdec(code,t,tb,'trunc','hard'); % Decode.

%%

t = poly2trellis(7,[171 133]); % Define trellis.
msg = randint(4000,1,2,139); % Random data
code = convenc(msg,t); % Encode the data.
ncode = awgn(code,6,'measured',244); % Add noise.

% Quantize to prepare for soft-decision decoding.
qcode = quantiz(ncode,[0.001,.1,.3,.5,.7,.9,.999]);
tblen = 48; delay = tblen; % Traceback length
decoded = vitdec(qcode,t,tblen,'cont','soft',3); % Decode.

% Compute bit error rate.
[number,ratio] = biterr(decoded(delay+1:end),msg(1:end-delay));

%%
trel = poly2trellis(3,[6 7]); % Define trellis.
msg = randint(100,1,2,123); % Random data
code = convenc(msg,trel); % Encode.
ncode = rem(code + randerr(200,1,[0 1;.95 .05]),2); % Add noise.
tblen = 3; % Traceback length

decoded1 = vitdec(ncode,trel,tblen,'cont','hard'); %Hard decision

% Use unquantized decisions.
```

```
ucode = 1-2*ncode; % +1 & -1 represent zero & one, respectively.
decoded2 = vitdec(ucode,trel,tblen,'cont','unquant');

% To prepare for soft-decision decoding, map to decision values.
[x,qcode] = quantiz(1-2*ncode,[-.75 -.5 -.25 0 .25 .5 .75],...
[7 6 5 4 3 2 1 0]); % Values in qcode are between 0 and 2^3-1.
decoded3 = vitdec(qcode',trel,tblen,'cont','soft',3);

% Compute bit error rates, using the fact that the decoder output is delayed by tblen
symbols.
[n1,r1] = biterr(decoded1(tblen+1:end),msg(1:end-tblen));
[n2,r2] = biterr(decoded2(tblen+1:end),msg(1:end-tblen));
[n3,r3] = biterr(decoded3(tblen+1:end),msg(1:end-tblen));
disp(['The bit error rates are:   ',num2str([r1 r2 r3])])

%%      Gibb's Phenomenon

% periodic signal x(t)=-1 for t=(-0.5,0) and x(t)=1 for t=(0,0.5), period T=1
% Fourier coeffs=4/(pi*k)

% t = [-500:500] * 0.001;
t1 = [-0.5:0.001:0.5]*3;t2 = [-0.5:0.0001:0.5];t3 = [-0.5:0.00001:0.5];
t=t1; %time scale
total = zeros(size(t));
for k = 1 : 2 : 200
total = total + (4/pi) * sin(2*pi*k*t) / k;
end
plot(t,total);grid minor;

%%      Power Spectrum Estimation

randn('state',0)
fs = 1000;          % Sampling frequency
t = (0:fs/10)/fs;    % One-tenth of a second worth of samples
A = [1 2];          % Sinusoid amplitudes
f = [150;140];       % Sinusoid frequencies
xn = A*sin(2*pi*f*t) + 0.1*randn(size(t)); % sum of 2 freq + noise
Hs = spectrum.periodogram; % define periodogram object
psd(Hs,xn,'Fs',fs,'NFFT',1024) % compute PSD estimate via periodogram
```

RESULT:

Convolution coding and Viterbi decoding is performed using MATLAB.

Wave Forms:



Power Spectral Density Estimate via Periodogram
Power/frequency (dB/Hz)
Frequency (kHz)
0
-10
-20
-30
-40
-50
-60
-70
-80
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5

9. DIGITAL MODULATION AND DEMODULATION

AIM:

Write a MATLAB program to Perform Digital modulation and demodulation.

PROGRAM:

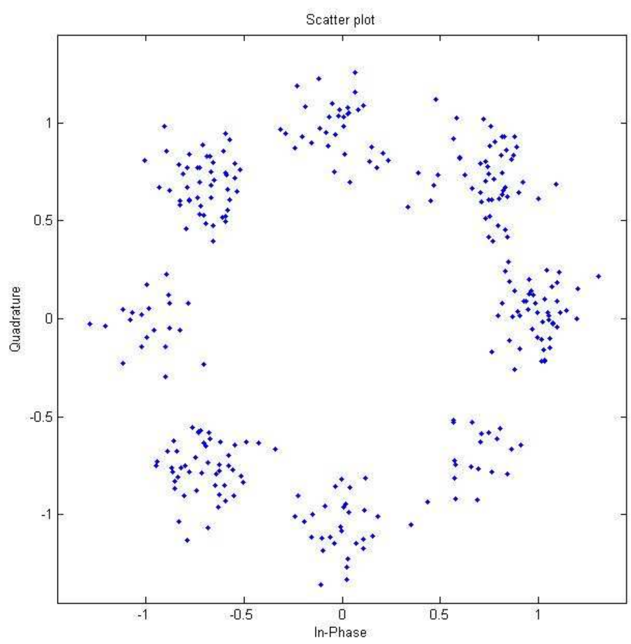
```
%%      Digital Modulation & Demodulation

clc;
clear all;
close all;
M = 8; % Use 16-ary modulation.
Fd = 1; % Assume the original message is sampled
% at a rate of 1 sample per second.
Fs = 3; % The modulated signal will be sampled
% at a rate of 3 samples per second.
x = randint(100,1,M); % Random digital message
% Use M-ary PSK modulation to produce y.
y = dmodce(x,Fd,Fs,'psk',M);
% Add some Gaussian noise.
ynoisy = y + .14*randn(300,1) + .14*j*randn(300,1);
% Create scatter plot from noisy data.
scatterplot(ynoisy,1,0,'b.');
% Demodulate y to recover the message.
z = ddemodce(ynoisy,Fd,Fs,'psk',M);
s = symerr(x,z) % Check symbol error rate.

% Create a random digital message
M = 16; % Alphabet size
x = randint(5000,1,M); % Message signal

% Use 16-QAM modulation.
y = qammod(x,M);

% Transmit signal through an AWGN channel.
ynoisy = awgn(y,15,'measured');

% Create scatter plot from noisy data.
scatterplot(ynoisy);

% Demodulate to recover the message.
z = qamdemod(ynoisy,M);

% Check symbol error rate.
[num,rt]= symerr(x,z)

M = 16; % Alphabet size
x = randint(5000,1,M); % Message signal
```

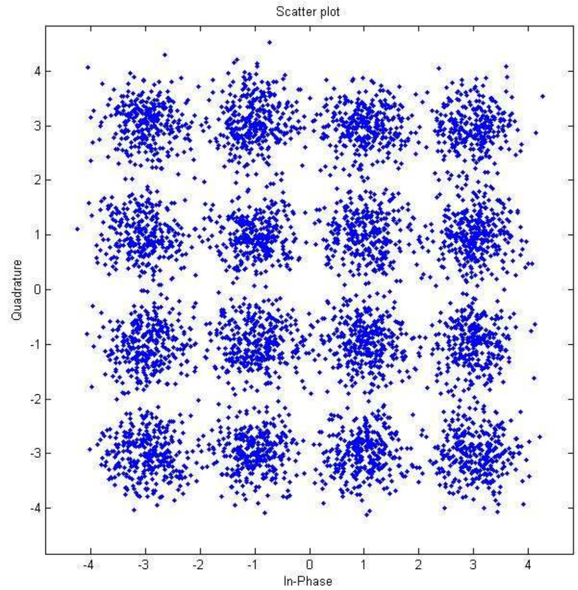
```
Nsamp = 4; % Oversampling rate

% Use 16-QAM modulation.
y = qammod(x,M);

% Follow with rectangular pulse shaping.
ypulse = rectpulse(y,Nsamp);

% Transmit signal through an AWGN channel.
ynoisy = awgn(ypulse,15,'measured');

% Downsample at the receiver.
ydownsamp = intdump(ynoisy,Nsamp);

% Demodulate to recover the message.
z = qamdemod(ydownsamp,M);
scatterplot(x,z);
```

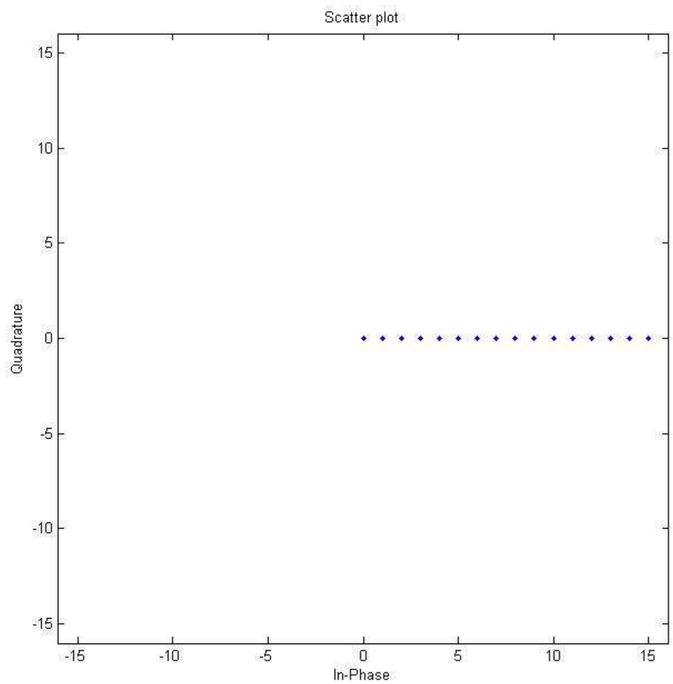
RESULT:

Digital modulation and demodulation is performed using MATLAB.

Wave Forms:




Scatter plot
Quadrature
In-Phase
4
3
2
1
0
-1
-2
-3
-4
-4
-3
-2
-1
0
1
2
3
4


Scatter plot
Quadrature
In-Phase
15
10
5
0
-5
-10
-15
-15
-10
-5
0
5
10
15

10. HUFFMAN SOURCE CODING

AIM:

Write a MATLAB program to Perform Huffman source coding.

PROGRAM:

```
%%      Huffman Source Coding

 %%  Create a Huffman code dictionary using HUFFMANDICT.
 [dict,avglen] = huffmandict([1:6],[.5 .125 .125 .125 .0625 .0625]);
%[dict,avglen] = huffmandict({'x1','x2','x3','x4','x5','x6'},[.5 .125 .125 .125 .0625 .0625]);
         % Define a signal of valid letters.
           sig = [ 2 1 4 2 1 1 5 4 ];
         % Encode the signal using the Huffman code dictionary.
           sig_encoded = huffmanenco(sig,dict);
%codetable = cell2mat(dict);   bits=ceil(log2(17));
          symbols = [1:6] % Alphabet vector
          prob = [.3 .3 .2 .1 .1 0.0] % Symbol probability vector
          [dict,avglen] = huffmandict(symbols,prob)

          prob(prob==0) = [];       % remove zero entries in prob
          H = -sum(prob.*log2(prob));eff1=H/avglen*100;
                    bits=ceil(log2(length(symbols)));eff2=H/bits*100;
           % Pretty print the dictionary.
          temp = dict;
          for i = 1:length(temp)
          temp{i,2} = num2str(temp{i,2});
          end
          temp;
         letters = [1:6]; % Distinct symbols the data source can produce
         p = [.5 .125 .125 .125 .0625 .0625]; % Probability distribution
         [dict,avglen] = huffmandict(letters,p); % Get Huffman code.
         % Pretty print the dictionary.
          codetable = dict;
          for i = 1:length(codetable)
          codetable{i,2} = num2str(codetable{i,2});
          end
         codetable
         sig  = randsrc(1,20,[letters; p]); % Create data using p.
         comp = huffmanenco(sig,dict);  % Encode the data.
         deco = huffmandeco(comp,dict); % Decode the encoded signal.
         equal = isequal(sig,deco); % Check whether the decoding is correct.
```

RESULT:

Huffman source coding is performed using MATLAB.

OUTPUT :

```
symbols =

     1     2     3     4     5     6


prob =

    0.3000    0.3000    0.2000    0.1000    0.1000         0


dict =

    [1]    [1x2 double]
    [2]    [1x2 double]
    [3]    [1x2 double]
    [4]    [1x4 double]
    [5]    [1x3 double]
    [6]    [1x4 double]


avglen =

    2.3000


codetable =

    [1]    '0'
    [2]    '1  0  0'
    [3]    '1  1  1'
    [4]    '1  1  0'
    [5]    '1  0  1  1'
    [6]    '1  0  1  0'
```

11. OVERALL COMMUNICATION SYSTEM

AIM:

Write a MATLAB program to Perform Overall communication system Simulation.

PROGRAM:

```
%%      Overall Communication System Simulation

% This example, described in the Getting Started chapter of the
% Communications Toolbox documentation, aims to solve the following
% problem:
%
% Modify the filtering example (COMMDOC_RRC) so that it
% includes convolutional coding and decoding, given the
% constraint lengths and generator polynomials of the
% convolutional code.

% Copyright 1996-2004 The MathWorks, Inc.
% $Revision: 1.1.6.2 $ $Date: 2004/03/30 13:01:48 $


%%      Setup
% Define parameters.
M = 16; % Size of signal constellation
k = log2(M); % Number of bits per symbol
n = 5e5; % Number of bits to process
nsamp = 4; % Oversampling rate

%%      Signal Source
% Create a binary data stream as a column vector.
x = randint(n,1); % Random binary data stream

% Plot first 40 bits in a stem plot.
stem(x(1:40),'filled');
title('Random Bits');
xlabel('Bit Index'); ylabel('Binary Value');

%%      Channel Encoder
% Define a convolutional coding trellis and use it
% to encode the binary data.
t = poly2trellis([5 4],[23 35 0; 0 5 13]); % Trellis
code = convenc(x,t); % Encode.
coderate = 2/3;

%%      Bit-to-Symbol Mapping
% Convert the bits in x into k-bit symbols, using
% Gray coding.
 % A. Define a vector for mapping bits to symbols using
```

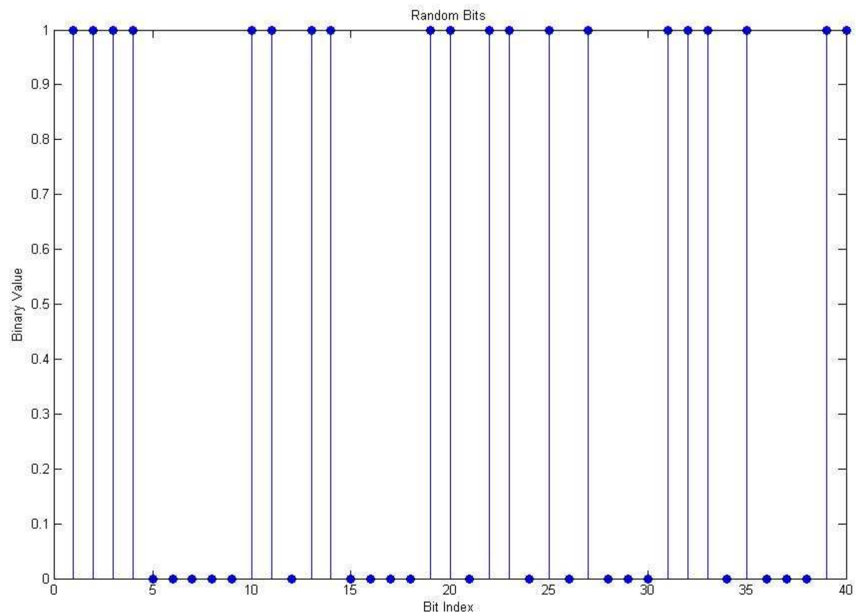
```
% Gray coding. The vector is specific to the arrangement
% of points in a 16-QAM constellation.
mapping = [0 1 3 2 4 5 7 6 12 13 15 14 8 9 11 10].';

% B. Do ordinary binary-to-decimal mapping.
xsym = bi2de(reshape(code,k,length(code)/k).','left-msb');

% C. Map from binary coding to Gray coding.
xsym = mapping(xsym+1);

%%      Stem Plot of Symbols
% Plot first 10 symbols in a stem plot.
figure; % Create new figure window.
stem(xsym(1:10));
title('Random Symbols');
xlabel('Symbol Index'); ylabel('Integer Value');

%%      Modulation
% Modulate using 16-QAM.
y = qammod(xsym,M);

%%      Filter Definition
% Define filter-related parameters.
filtorder = 40; % Filter order
delay = filtorder/(nsamp*2); % Group delay (# of input samples)
rolloff = 0.25; % Rolloff factor of filter

% Create a square root raised cosine filter.
rrcfilter = rcosine(1,nsamp,'fir/sqrt',rolloff,delay);

% Plot impulse response.
figure; impz(rrcfilter,1);

%%      Transmitted Signal
% Upsample and apply square root raised cosine filter.
ytx = rcosflt(y,1,nsamp,'filter',rrcfilter);

% Create eye diagram for part of filtered signal.
eyediagram(ytx(1:2000),nsamp*2);

%%      Channel
% Send signal over an AWGN channel.
EbNo = 10; % In dB
snr = EbNo + 10*log10(k*coderate)-10*log10(nsamp);
ynoisy = awgn(ytx,snr,'measured');

%%      Received Signal
% Filter received signal using square root raised cosine filter.
yrx = rcosflt(ynoisy,1,nsamp,'Fs/filter',rrcfilter);
yrx = downsample(yrx,nsamp); % Downsample.
```

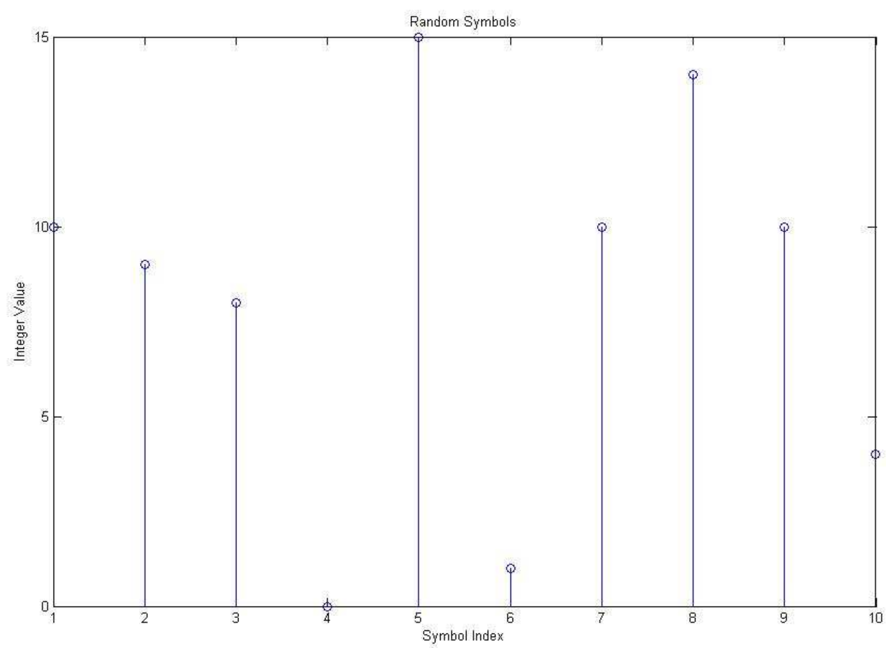
```
yrx = yrx(2*delay+1:end-2*delay); % Account for delay.

%%        Scatter Plot
% Create scatter plot of received signal before and
% after filtering.
h = scatterplot(sqrt(nsamp)*ynoisy(1:nsamp*5e3),nsamp,0,'g.');
hold on;
scatterplot(yrx(1:5e3),1,0,'kx',h);
title('Received Signal, Before and After Filtering');
legend('Before Filtering','After Filtering');
axis([-5 5 -5 5]); % Set axis ranges.

%%        Demodulation
% Demodulate signal using 16-QAM.
zsym = qamdemod(yrx,M);

%%        Symbol-to-Bit Mapping
% Undo the bit-to-symbol mapping performed earlier.

% A. Define a vector that inverts the mapping operation.
[dummy demapping] = sort(mapping);
% Initially, demapping has values between 1 and M.
% Subtract 1 to obtain values between 0 and M-1.
demapping = demapping - 1;

% B. Map between Gray and binary coding.
zsym = demapping(zsym+1);

% C. Do ordinary decimal-to-binary mapping.
z = de2bi(zsym,'left-msb');
% Convert z from a matrix to a vector.
z = reshape(z.',prod(size(z)),1);

%%        Channel Decoder
% Decode the convolutional code.
tb = 16; % Traceback length for decoding
z = vitdec(z,t,tb,'cont','hard'); % Decode.

%%        BER Computation
% Compare x and z to obtain the number of errors and
% the bit error rate. Take the decoding delay into account.
decdelay = 2*tb; % Decoder delay, in bits
[number_of_errors,bit_error_rate] = ...
   biterr(x(1:end-decdelay),z(decdelay+1:end))
```

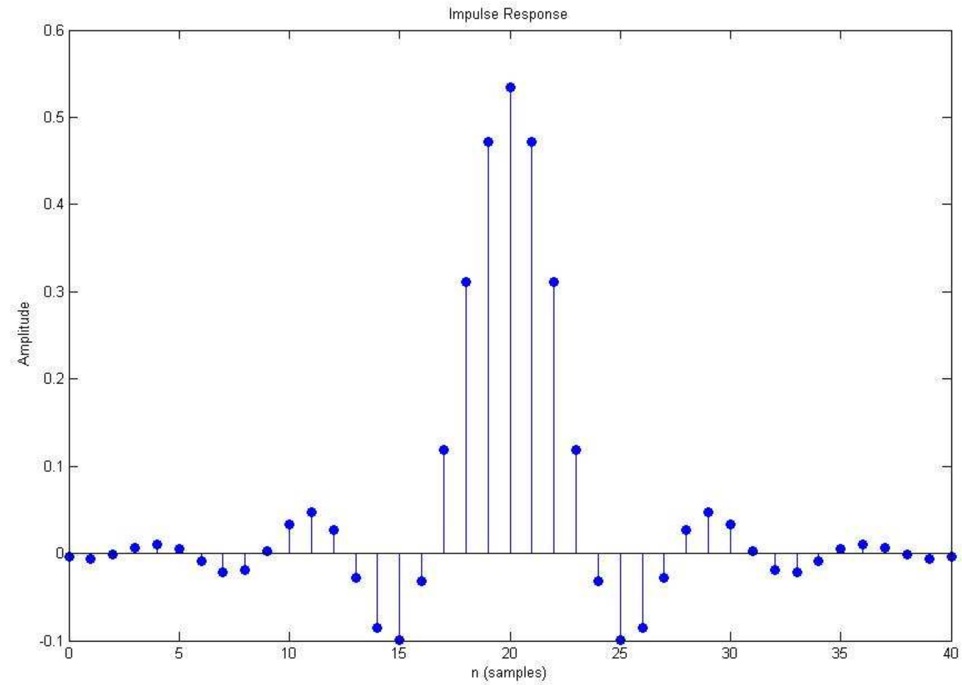
RESULT:

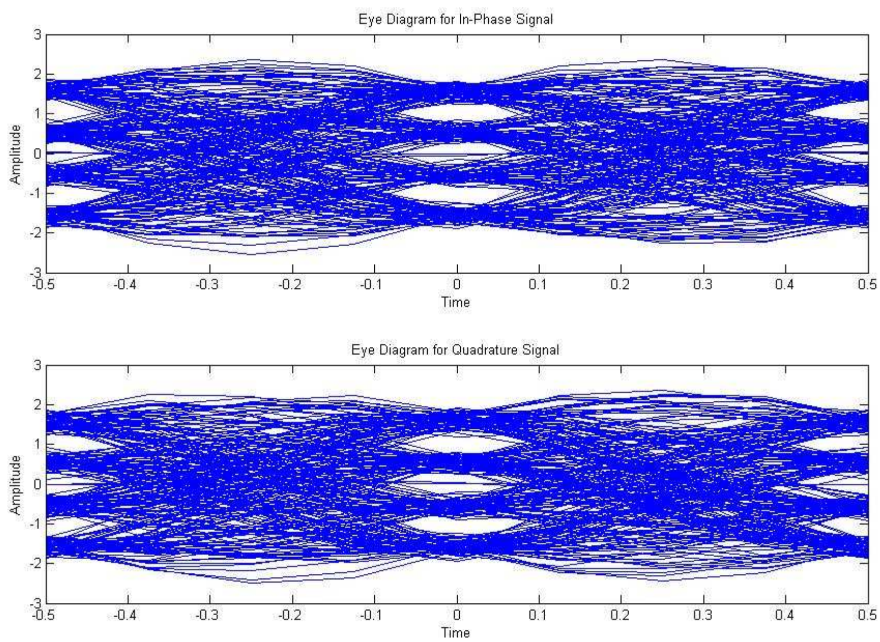
Overall communication system simulation is performed using MATLAB.

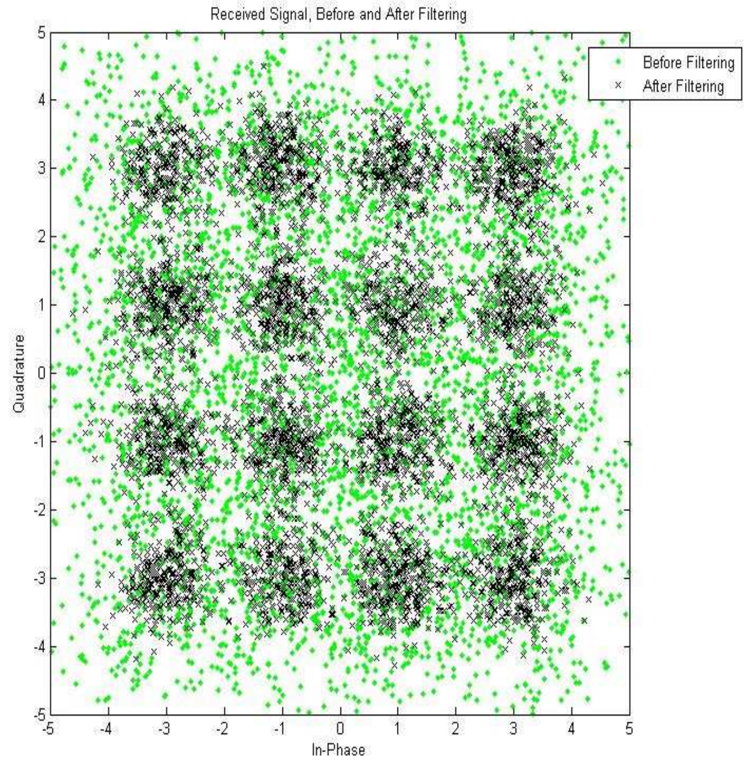
Wave Forms:


Random Bits
Binary Value
Bit Index
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
5
10
15
20
25
30
35
40


Random Symbols
Integer Value
Symbol Index
0
5
10
15
1
2
3
4
5
6
7
8
9
10


Impulse Response
Amplitude
n (samples)
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
0
5
10
15
20
25
30
35
40


Eye Diagram for In-Phase Signal
Amplitude
Time
Eye Diagram for Quadrature Signal
Amplitude
Time
3
2
1
0
-1
-2
-3
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5


Received Signal, Before and After Filtering
Before Filtering
After Filtering
Quadrature
In-Phase
5
4
3
2
1
0
-1
-2
-3
-4
-5
-5
-4
-3
-2
-1
0
1
2
3
4
5

12. FADING CHANNEL SIMULATION

AIM:

Write a MATLAB program to Perform Fading channel Simulation.

PROGRAM:

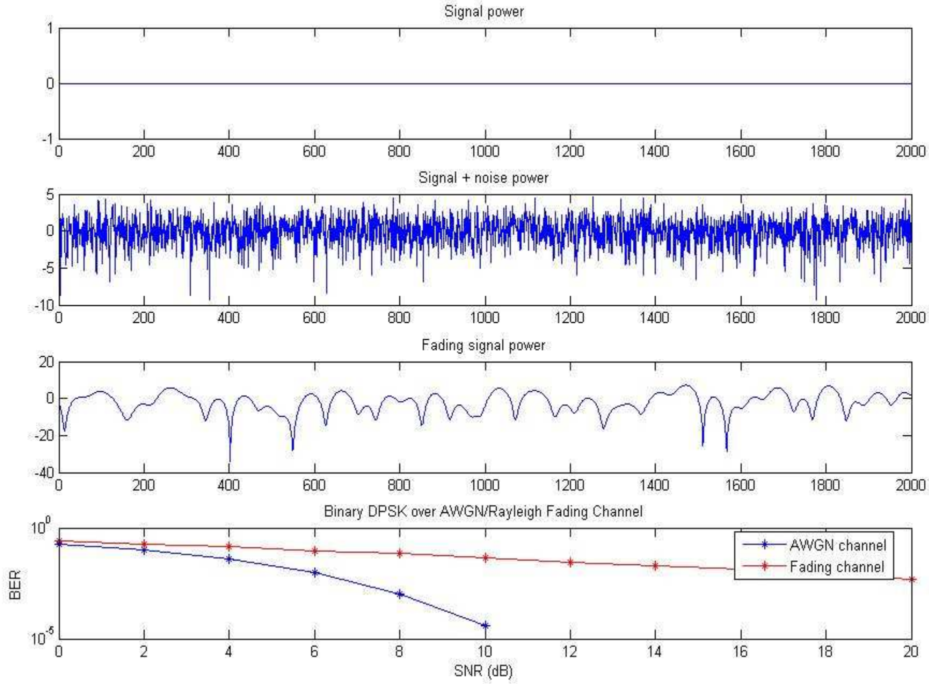
```
%%      Fading Channel Simulation

c = rayleighchan(1/10000,100); % Create channel
c % Display all properties of the channel object.
db = 10; % noise level
sig = j*ones(2000,1); % Signal
% sig = pskmod(randint(1000,1),4); % BPSK symbols

noisysig = awgn(sig,db,'measured'); % add noise
fadingsig = filter(c,sig); % Pass signal through fading channel.
noisyfadingsig = awgn(fadingsig,db,'measured'); % fading + noise
figure,
subplot(4,1,1);plot(20*log10(abs(sig))); % Plot power of signal, versus sample number.
title('Signal power');

subplot(4,1,2);plot(20*log10(abs(noisysig))); % Plot power of noisy-signal
title('Signal + noise power');

subplot(4,1,3);plot(20*log10(abs(fadingsig))); % Plot power of faded signal
title('Fading signal power');

subplot(4,1,4);plot(20*log10(abs(noisyfadingsig))); % Plot power of noisy-fading-signal
title('Fading signal + noise power');

%%      BER of DBPSK over AWGN/frequency-flat Rayleigh fading channel

% Create Rayleigh fading channel object.
chan = rayleighchan(1/10000,100);

% Generate data and apply fading channel.
M = 2; % DBPSK modulation order
tx = randint(50000,1,M);  % Random bit stream
dpskSig = dpskmod(tx,M);  % DPSK signal
fadedSig = filter(chan,dpskSig); % Effect of channel

% Compute error rate for different values of SNR.

SNR = 0:2:20; % Range of SNR values, in dB.

for n = 1:length(SNR)
   rxSig1 = awgn(dpskSig,SNR(n)); % Add Gaussian noise.
   rxSig2 = awgn(fadedSig,SNR(n)); % Add Gaussian noise after fading
```

```
   rx1 = dpskdemod(rxSig1,M); % Demodulate.
   rx2 = dpskdemod(rxSig2,M); % Demodulate.

   % Compute error rate.
   % Ignore first sample because of DPSK initial condition.
   [nErrors1, BER1(n)] = biterr(tx(2:end),rx1(2:end));
   [nErrors2, BER2(n)] = biterr(tx(2:end),rx2(2:end));

end

% Compute theoretical performance results, for comparison.
BERtheory = berfading(SNR,'dpsk',M,1);

% Plot BER results.
semilogy(SNR,BER1,'b*-',SNR,BER2,'r*-');legend('AWGN channel','Fading channel');

% semilogy(SNR,BERtheory,'b-',SNR,BER1,'r*',SNR,BER2,'r*');
% legend('Theoretical BER','Empirical BER');

xlabel('SNR (dB)'); ylabel('BER');
title('Binary DPSK over AWGN/Rayleigh Fading Channel');

%%      Working with Delays

M = 2; % DQPSK modulation order
bitRate = 50000;

% Create Rayleigh fading channel object.
ch = rayleighchan(1/bitRate,4,[0 0.5/bitRate],[0 -10]);
delay = ch.ChannelFilterDelay;

tx = randint(50000,1,M); % Random bit stream
dpskSig = dpskmod(tx,M); % DPSK signal
fadedSig = filter(ch,dpskSig); % Effect of channel
rx = dpskdemod(fadedSig,M); % Demodulated signal

% Compute bit error rate, taking delay into account.
% Remove first sample because of DPSK initial condition.
tx = tx(2:end); rx = rx(2:end);
% Truncate to account for channel delay.
tx_trunc = tx(1:end-delay); rx_trunc = rx(delay+1:end);
[num,ber] = biterr(tx_trunc,rx_trunc) % Bit error rate

%%
```

RESULT:

Fading channel simulation is performed using MATLAB.

Wave Forms:



Signal power
Signal + noise power
Fading signal power
Binary DPSK over AWGN/Rayleigh Fading Channel
AWGN channel
Fading channel
BER
SNR (dB)