CT 303: Digital Communications
Lab 5: Exercises

Name : Dhrumil Mevada **ID :** 201901128

Problem 1:

(a) Write a Matlab function signalx that evaluates the following signal at an arbitrary set of points:

$$x(t) = \begin{cases} 2et+2 & , \quad -3 \le t \le -1 \\ 2e-t \cos 2\pi t & , \quad -1 \le t \le 4 \\ 0 & , \quad \text{else} \end{cases}$$

That is, given an input vector of time points, the function should give an output vector with the values of x evaluated at those time points. For time points falling outside [−3, 4], the function should return the value zero.

(b) Use the function signalx to plot x(t) versus t, for −6 ≤ t ≤ 6. To do this, create a vector of sampling times spaced closely enough to get a smooth plot. Generate a corresponding vector using signalx. Then plot one against the other.

(c) Use the function signalx to plot x (t − 3) versus t.

(d) Use the function signalx to plot x (3 − t) versus t.

(e) Use the function signalx to plot x(2t) versus t.

- **Codes:-**
- %1 a and 1 b

```
syms f(x);
```

```
f(x)= piecewise(-3 < x <-1, 2*exp(x+2),-1< x <4, 2*exp(-x)*cos(2*pi*x) , 0);
y=linspace(-6,6,1000);
answer_of_b = f(y);
figure;
plot(y,answer_of_b);
title(' x(t) - t ');
xlabel('Time (t)');
ylabel('x(t)');
grid on;
```

- %1 c

```
y=linspace(-2,10,1000);
answer_c = f(y-3);
figure;
plot(y,answer_c);
title(' x(t-3) - t ');
xlabel('Time(t)');
ylabel('x(t-3)');
grid on;
```

- %1 d

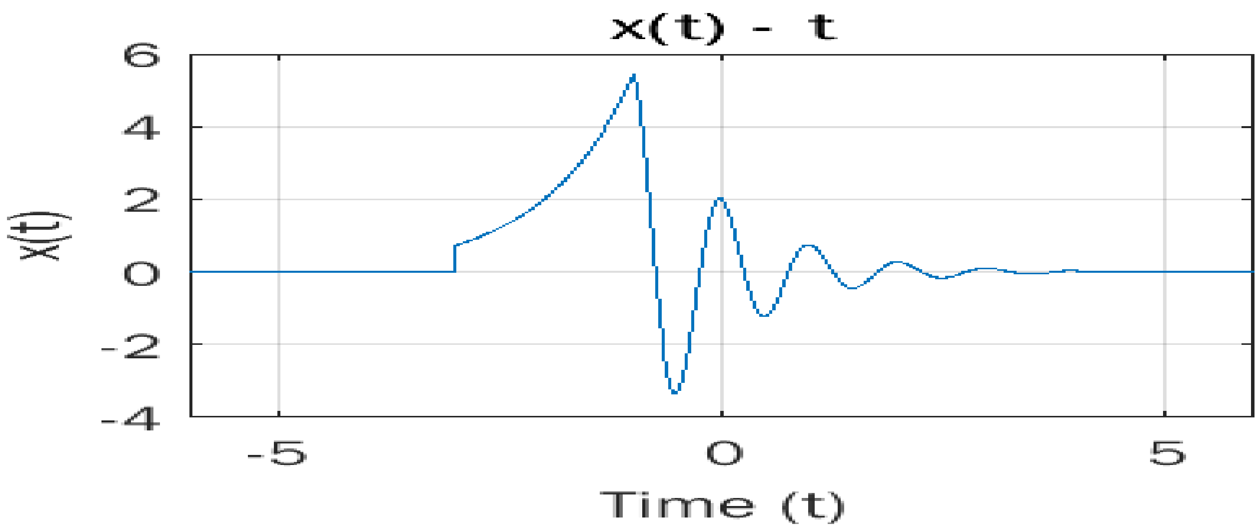
```
y=linspace(-5,10,1000);
answer_d = f(3-y);
figure;
plot(y,answer_d);
title(' x(3-t) - t ');
xlabel('Time(t)');
ylabel('x(3-t)');
grid on;
```

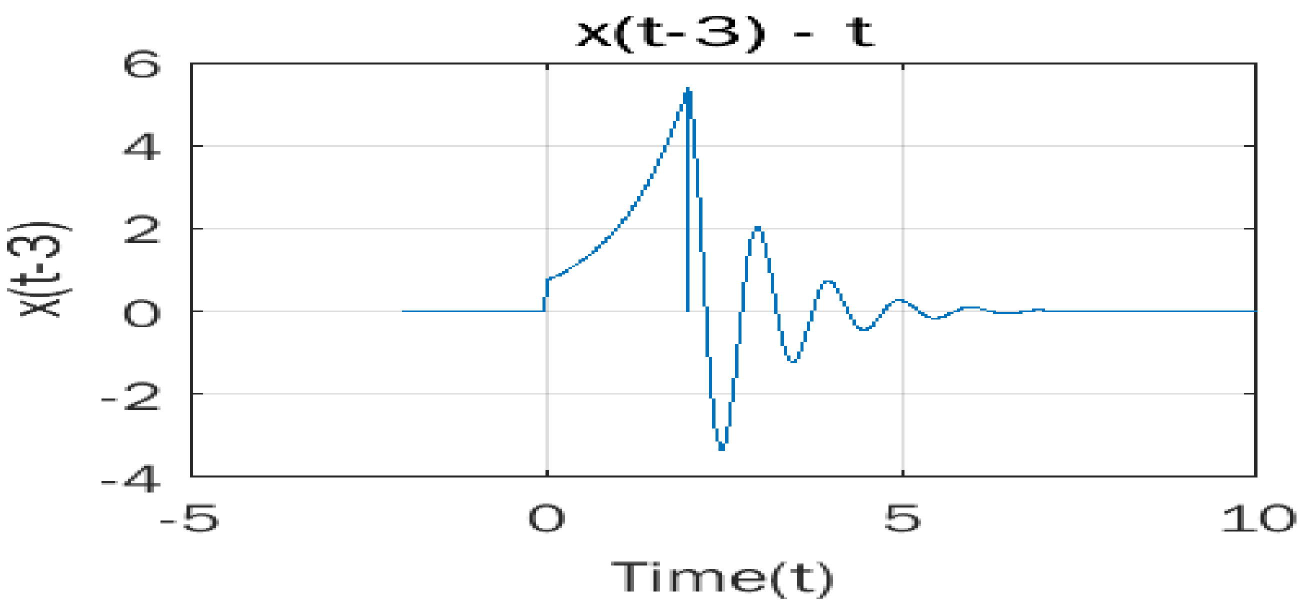
- %1 e

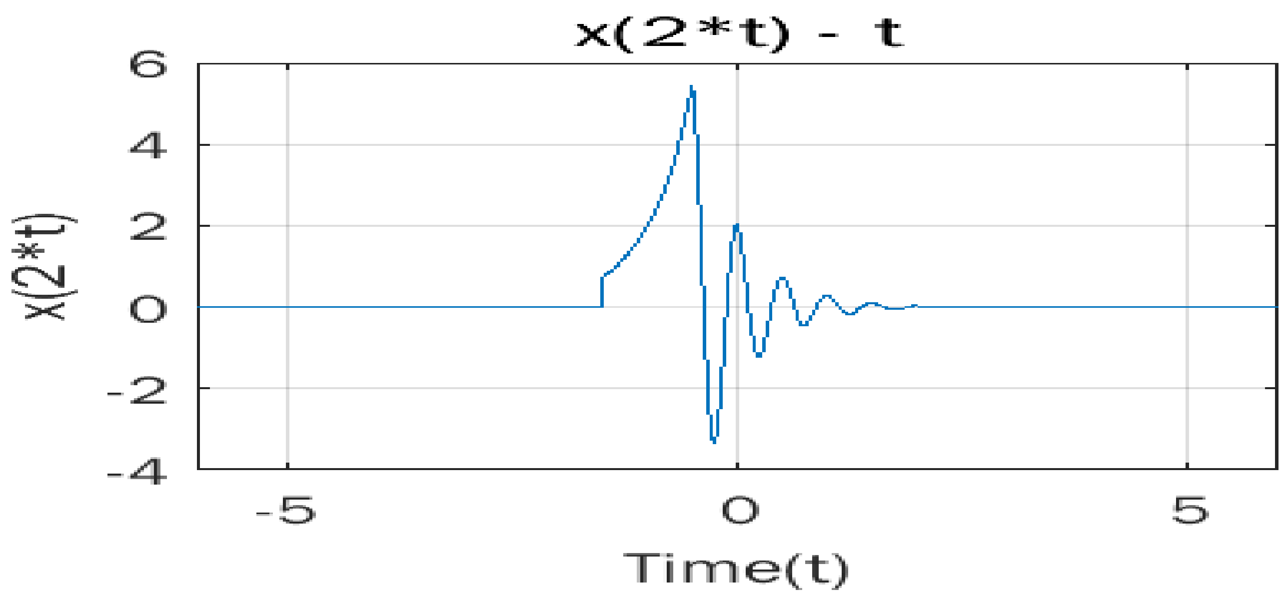
```
y=linspace(-6,6,1000);
answer_e = f(2*y);
figure;
plot(y,answer_e);
title(' x(2*t) - t ');
xlabel('Time(t)');
```

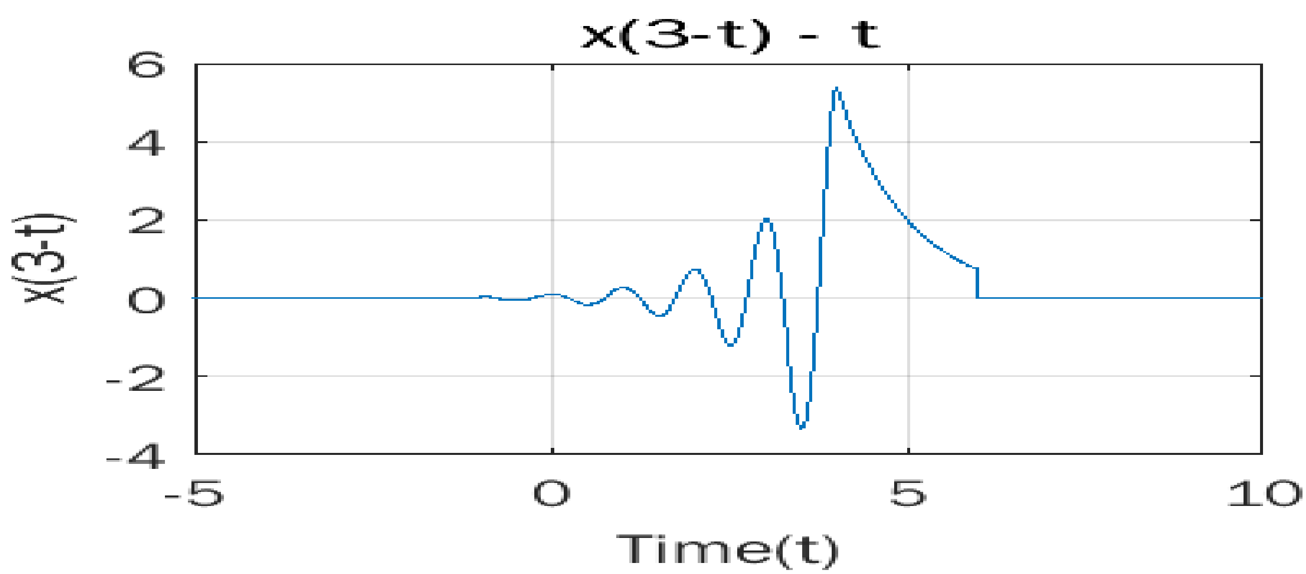
```
ylabel('x(2*t)');
grid on;
```

Plots:









Observations :

1. **1.** 1st pic -3 to -1 it increases exponentially, and then from -1 to 4, it behaves as an exponentially decreasing sinusoidal wave.
2. **2.** 2nd x(t-3) is right shifted x(t) by t=3.
3. **3.** x(3-t) is also a shifted version of x(t). First, x(t) is time-reversed and then shifted rightwards by t=3.
4. **4.** x(2t) is the compressed version of x(t) by 2 times.

Problem 2:

Suppose that we want to compute the Fourier transform of the sine pulse u(t) = sin πtI[0,1](t). The Fourier transform for this can be computed analytically to be

U(f) = (2 cos πf /π(1 − 4f^(2)) * e−jπf (2.63)

Note that U(f) has a 0/0 form at f = 1/2, but using L'Hospital's rule, we can show that U(1/2) != 0. Thus, the first zeros of U(f) are at f = ±3/2. This is a timelimited pulse and hence cannot be bandlimited, but U(f) decays as 1/f2 for f large, so we can capture most of the energy of the pulse within a suitably chosen finite frequency interval. Let us use the DFT to compute U(f) over f ∈ (−8, 8). This means that we set 1/(2ts) = 8, or ts = 1/16, which yields about 16 samples over the interval [0, 1] over which the signal u(t) has support. Suppose now that we want the frequency granularity to be at least fs = 1/160. Then we must use a DFT with N ≥ 1/tsfs = 2560 = Nmin. In order to efficiently compute the DFT using the FFT, we choose N = 4096, the next power of 2 at least as large as Nmin.

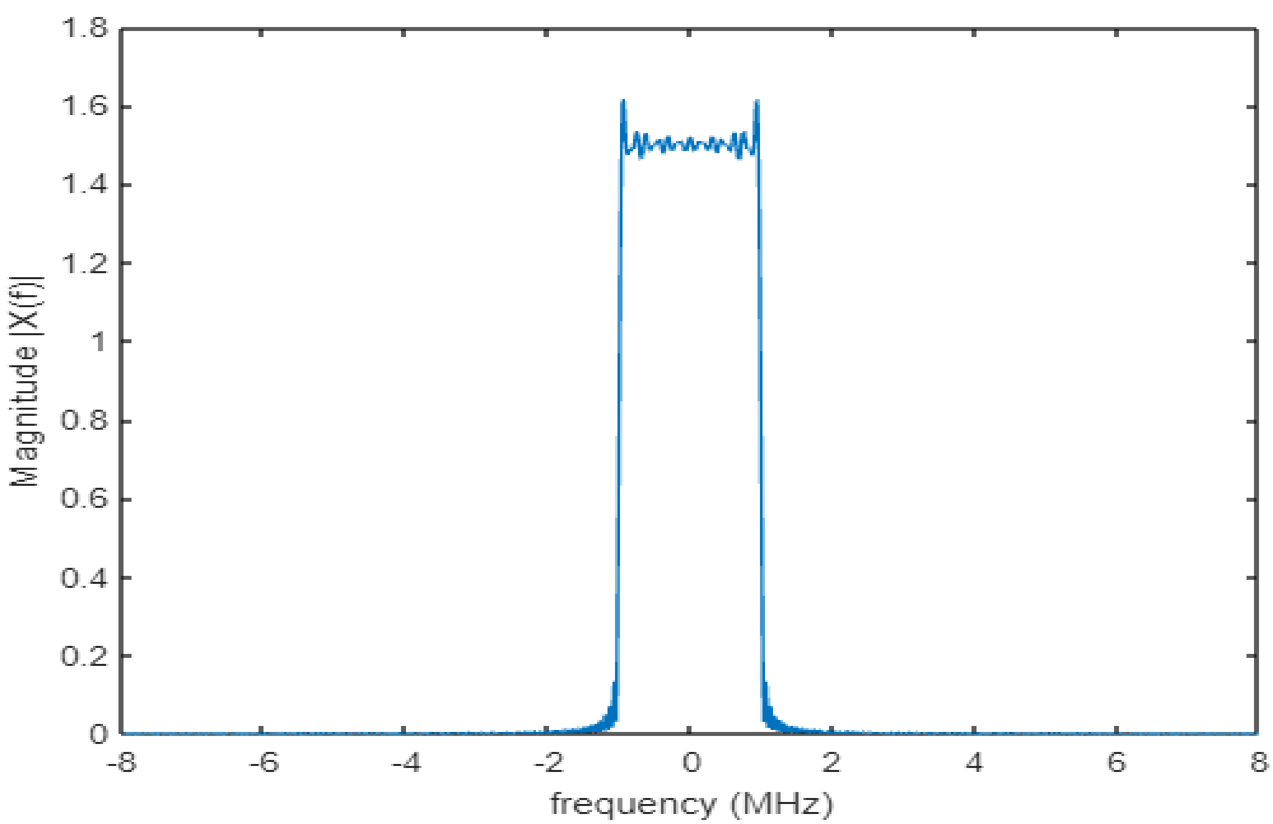
Code :-

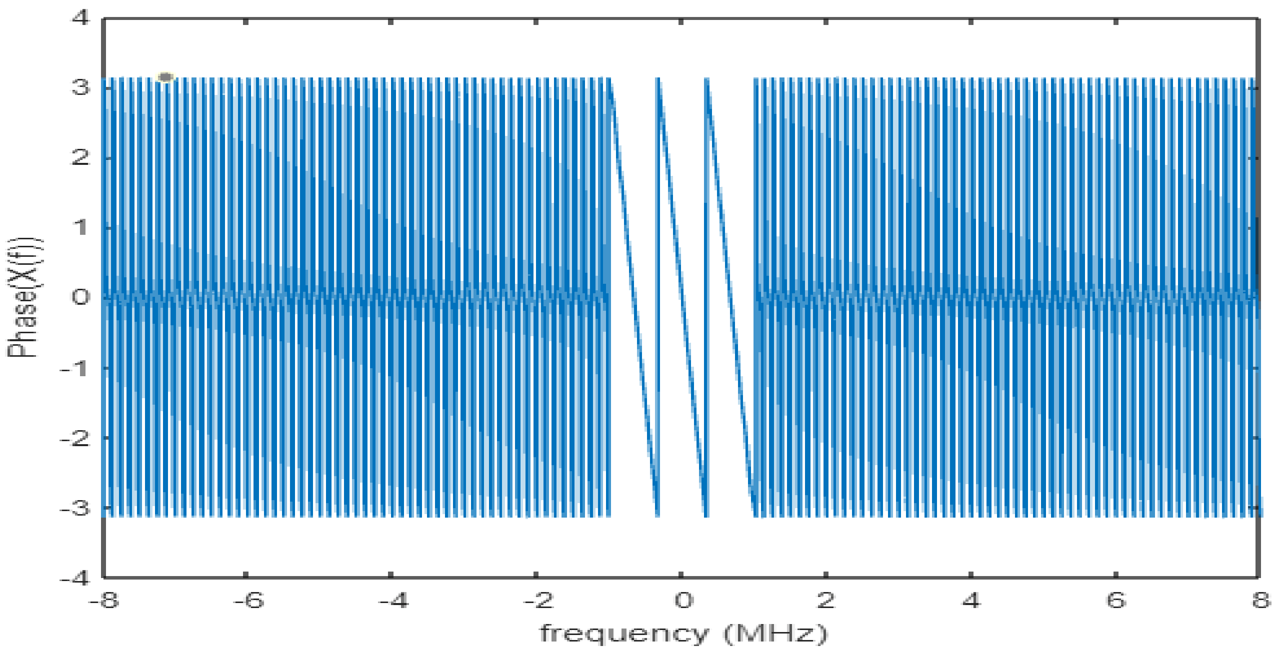
```
clear ;
close all;
clear vars;
t = -8:1/16:8;
s = 3*sinc(2*t-3);
[X,f,df] = contFT(s,-8,1/16,1e-3);
figure(1);
title('Magnitude Response of s(t)');
plot(f,abs(X));
ylabel('Magnitude |X(f)|');
xlabel('frequency (MHz)');
figure(2);
title('Phase Response of s(t)');
plot(f,angle(X));
ylabel('Phase(X(f))');
xlabel('frequency (MHz)');
```

Function:-

```
function [X,f,df] = contFT(x,tstart,dt,df_desired)
Nmin=max(ceil(1/(df_desired*dt)),length(x));
%choose FFT size to be the next power of 2
Nfft = 2^(nextpow2(Nmin));
%compute Fourier transform, centering around DC
X=dt*fftshift(fft(x,Nfft));
%achieved frequency resolution
df=1/(Nfft*dt);
%range of frequencies covered
f = ((0:Nfft-1)-Nfft/2)*df; %same as f=-1/(2*dt):df:1/(2*dt) - df
%phase shift associated with start time
X=X.*exp(-1i*2*pi*f*tstart);
end
```

Plots:





Observations: -

1. 1st pic shows the magnitude response of 3*sinc(2t-3). It is a rectangular pulse. However, due to the sampling and truncation of the time domain sinc pulse, the rectangular pulse has some ripples at the corner. This effect is known as **Gibbs Phenomenon**.
2. 2nd pic shows the Phase response of 3*sinc(2t-3). The Phase plot has a meaning in the interval f = [-1,1].

Problem 3 :

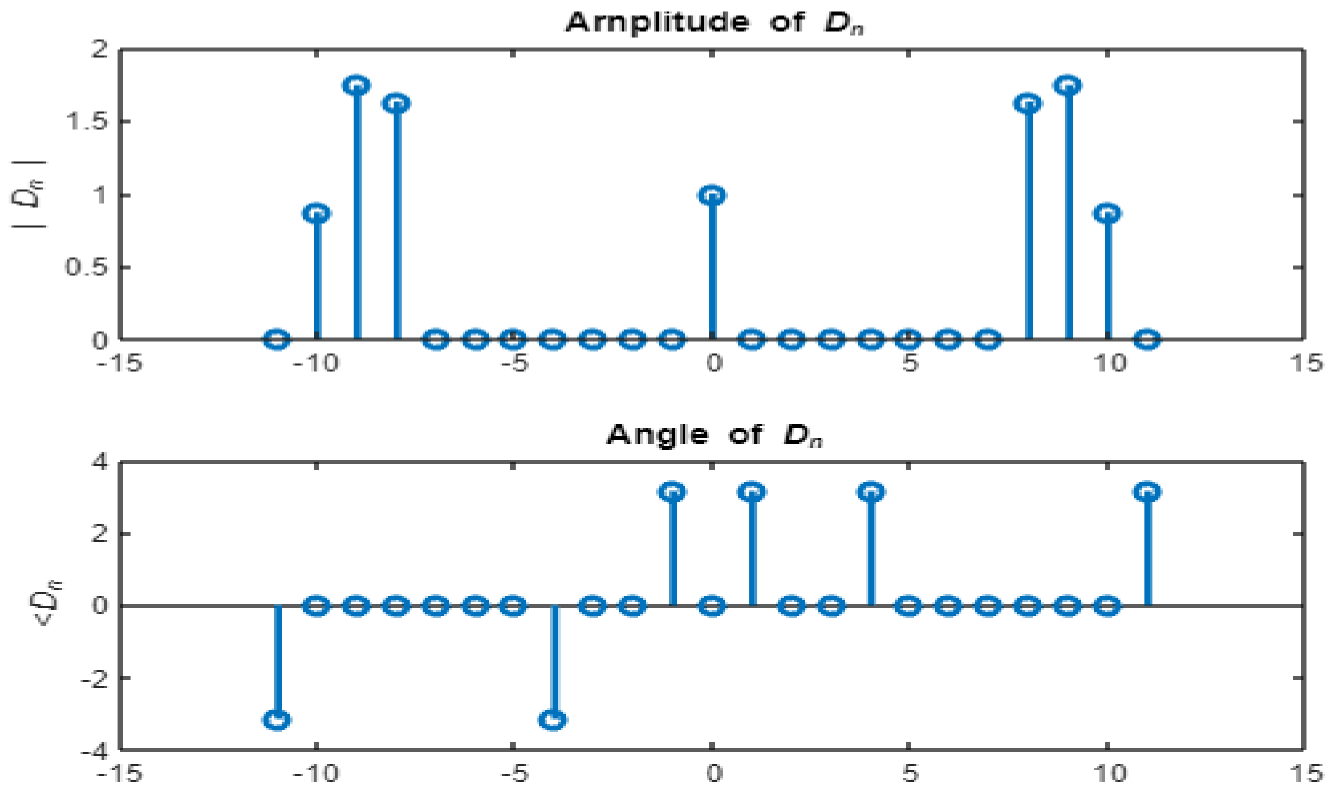
Find the exponential Fourier series for the periodic square wave w(t).

Code :-

```
clc;
clear ;
close all;
clear vars;
```

```
j = sqrt(-1);          % Define j for complex algebra
b = 2 ; a = -2;         %determine one signal period
tol = exp(-5);          % set integration error tolerance
T = b-a;               % length of the period
N = 11;                %Number of FS coefficients on each side of
zero frequency
Fi = (-N:N)*2*pi/T ;    % Set frequency range

% now calculate D_O and store it in D(N+l) ;
Func = @(t) funct_sqr(t/2) ;
D(N+1) = (1/T)*(quad(Func,a,b,tol));      % Using quad.m
integration

for i = 1 : N

%Calculate Dn for n=l,...,N(stored in D(N+2)...D(2N+l)
Func = @(t)exp(-j*2*pi*t*i/T).*funct_sqr(t/2);
D(i+N+1)=quad(Func,a,b,tol);

% Calculate Dn for n=-N,...,-1(stored in D(l)...D(N)
Func=@(t)exp(j*2*pi*t*(N+1-i)/T).*funct_sqr(t/2);
D(i)=quad(Func,a,b,tol);

end

figure (1) ;
subplot(211);
s1 = stem((-N:N),abs(D));
set(s1,'Linewidth',2);
ylabel('| {\itD}_{\itn} |');
title('Arnplitude of {\itD}_{\itn}');

subplot(212);
s2 = stem((-N:N), angle(D));
set(s2 ,'Linewidth',2) ; ylabel('<{\itD}_{\itn}');
title('Angle of {\itD}_{\itn}');
```

Plots :



Observations :-

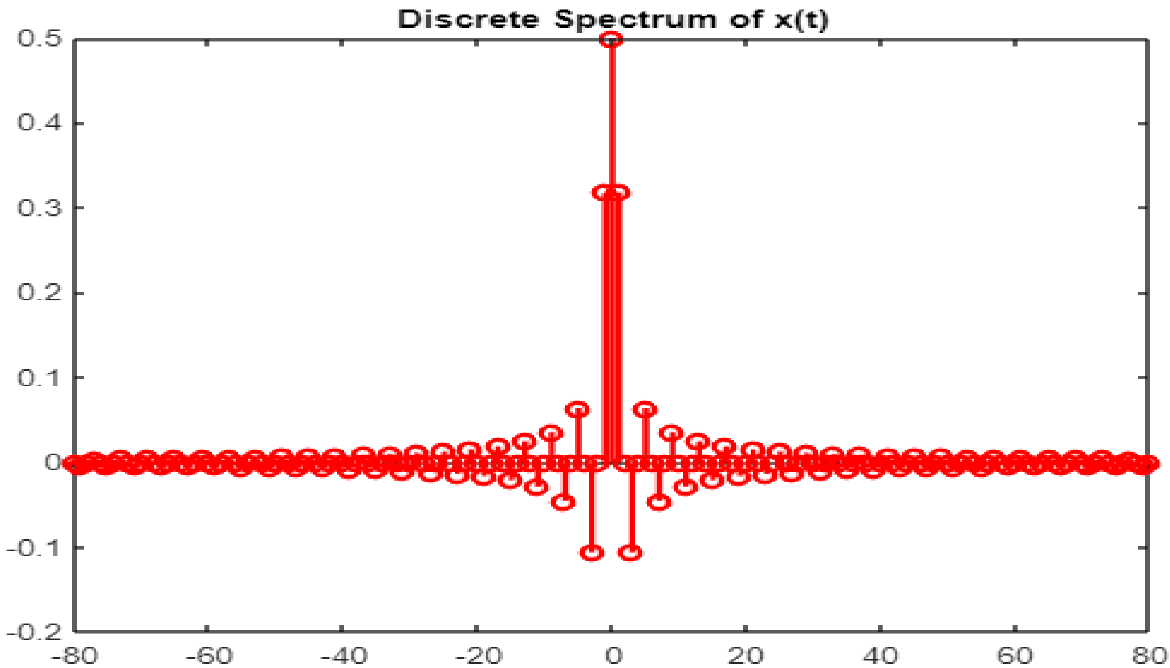
1. 1st pic shows the Fourier Series coefficients of Square Wave. The magnitude Spectrum of Dn is symmetric for a Square Pulse. Phase Spectrum is additive inverse on both the sides of origin for a Square Pulse.
2. 2nd pic shows the Fourier Series coefficients of a Triangular Wave. Both the Phase and Magnitude of Dn are symmetric about the origin of a triangular Pulse.

Problem 4 :

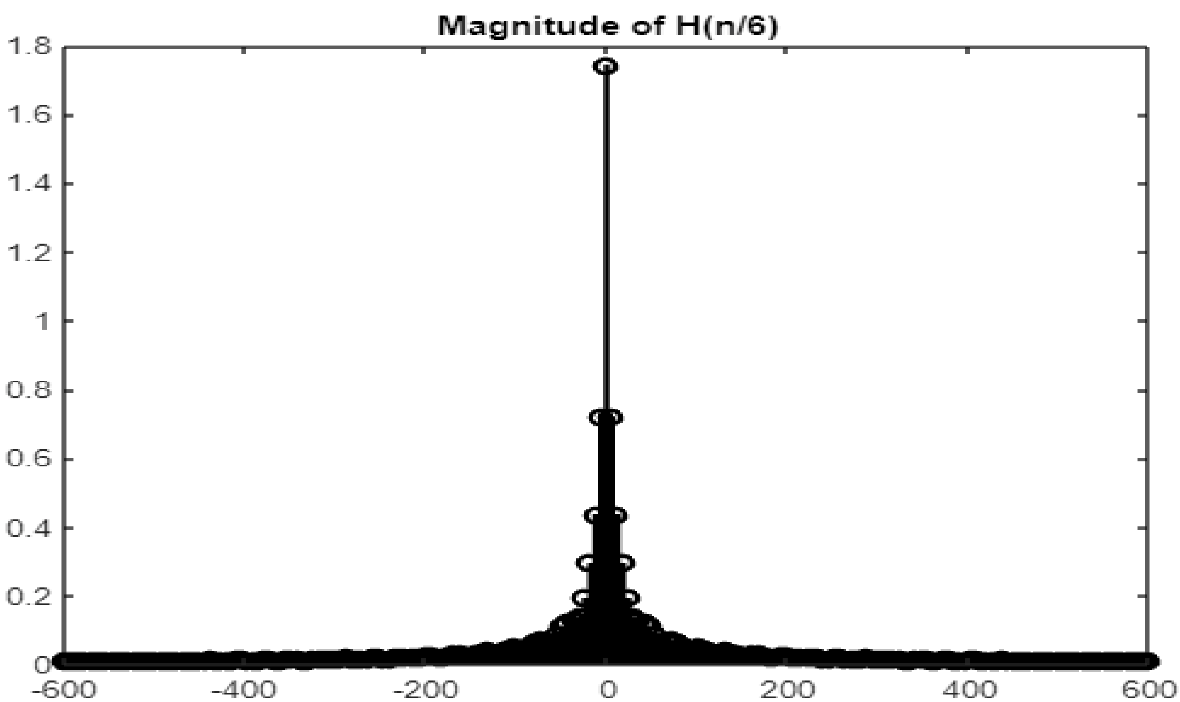
Code :-

```
clc; clear; close all; clear vars;
n = [-80:80];
%Fourier Series coefficient of x(t)
x=(1/2)*(sinc(n/2));
%Sampling Interval
ts = 1/40; %time vector
t = [-0.5:ts:4.5]; h_t = exp(-t(21:181)./2);
%Impulse Response
fs = 1/ts; h = [zeros(1,20) h_t zeros(1,20)];
%Transfer Function
H = fft(h)/fs;
%Frequency Resolution
df = fs/80;
f = [0:df:fs] - fs/2;
%Rearrange H
H1 = fftshift(H); y = x.*H1(21:181);

figure(1);
stem(n,x,'-o','linewidth',1.75,'color','red');
title('Discrete Spectrum of x(t)');

figure(2);
stem((-600:6:600),abs(H1),'o','linewidth',1.75,'color','black');
title('Magnitude of H(n/6)');

figure(3);
stem(n,abs(y),'-o','linewidth',1.75,'color','blue');
title('Discrete Spectrum of Output');
```

Plots:



<u>Figure 1:Fourier series co-efficient of message signal</u>



<u>Figure 2:Fourier series co-efficient H(n/6)</u>

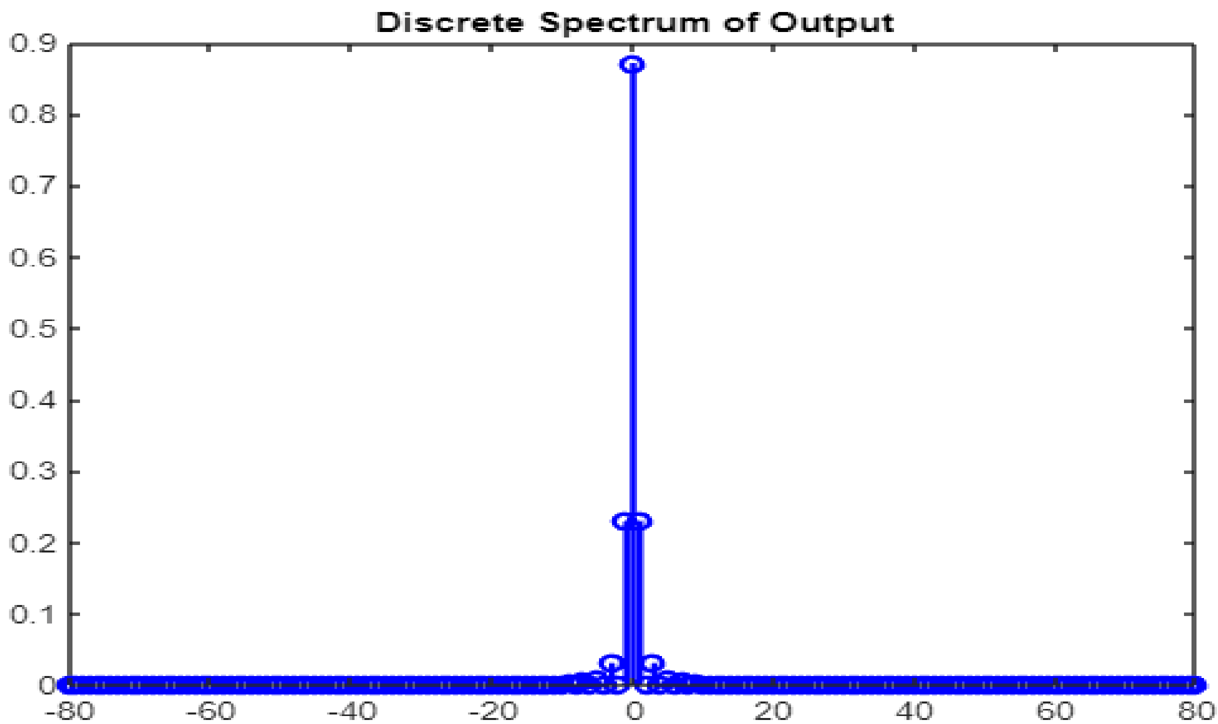


Figure 3:Fourier series co-efficient of output signal

Observations :-

1. Xn = (½) x sinc(n/2) which is plotted in figure 1.
2. H(n/T), where T = 6; is plotted in figure 2.
3. Yn = (½) x sinc(n/2) x H(n/6) which is plotted in figure 3.

Lab 6

Problem 1:-

a) 4.13 The periodic signal x () has a period of 2 and in the interval [0, 2] is defined as

$$x(t) = \begin{cases} t & 0 <= t < 1 \\ -t + 2 & 1 <= t < 2 \end{cases}$$

a. Design an 8-level uniform PCM quantizer for this signal and plot the quantized output of this system.
b. Plot the quantization error for this system.
c. By calculating the power in the error signal, determine the SQNR for this system in decibels.
d. Repeat parts a, b, and c using a 16-level uniform PCM system.

- **Function**:

```
function [sqnr,a_quan,code]=u_pcm(a,n)

amax=max(abs(a));
a_quan=a/amax;
b_quan=a_quan;
d=2/n;
q=d.*(0:n-1);
q=q-((n-1)/2)*d;

for i=1:n
a_quan(find((q(i)-d/2 <= a_quan) & (a_quan <= q(i)+d/2)))=...
q(i).*ones(1,length(find((q(i)-d/2 <= a_quan) & (a_quan <=
q(i)+d/2))));
b_quan(find( a_quan==q(i) ))=(i-1) *ones(1,length(find(
a_quan==q(i))));
end
a_quan=a_quan*amax;
```

```
nu=ceil(log2(n));
code=zeros(length(a),nu);
for i=1:length(a)
for j=nu:-1:0
if (fix(b_quan(i)/(2^j))==1)
code(i,(nu-j)) = 1;
b_quan(i) = b_quan(i) - 2^j;
end
end
end
sqnr=20*log10(norm(a)/norm(a-a_quan));
```

❖ Code:-

```
%for 8 level and 16 level quantized output:
% MATLAB script for Illustruive Problem 9. Chapter +
clear;
clc;
echo on

t = 0:0.01:2;
yval = zeros(1,length(t));

f1 = @(x)(x); %Function
f2 = @(x)(-x+2);

for i = 1:length(t)
    p = t(i);
    if( p >= 0) && ( p < 1)
        yval(i) = f1(p);
    elseif ( p >= 1) && ( p < 2)
        yval(i) = f2(p);
    else
        yval(i) = 0;
    end
end
a = yval;
```

```
[sqnr8,aquan8,code8]=u_pcm(a,8);
[sqnr16,aquan16,code16]=u_pcm(a,16);
%Press a key to see the SQNR for N = 8
%pause
sqnr8
%pause
% Press a key to see the SONR for N = 16
sqnr16
% Press a key to see the plot of the signal and its quantized
versions
%pause

figure;
plot(t,a,t,aquan8,'k-','linewidth',0.8);
legend('Original function', '8 level PCM Quantized
output','Location','south');
xlabel('Time (t)');
ylabel('f(t) and f^{-}(t)');
title('8 level quantized output');

grid on;
figure;
plot(t,a,t,aquan16,'k-','linewidth',0.8);
legend('Original function', '16 level PCM Quantized
output','Location','south');
xlabel('Time (t)');
ylabel('f(t) and f^{-}(t)');
title('16 level quantized output');
grid on;

figure;
plot(t, (a-aquan8), 'k-','linewidth',0.8);
xlabel('Time (t)');
ylabel('Quantization Error for 8 level');
grid on;

figure;
plot(t, (a-aquan16), 'k-','linewidth',0.8);
xlabel('Time (t)');
```

```
ylabel('Quantization Error for 16 level');
grid on;

fprintf('SQNR for 8-level Quantization %f\n\n', (sqnr8));
fprintf('SQNR for 16-level Quantization %f\n\n', (sqnr16));
```

- For 8 level quantizer the number of bits is n = 3.
- So according to the known formula of SQNR,

  SQNR = (6)*(n) + 1.8

- So, for the 8 level quantizer calculated SQNR is 19.8 and observed SQNR is 17.9837.
- Similarly for the 16 level quantizer the number of bits is n=4.

So, for the 16 level quantizer calculated SQNR is 25.8 and observed SQNR is 24.0043.

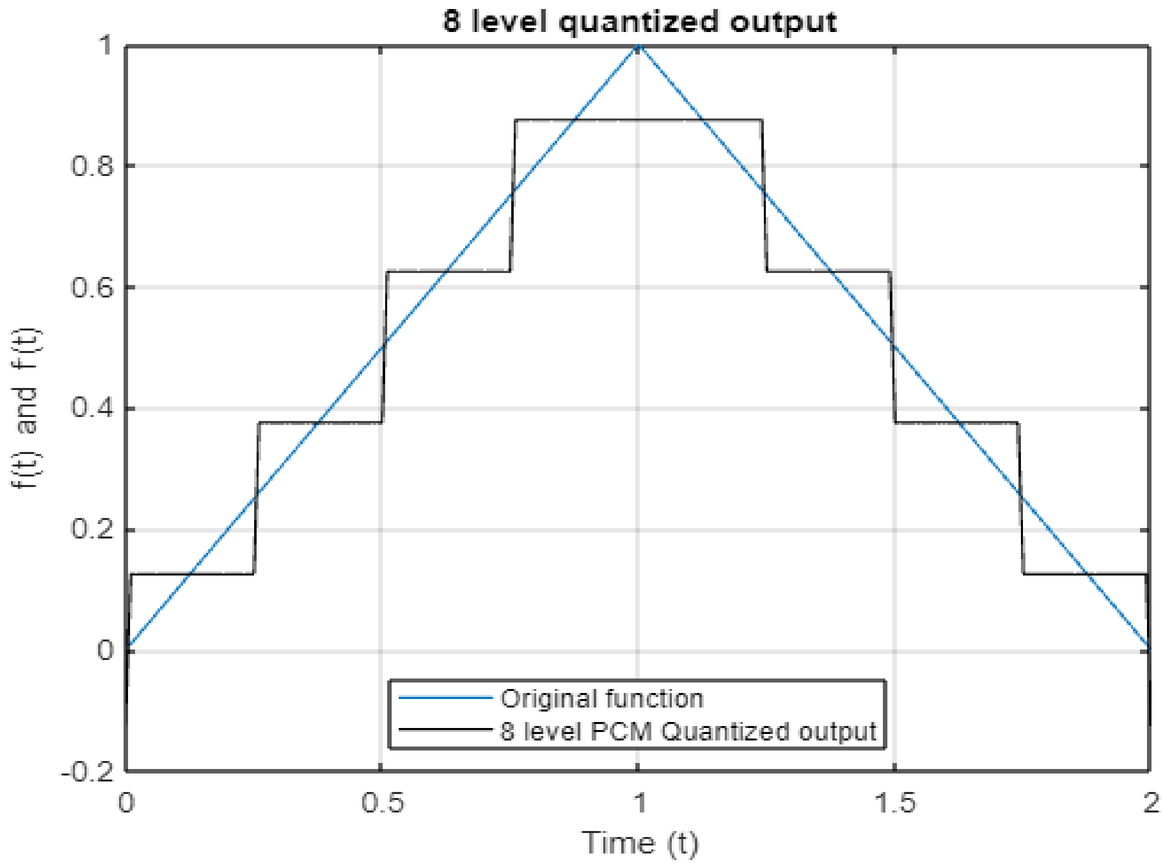


Figure 1:8 level Quantized Output

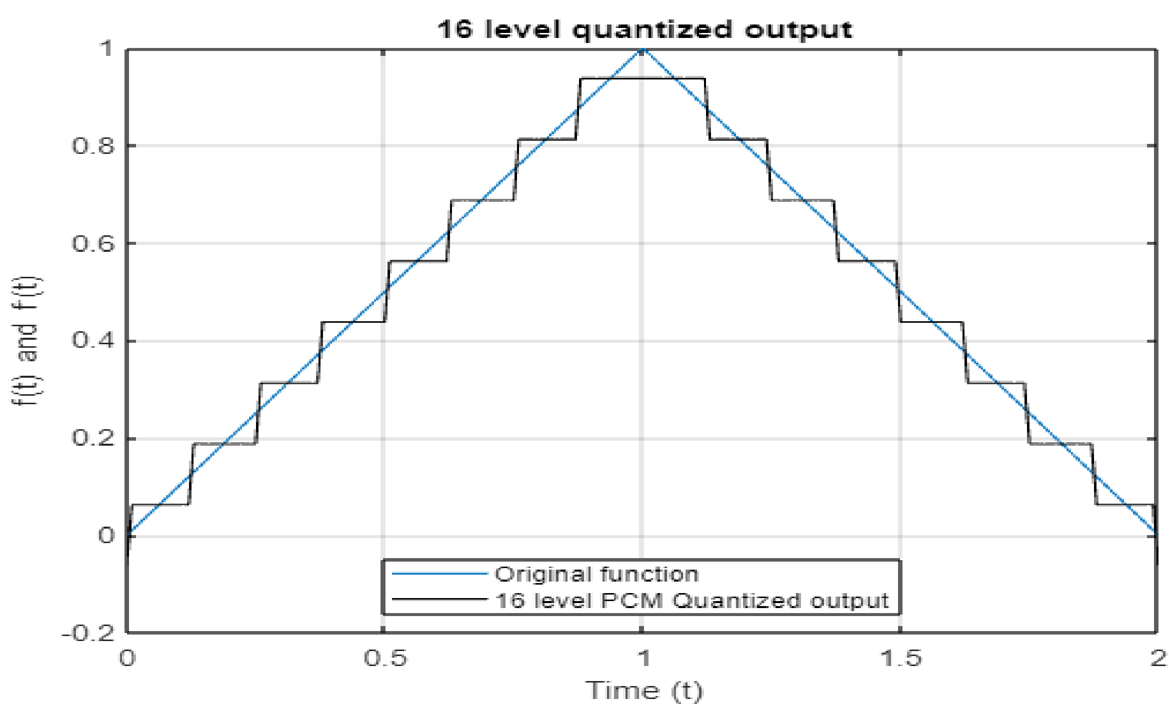


Figure 2:16 level Quantized Output

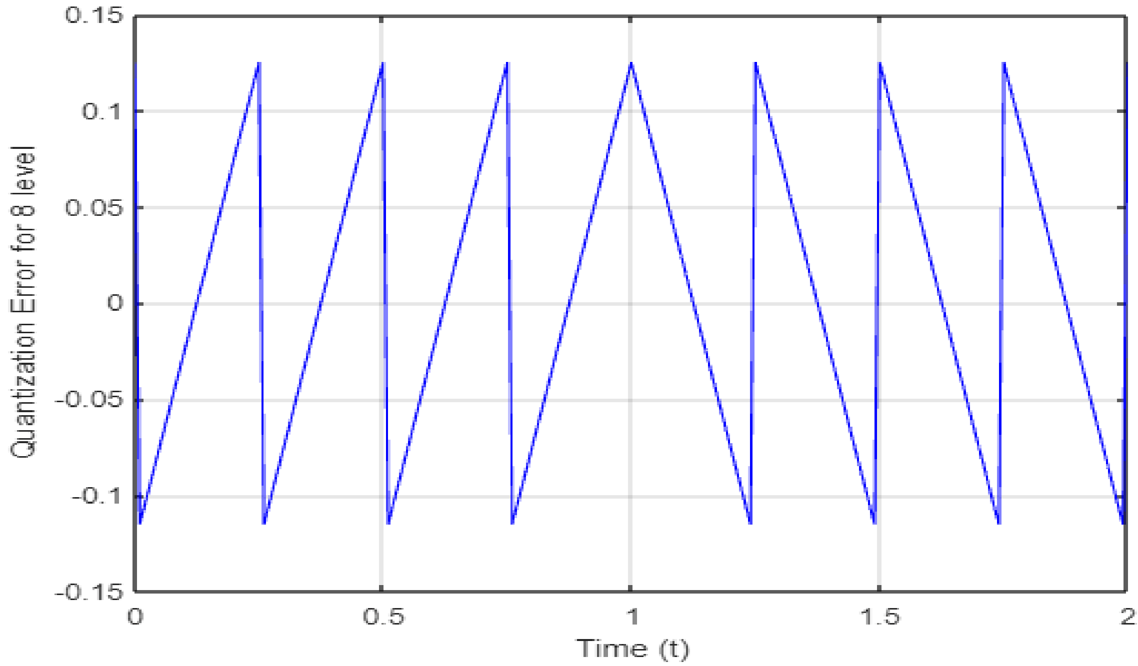


Figure 3:Quantized Error Of 8 level

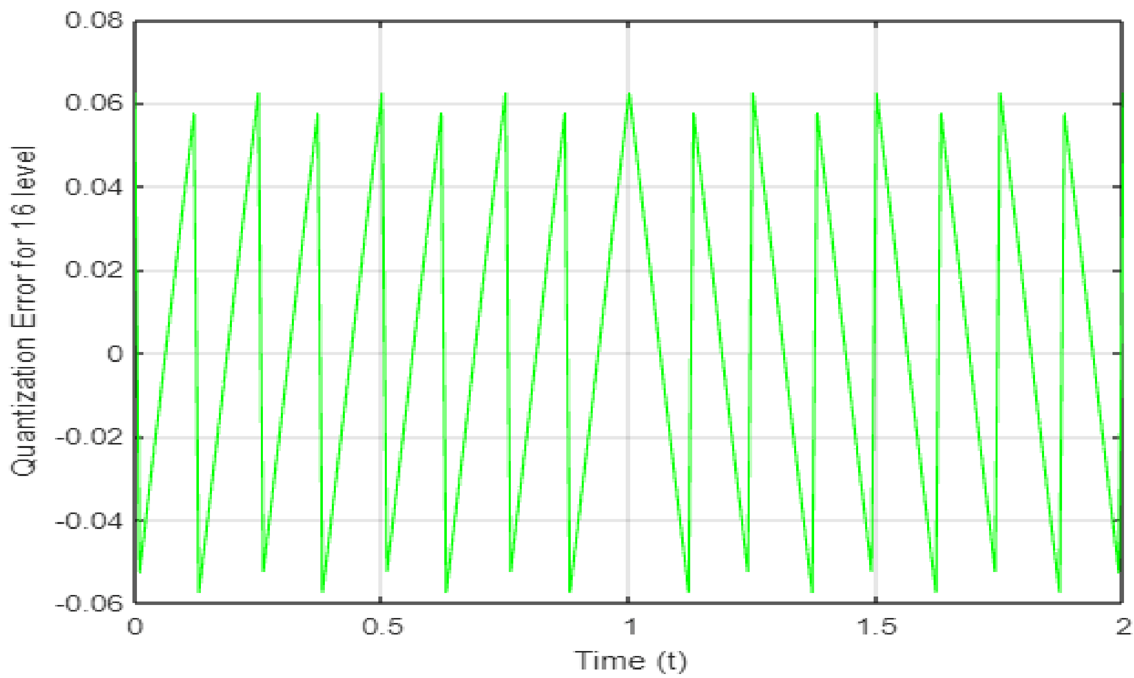


Figure 4: Quantized Error Of 8 level

Observations:

1. SQNR = 8.3898 dB.
2. Non uniform Quantizer. Step size increases as we move away form the origin.
3. Quantization Error is non Uniform. It is less for sample values nearer to the origin where as it increases gradually for higher sample values.

Problem 4.16 :-

For 8 level non-uniform u-law PCM with u=255.

- **Main Code:-**

```
clear;
clc;
close all;
t = 0:0.01:2;
yval = zeros(1,length(t));

f1 = @(x)(x); %Function
f2 = @(x)(-x+2);

for i = 1:length(t)
    p = t(i);
    if( p >= 0) && ( p < 1)
        yval(i) = f1(p);
    elseif ( p >= 1) && ( p < 2)
        yval(i) = f2(p);
    else
        yval(i) = 0;
    end
end
a = yval;
mew = 255;
[sqnr8,aquan8,code8]=mula_pcm(a,8,mew);
[sqnr16,aquan16,code16]=mula_pcm(a,16,mew);

sqnr8;
```

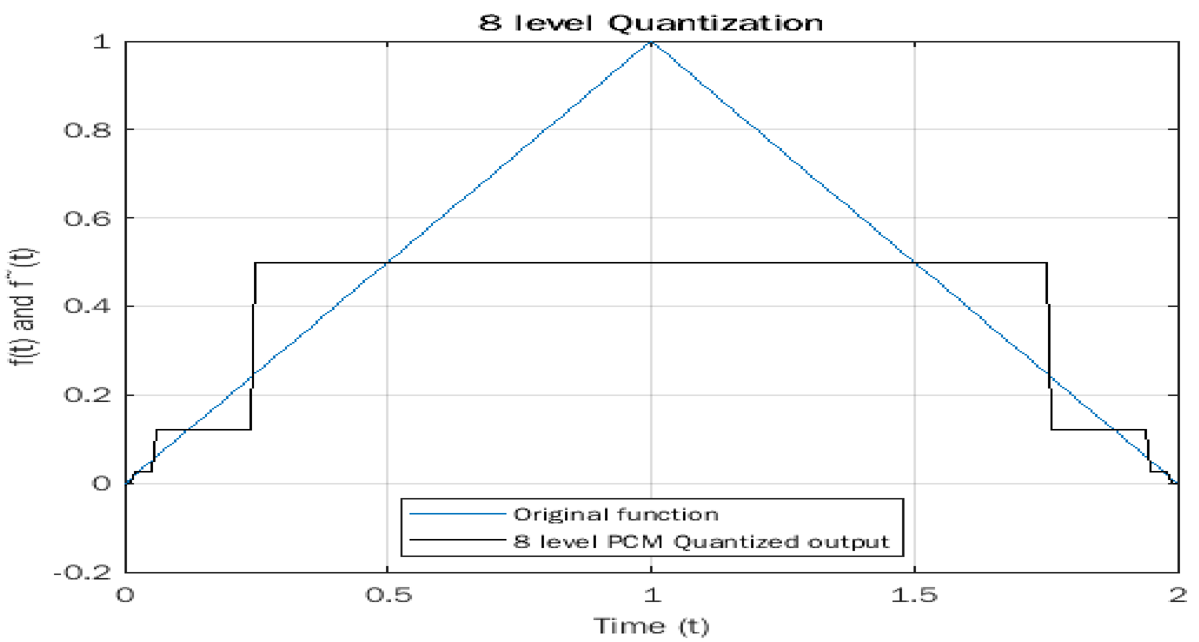
```
sqnr16;


figure;
plot(t,a,t,aquan8,'k-','linewidth',0.8);
legend('Original function', '8 level PCM Quantized
output','Location','south');
xlabel('Time (t)');
ylabel('f(t) and f^{~}(t)');
title('8 level Quantization');
grid on;
saveas(gcf,'Lab6_Q1_NonUni_8.png');

figure;
plot(t,a,t,aquan16,'k-','linewidth',0.8);
legend('Original function', '16 level PCM Quantized
output','Location','south');
xlabel('Time (t)');
ylabel('f(t) and f^{~}(t)');
title('16 level Quantization');
grid on;
saveas(gcf,'Lab6_Q1_NonUni_16.png');

figure;
plot(t, (a-aquan8), 'r-','linewidth',0.8);
xlabel('Time (t)');
ylabel('Quantization Error');
ylabel('8 level Quantization Error');
grid on;
saveas(gcf,'Lab6_Q1_NonUniError_8.png');

figure;
plot(t, (a-aquan16), 'b-','linewidth',0.8);
xlabel('Time (t)');
ylabel('16 level Quantization Error');
grid on;
saveas(gcf,'Lab6_Q1_NonUniError_16.png');

fprintf('SQNR for 8-level Quantization %f\n\n', (sqnr8));
```

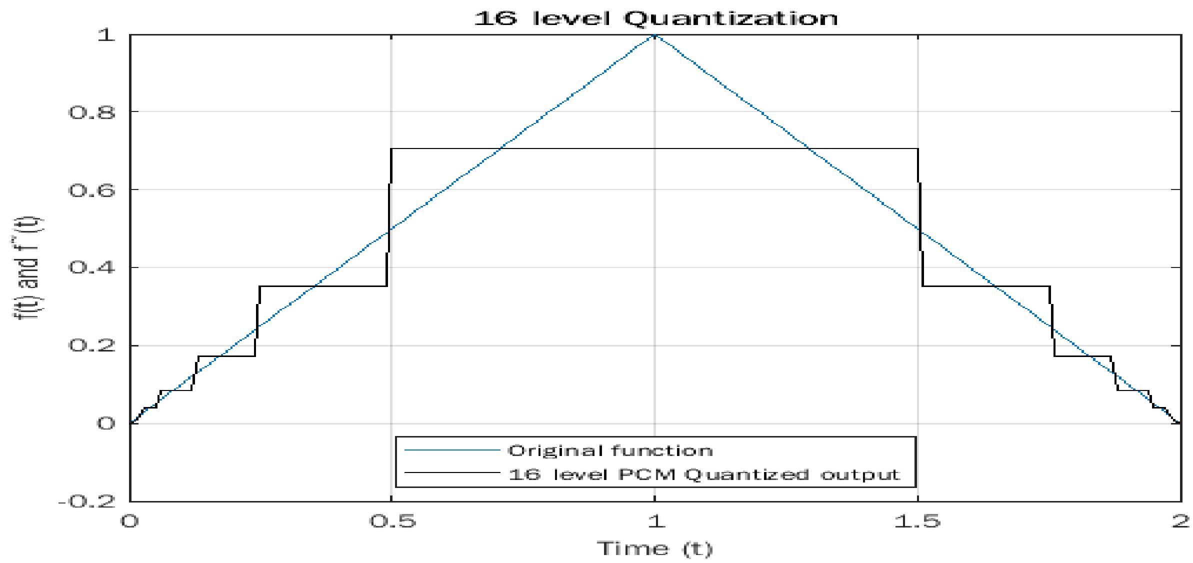
```
fprintf('SQNR for 16-level Quantization %f\n\n', (sqnr16));
```

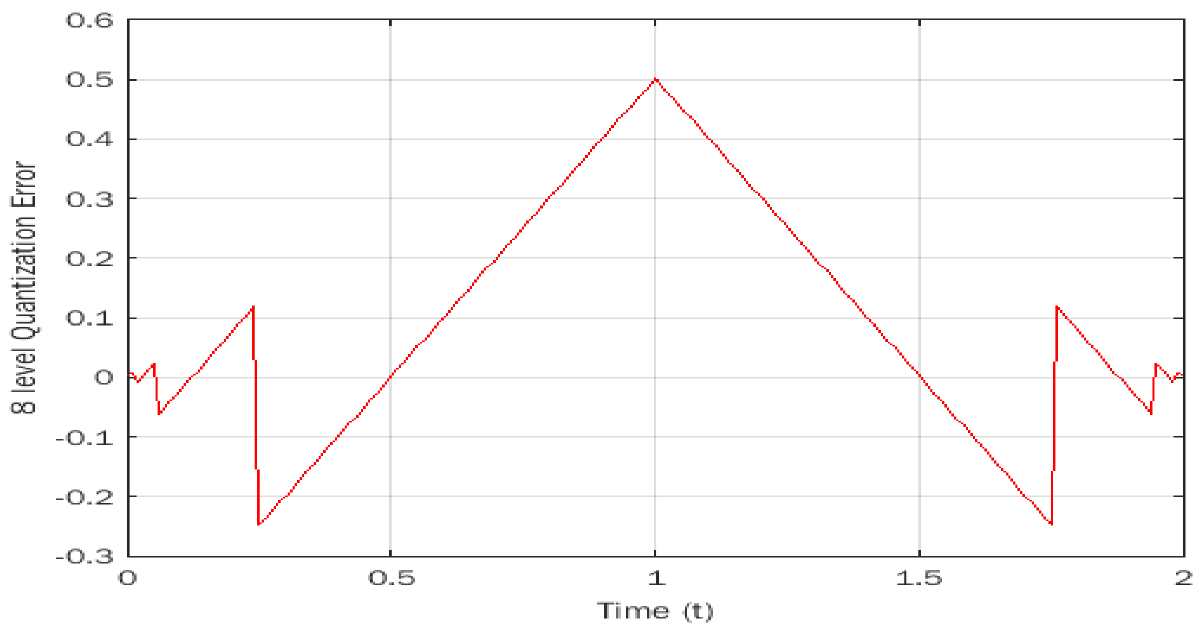
- **Function:-**

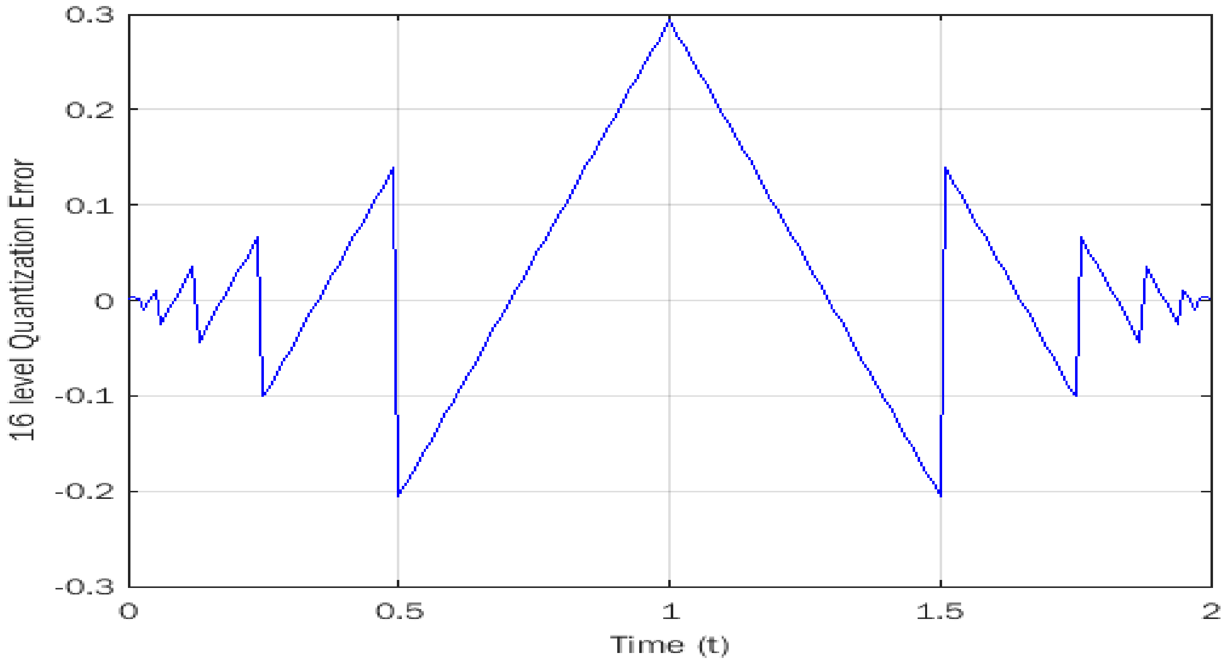
```
function [sqnr,a_quan,code]=mula_pcm(a,n,mu)
   [y,maximum]=mulaw(a,mu);
   [sqnr,y_q,code]=u1_pcm(y, n);
       a_quan=invmulaw(y_q,mu);
       a_quan=maximum*a_quan;
       sqnr=20*log10(norm(a)/norm(a-a_quan));

function [sqnr,a_quan,code]=u1_pcm(a,n)
   amax=max(abs(a));
       a_quan=a/amax;
       b_quan=a_quan;
       d=2/n;
       q=d.*(0:n-1);
       q=q-((n-1)/2)*d;

       for i=1:n
           a_quan(find((q(i)-d/2 <= a_quan) & (a_quan <=
           q(i)+d/2)))=...
           q(i).*ones(1,length(find((q(i)-d/2 <= a_quan) & (a_quan
           <= q(i)+d/2))));
           b_quan(find( a_quan==q(i) ))=(i-1) *ones(1,length(find(
           a_quan==q(i))));
       end
       a_quan=a_quan*amax;

       nu=ceil(log2(n));
       code=zeros(length(a),nu);
       for i=1:length(a)
       for j=nu:-1:0
       if (fix(b_quan(i)/(2^j))==1)
       code(i,(nu-j)) = 1;
       b_quan(i) = b_quan(i) - 2^j;
       end
```

```
        end
        end
        sqnr=20*log10(norm(a)/norm(a-a_quan));

function [y,a]=mulaw(x,mu)
        %MULAW mu-law nonlinearity for nonuniform PCM
        % Y=MULAW(X,MU).
        % X=input vector.
        a=max(abs(x));
        y=(log(1+mu*abs(x/a))./log(1+mu)).*sign(x);
end

function x=invmulaw(y,mu)
        %INVMULAW the inverse of mu-law nonlinearity
    %X=INVMULAW(Y,MU) Y=normalized output of the mu-law
    nonlinearity.
        x=(((1+mu).^(abs(y))-1)./mu).*sign(y);
end
```




16 level Quantization
f(t) and f^(t)
Original function
16 level PCM Quantized output
Time (t)


8 level Quantization Error
Time (t)



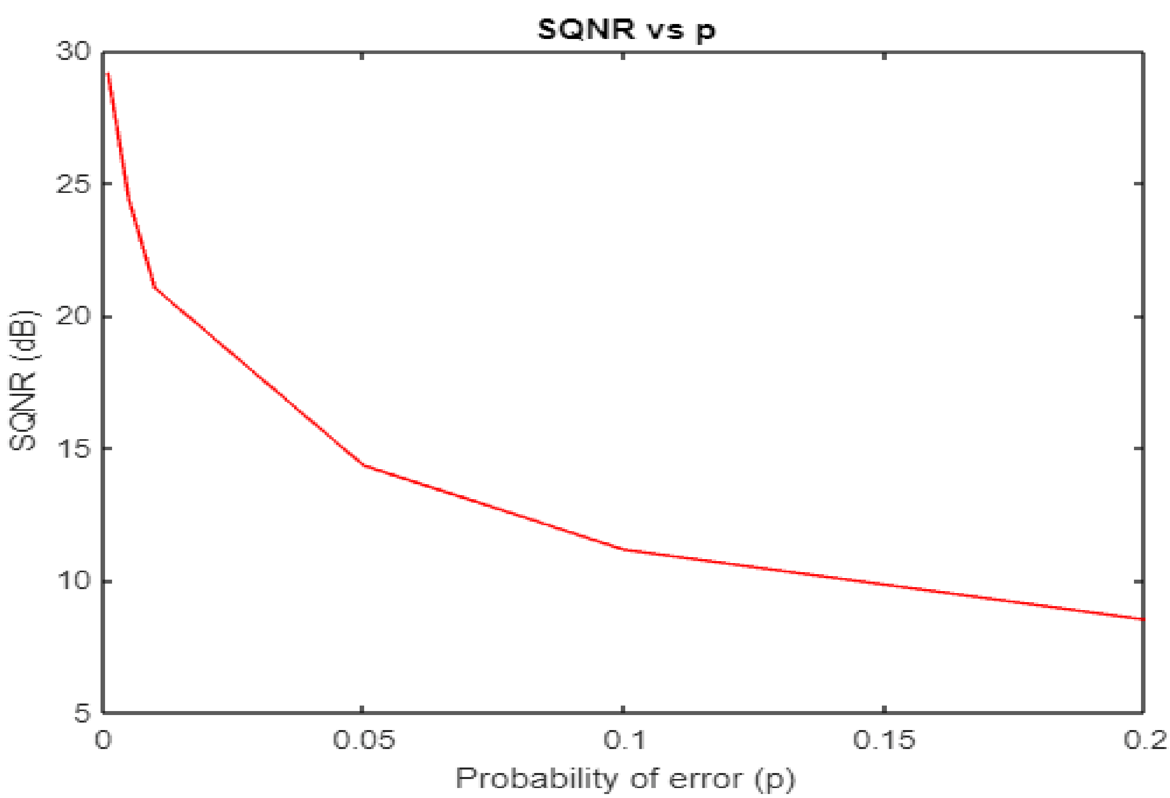
Observations:

1. SQNR = 14.0273 dB. SQNR increase as number of level increase.
2. Non Uniform Quantizer. Step size increases as we move away from the origin.
3. . Quantization Error is non Uniform. It is less for sample values nearer to the origin where as it increases gradually for higher sample values.
4. However, quantization error for a 16 level Non Uniform PCM system is less as compared to a 8 level Non Uniform PCM System.

Problem 4.15:-

- **Main Code:-**

```
a = randn(1,1000);
[sqnr64,aquan64,code64] = pcm(a,64);
p = [0.001 0.005 0.01 0.05 0.1 0.2];
SNR = zeros(1,6);
for i=1:6
noise = rand(1000,6)< p(i);
y = mod(code64 + noise,2);
SNR(i)=20*log10(norm(code64)/norm(code64-y));
end
figure(1);
plot(p,SNR);
title('SQNR vs p');
xlabel('Probability of error (p)');
ylabel('SQNR (dB)');
```

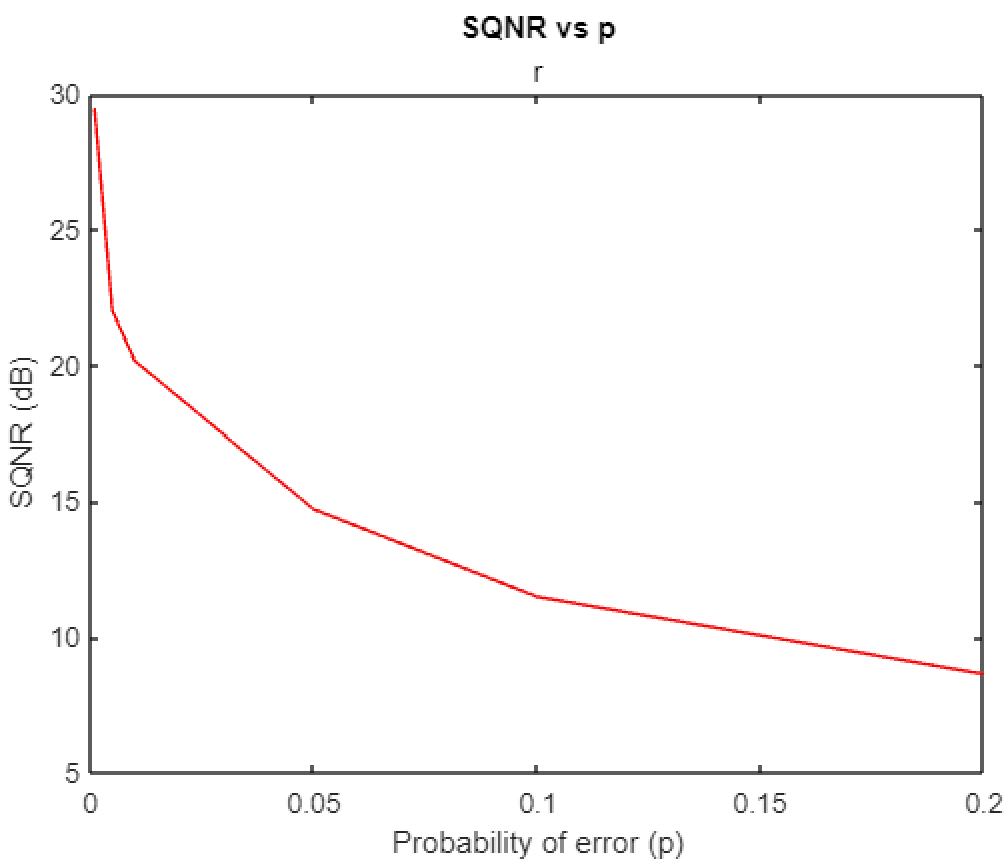


- For 6-bit-per-symbol Non-Uniform PCM with u=255.

Problem 4.18:-

➢ Main Code:-

```
a = randn(1,1000);
[sqnr64,aquan64,code64] = mula_pcm(a,64,255);
p = [0.001 0.005 0.01 0.05 0.1 0.2];
SNR = zeros(1,6);
for i=1:6
noise = rand(1000,6)< p(i);
y = mod(code64 + noise,2);
SNR(i)=20*log10(norm(code64)/norm(code64-y));
end
figure(1);
plot(p,SNR,'r');
title('SQNR vs p','r');
xlabel('Probability of error (p)');
ylabel('SQNR (dB)');
```



Observations: -

1. SQNR (dB) decreases as the probability of error increases as shown above.

Problem 2:-

sampling and reconstructing sum of two cosine functions of duration 2 seconds and frequencies 5 Hz and 8 Hz, respectively.

- **main code:**

```
clear ;
clc;
td=0.002;
%original sampling rate 500 Hz
t= [0:td:2.] ;
%time interval of 1 second
% 1Hz+3Hz sinusoids
xsig=cos(10*pi*t) + cos(16*pi*t) ;
Lsig=length(xsig);

ts=0.02;
%inew sampling rate = 50Hz.
Nfactor=ts/td;
%send the signal through a 16-level uniform quantizer
[s_out,sq_out,sqh_out,Delta,SQNR ] =
sampandquant(xsig,16,td,ts);


% calculate the Fourier transforms
Lfft=2^ceil(log2(Lsig)+1) ;
Fmax=1/ (2*td) ;
Faxis=linspace (-Fmax, Fmax, Lfft) ;
Xsig=fftshift (fft (xsig, Lfft) ) ;

S_out=fftshift (fft (s_out, Lfft) ) ;
```

```
%Examples of sampling and reconstruction using
%a) ideal impulse train through LPF
%b) flat top pulse reconstruction through LPF

%plot the original signal and the sample signals in time
%and frequency domain
figure (1) ;
subplot (311); sfigla=plot (t, xsig, 'k' ) ;

hold on; sfiglb=plot (t, s_out (1:Lsig) , 'b' ) ; hold off;
set (sfigla, 'Linewidth' , 2) ; set (sfiglb, 'Linewidth' , 2. ) ;
xlabel ( 'time ( sec) ' ) ;
title ( 'Signal {\itg}_T({\itt}) and its uniform samples' ) ;
subplot (312) ; sfiglc=plot (Faxis, abs (Xsig) ) ;
xlabel ( ' frequency (Hz) ') ;

axis ( [-150 150 0 300])
set (sfiglc, 'linewidth' , 1) ; title('Spectrum of {\itg}_T({\itt})' ) ;
subplot (313) ; sfigld=plot (Faxis, abs (S_out) ) ;
xlabel ('frequency(Hz)') ;
axis ( [-150 150 0 300/Nfactor] )

set(sfiglc , 'linewidth' , 1 ) ;
title ( ' Spectrum of {\itg}_T({\itt})' ) ;

BW= 10;
%Bandwidth i s no l arger than l 0H z .
H_lpf = zeros (1 , Lfft ) ; H_lpf (Lfft / 2 - BW : Lfft / 2+BW- 1 ) = 1 ;
% i deal LPF
S_recv=Nfactor * S_out .* H_lpf ;
% ideal f i l ter ing
s_recv= real(ifft ( fftshift ( S_recv ) ) ) ;
% recons t ructed £ - domain
s_recv= s_recv (1 : Lsig ) ;

figure (2)
subplot(211); sfig2a=plot ( Faxis , abs ( S_recv ) ) ;
xlabel ( ' frequency ( Hz ) ' ) ;
axis ( [ -150 150 0 300 ] ) ;
title ( ' Spectrum of ideal filtering ( reconstruction ) ' ) ;
```

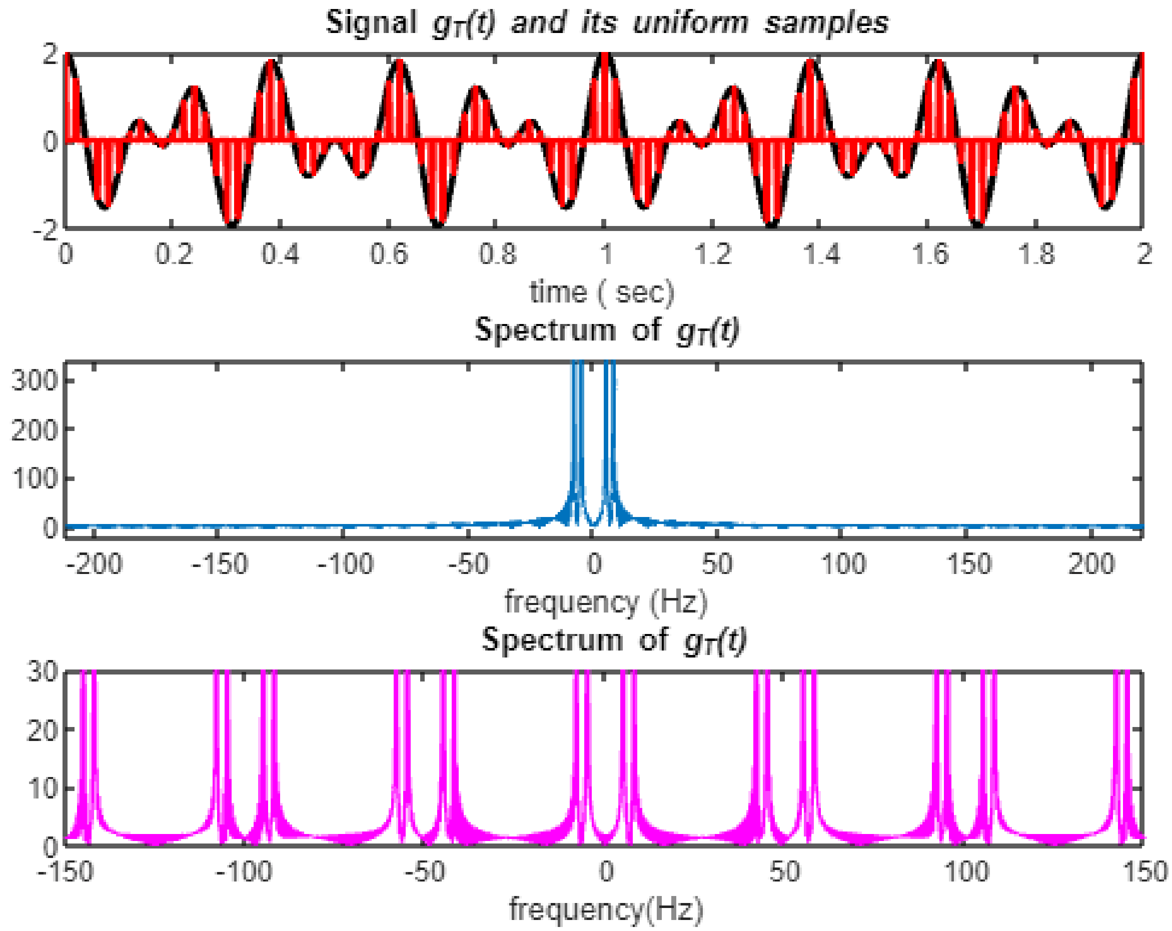
```
subplot(212 ) ;
sfig2b=plot ( t , xsig , 'k-.' , t , s_recv (1 : Lsig ) , 'b' );
legend ( ' original signal ' , ' reconstructed signal ' ) ;
xlabel ( ' time(sec)' ) ;
title ( ' original signal versus ideally reconstructed signal ' ) ;
set ( sfig2b , 'linewidth' , 2 ) ;


%non-ideal reconstruction
ZOH=ones (1, Nfactor) ;
s_ni=kron (downsample (s_out, Nfactor ) , ZOH) ;
S_ni=fftshift (fft (s_ni, Lfft) ) ;

S_recv2=S_ni.*H_lpf;
%ideal filtering
s_recv2=real (ifft (fftshift (S_recv2) ) ) ;
% reconstructed f-domain
s_recv2=s_recv2 (1: Lsig) ;
% reconstructed t-domain
% plot the ideally reconstructed signal in time and frequency
domain
figure(3)
subplot(211) ; sfig3a=plot (t, xsig, 'b', t, s_ni (1:Lsig) , 'b' );
xlabel ( 'time (sec) ' ) ;
title ( 'original signal versus flat-top reconstruction' ) ;

subplot (212 ) ;
sfig3b=plot (t, xsig, 'b' , t, s_recv2 (1:Lsig) , 'b--' ) ;
legend ( 'original signal' , 'LPF reconstruction' ) ;
xlabel ('time (sec) ' ) ;
set (sfig3a, 'Linewidth' , 2) ; set (sfig3b, 'Linewidth' , 2) ;
title ( 'original and flat-top reconstruction after LPF' ) ;
```

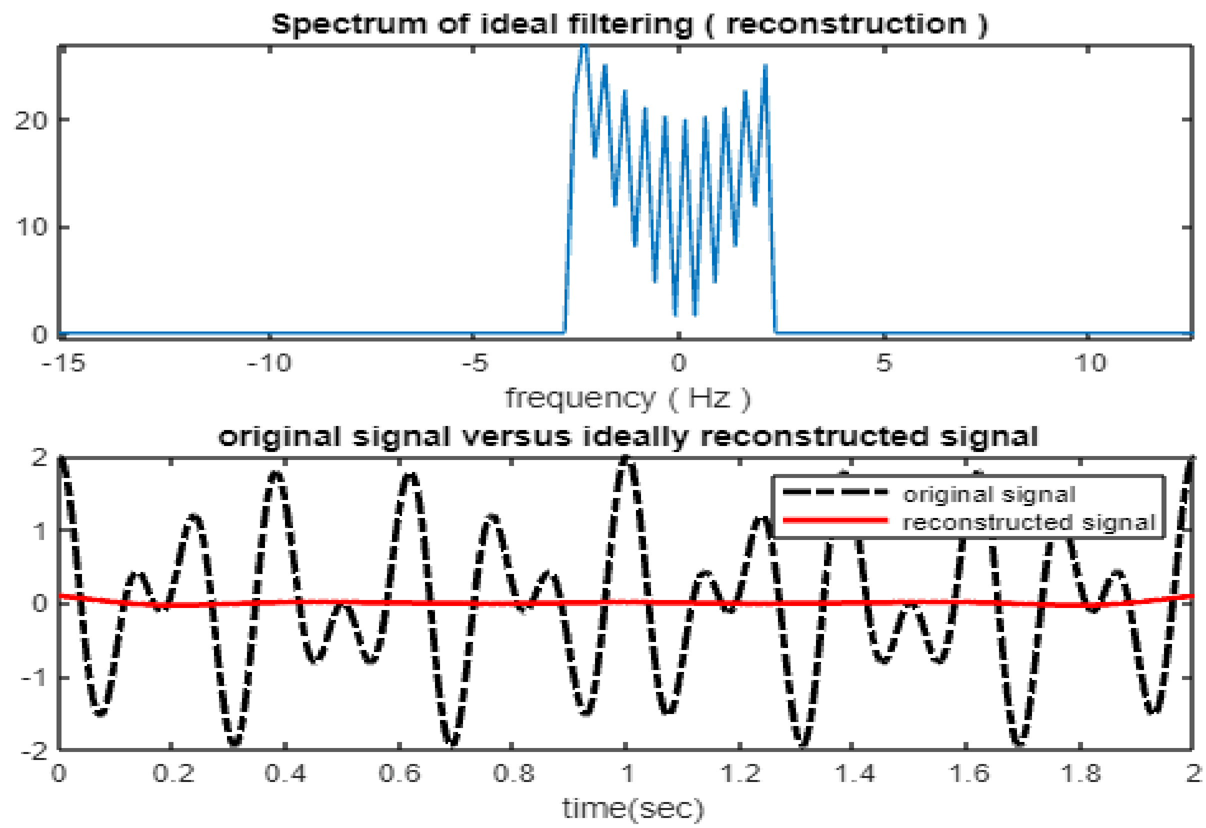
- **functions :**

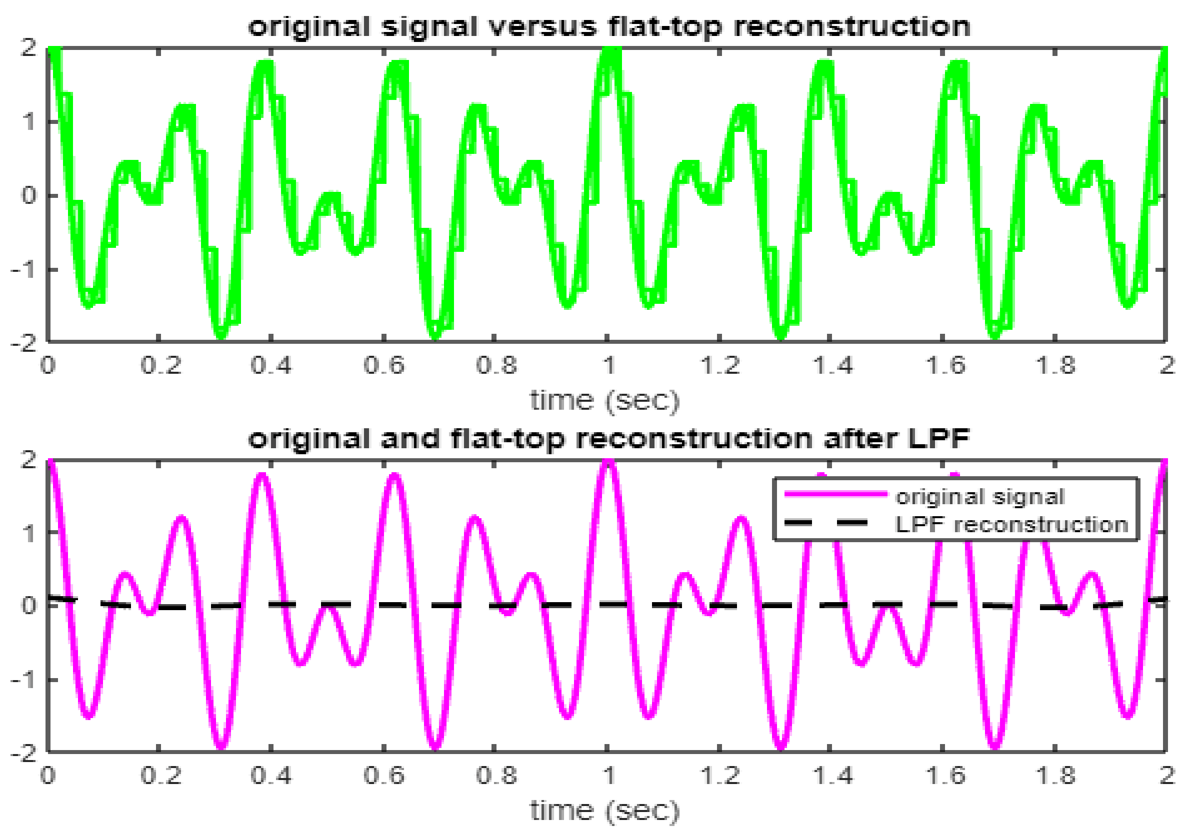
```
function [ s_out , sq_out , sqh_out , Delta , SQNR ] =
sampandquant(sig_in,L,td,ts )
```

```
if ( rem(ts/td, 1 ) == 0 )
nfac=round (ts/td);
p_zoh=ones (1,nfac) ;
s_out=downsample (sig_in, nfac) ;
[sq_out, Delta, SQNR] =uniquan(s_out, L) ;

s_out=upsample (s_out , nfac) ;
sqh_out=kron (sq_out, p_zoh) ;
sq_out=upsample (sq_out, nfac) ;
else

warning ( 'Error! ts/td is not an integer! ') ;
s_out=[] ; sq_out= [] ; sqh_out= [] ; Delta= [ ] ; SQNR= [] ;
end
end



function [ q_out , Delta , SQNR ] =uniquan(sig_in , L)

sig_pmax=max(sig_in) ;

sig_nmax=min(sig_in);
Delta= (sig_pmax-sig_nmax) /L;

q_level=sig_nmax+Delta/2 : Delta: sig_pmax-Delta/2; % define
Q-levels

L_sig=length(sig_in);

sigp= (sig_in-sig_nmax) /Delta+1/2;
qindex=round (sigp) ;

qindex=min (qindex, L) ;

q_out=q_level (qindex) ;
SQNR=20*log10 (norm(sig_in)/norm(sig_in-q_out) ) ;
end
```

Plots :


Signal $g_T(t)$ and its uniform samples
time (sec)
Spectrum of $g_T(t)$
frequency (Hz)
Spectrum of $g_T(t)$
frequency(Hz)


Spectrum of ideal filtering (reconstruction)
frequency (Hz)
original signal versus ideally reconstructed signal
original signal
reconstructed signal
time(sec)


original signal versus flat-top reconstruction
time (sec)
original and flat-top reconstruction after LPF
original signal
LPF reconstruction
time (sec)