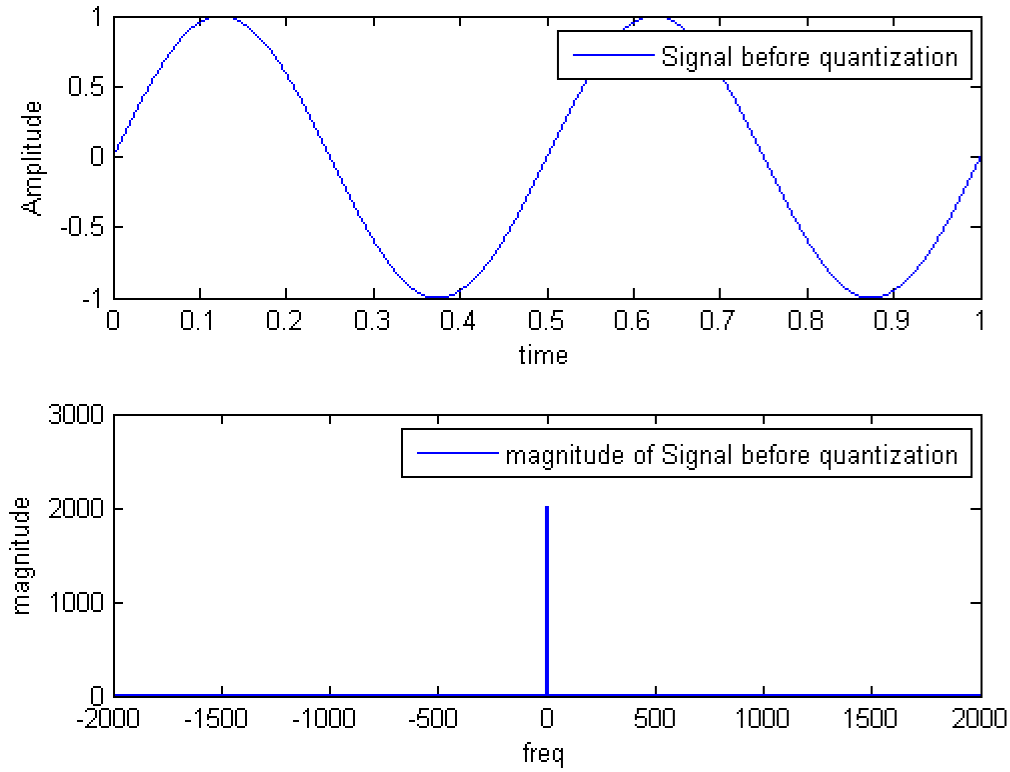
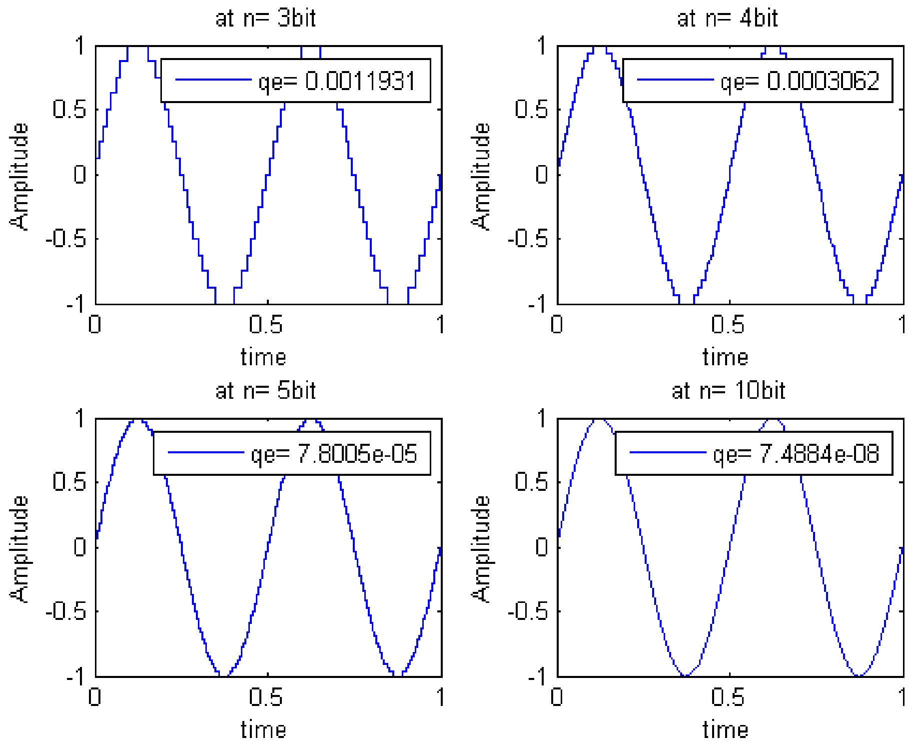
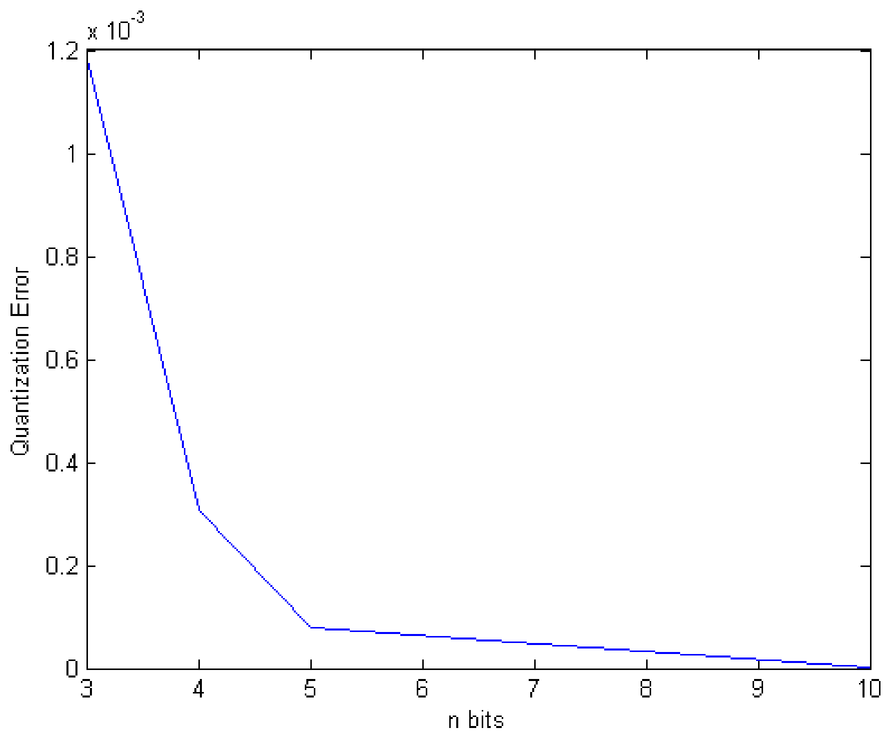
```
%------------------------Pulse Code Modulation--------------------

clear

clc

%______(1) Generate a sinusoidal wave of the following parameters:___%

amplitude = 1;

frequency = 2;

sampling_frequency = 4000;

time = 0:1/sampling_frequency:1;

sinusoid = amplitude * sin(2*pi*frequency*time);

figure(1)

subplot(2,1,1)

plot(time,sinusoid)

ylabel('Amplitude');

xlabel('time');

legend('Signal before quantization')

fftsinusoid=fft(sinusoid);

fftsinusoid=fftshift(fftsinusoid);

fs=4000;

freq = linspace(-fs/2, fs/2, length(fftsinusoid));

subplot(2,1,2)

plot(freq,abs(fftsinusoid))

xlabel('freq')

ylabel('magnitude');

legend('magnitude of Signal before quantization')
```




```
%______(2) Quantize the sampled signal by m bits where m= 2n+1 and n is the
%number of bits that will represents the integer value and the fraction part
% and the last bit is the sign bit %
n = [3,4,5,10]; % number of bits for integer and fraction part
m = 2.*n+1; % total number of bits (including sign bit)
figure(2)
mean_square_error = zeros(size(n));
for i=1:length(n)
%  the fi function is being used to quantize a sinusoidal signal,
%  resulting in a fixed-point data object that represents the quantized signal.
%  The double function is then used to convert this fixed-point data object to
%  a double precision floating-point format, which can be used with other MATLAB
%  functions and operations that work with floating-point data types.
```

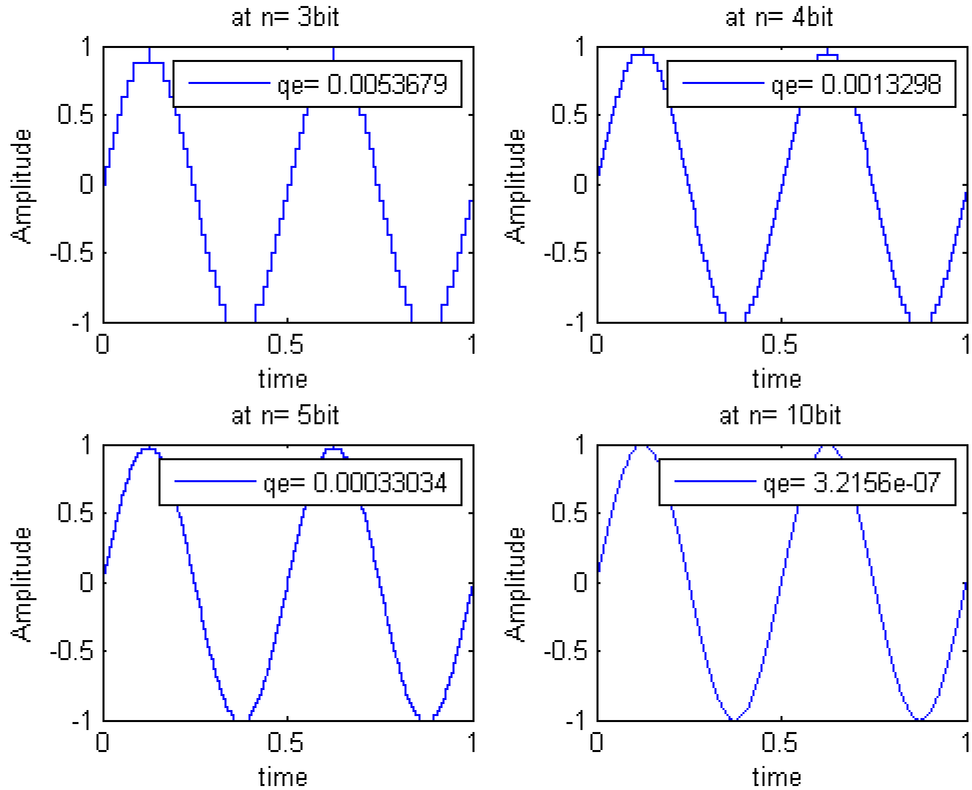
```
    quantized_signal = double(fi(sinusoid,1,m(i),n(i)));

    quantization_error = mean((quantized_signal - sinusoid).^2);   %qe from the equation

    mean_square_error(i) = quantization_error;

    subplot(2,2,i)

    plot(time,quantized_signal,'b')

    ylabel('Amplitude'); xlabel('time');

    legend(['qe= ',num2str(quantization_error)])

    title(['at n= ',num2str(n(i)),'bit'])

    binary_signal = dec2bin(abs(quantized_signal));

%     % Display the quantized and binary signals for n=3

     if n(i) == 3

%         disp('Quantized signal:')

%         disp(quantized_signal')

%         disp('Binary signal:')

%         disp(binary_signal')

   end

end

figure(3)

plot(n,mean_square_error);

ylabel('Quantization Error');

xlabel('n bits');
```

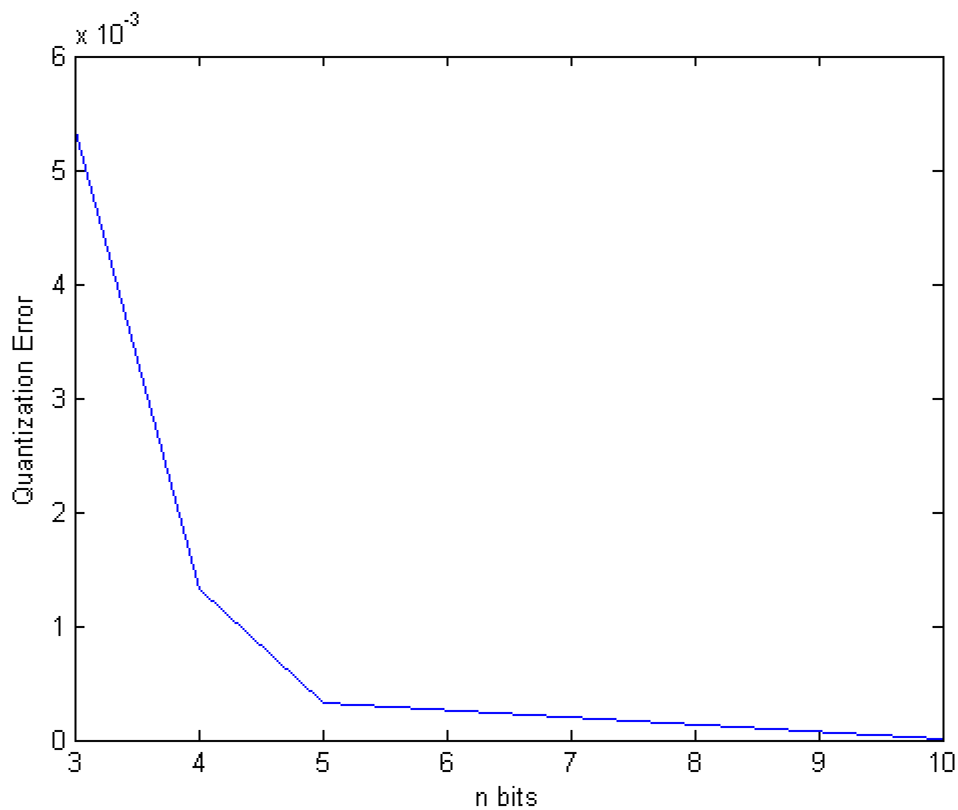





```
figure(4)

mean_square_error = zeros(size(n));
```

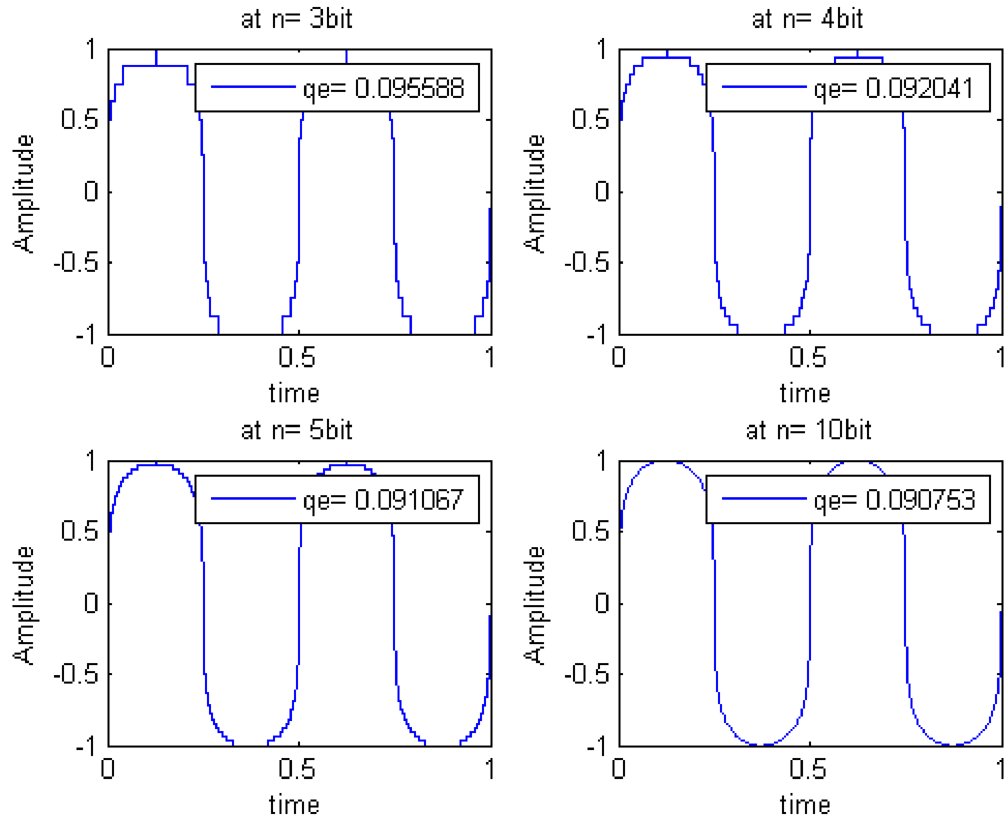
```
for i=1:length(n)

   q = quantizer('fixed', [m(i) n(i)]);

   %a quantizer object with properties set to its inputs.


   quantized_signal=quantize(q, sinusoid);

   quantization_error = mean((quantized_signal - sinusoid).^2);  %--> quantization error from eqn

   mean_square_error(i) = quantization_error;

   subplot(2,2,i)

   plot(time,quantized_signal,'b')

   legend(['qe= ',num2str(quantization_error)])

    ylabel('Amplitude');

    xlabel('time');

   title(['at n= ',num2str(n(i)),'bit'])

end

figure(5)

plot(n,mean_square_error);

ylabel('Quantization Error');

xlabel('n bits');
```

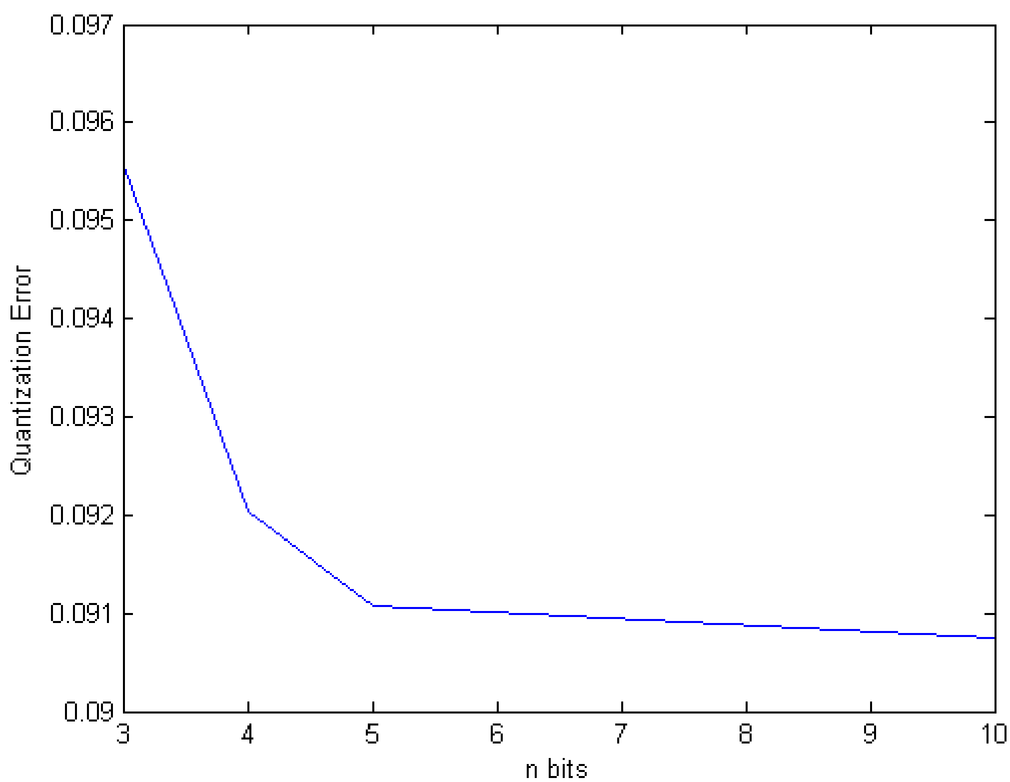





```
compressed = compand(sinusoid,255,max(sinusoid),'mu/compressor');
```

```
% Non-uniform quantization is achieved by, first passing the input signal through a compressor.

% The output of the compressor is then passed through a uniform quantizer.

% The combined effect of the compressor and the uniform quantizer is that of a non-uniform
quantizer.

% At the receiver the signal is restored to its original form by using an expander.



%*******************

figure(6)

mean_square_error = zeros(size(n));

for i=1:length(n)

   q = quantizer('fixed', [m(i) n(i)]);



   quantized_signal=quantize(q, compressed);      %Non uniform quantization

   quantization_error = mean((quantized_signal - sinusoid).^2);  %--> quantization error from eqn

   mean_square_error(i) = quantization_error;



   subplot(2,2,i)

   plot(time,quantized_signal,'b')

   legend(['qe= ',num2str(quantization_error)])

   ylabel('Amplitude'); xlabel('time');

   title(['at n= ',num2str(n(i)),'bit'])




end

figure(7)

plot(n,mean_square_error);

ylabel('Quantization Error');

xlabel('n bits');
```


at n= 3bit
qe= 0.095588
at n= 4bit
qe= 0.092041
at n= 5bit
qe= 0.091067
at n= 10bit
qe= 0.090753
Amplitude
time

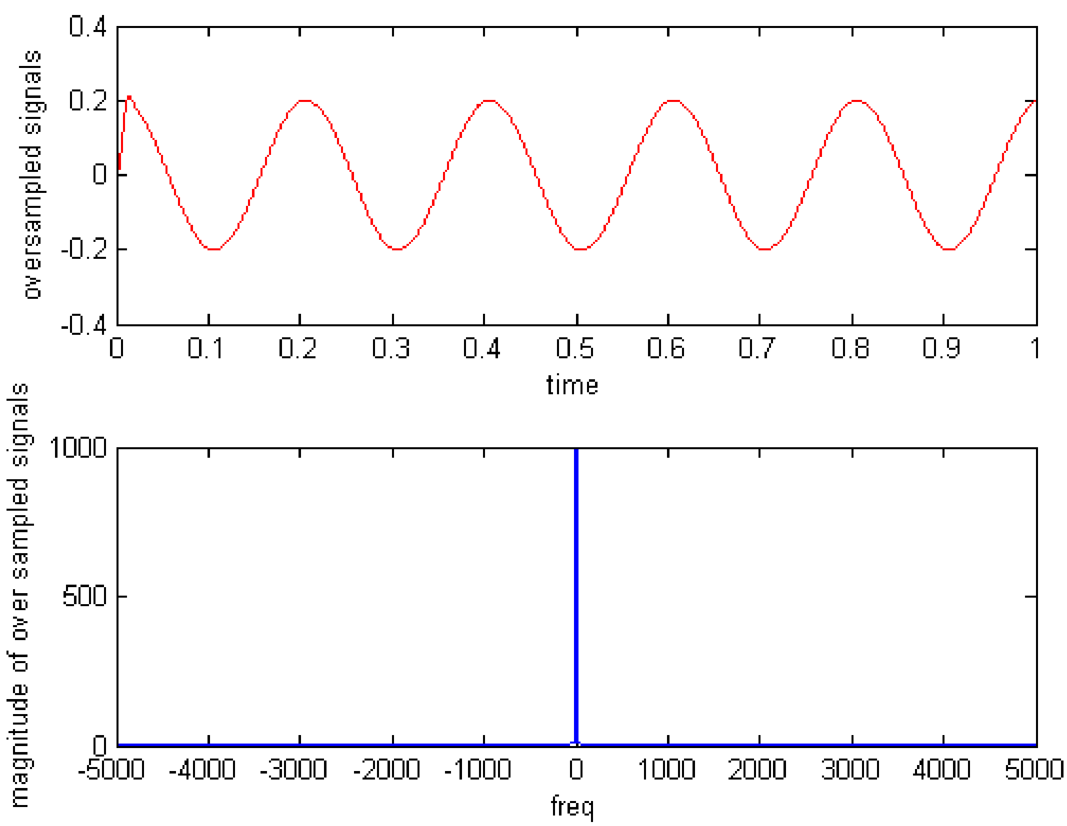

Quantization Error
n bits

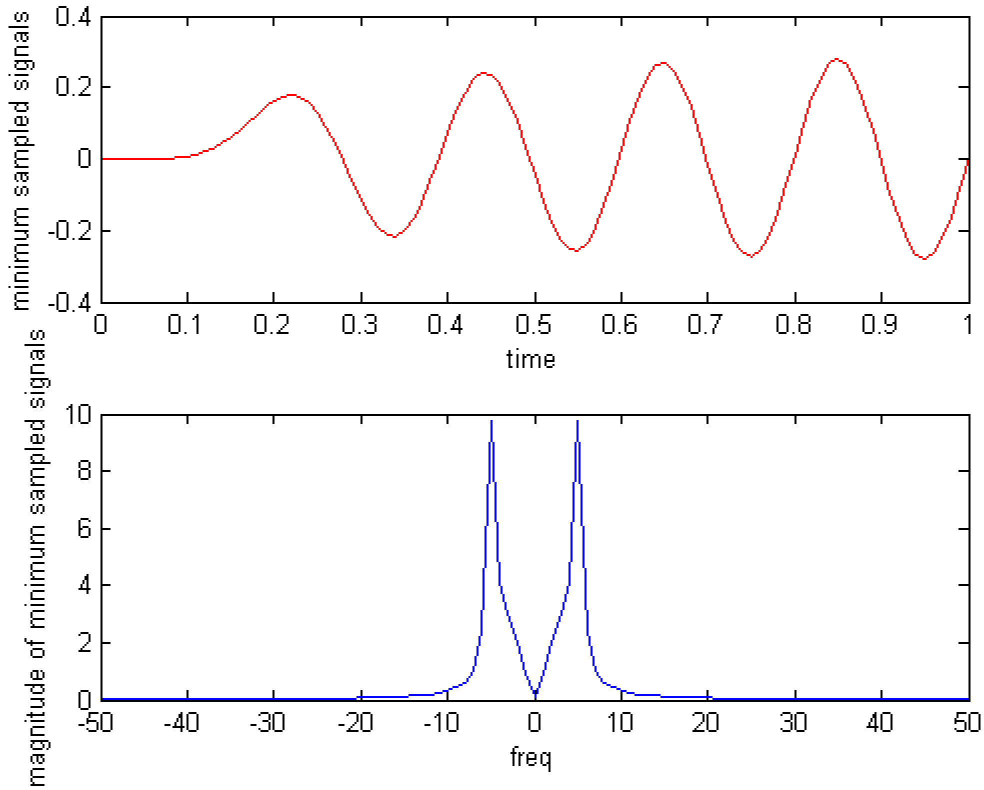
```
%-------------part2--------------------------------
% reconstruction from oversampling
t=0:0.001:1;% time signal Ts = 0.001 then fs =1000;
y=2*cos(2*pi*5*t); %fm = 5
[B,A] = butter(3,1000/100000,'low'); % normalized cutoff frequency = 0.01
zero_added_signal=zeros(1,length(y)*10);
for i=1:length(y)
zero_added_signal(i*10)=y(i);
end
zero_added_signal(1:9)=[];

% Adding zeros enhances the signal display and don't change the
%spectrum,it changes sampling freq. only
t=linspace(0,1,length(zero_added_signal));
filtered_signal = filter(B,A,zero_added_signal);
figure(1)

subplot(2,1,1)
plot(t,filtered_signal,'r' )
xlabel('time')
ylabel('oversampled signals')
s=fft(filtered_signal);
s=fftshift(s);
fs=10000;
freq=linspace(-fs/2,fs/2,length(s));

subplot(2,1,2)
```

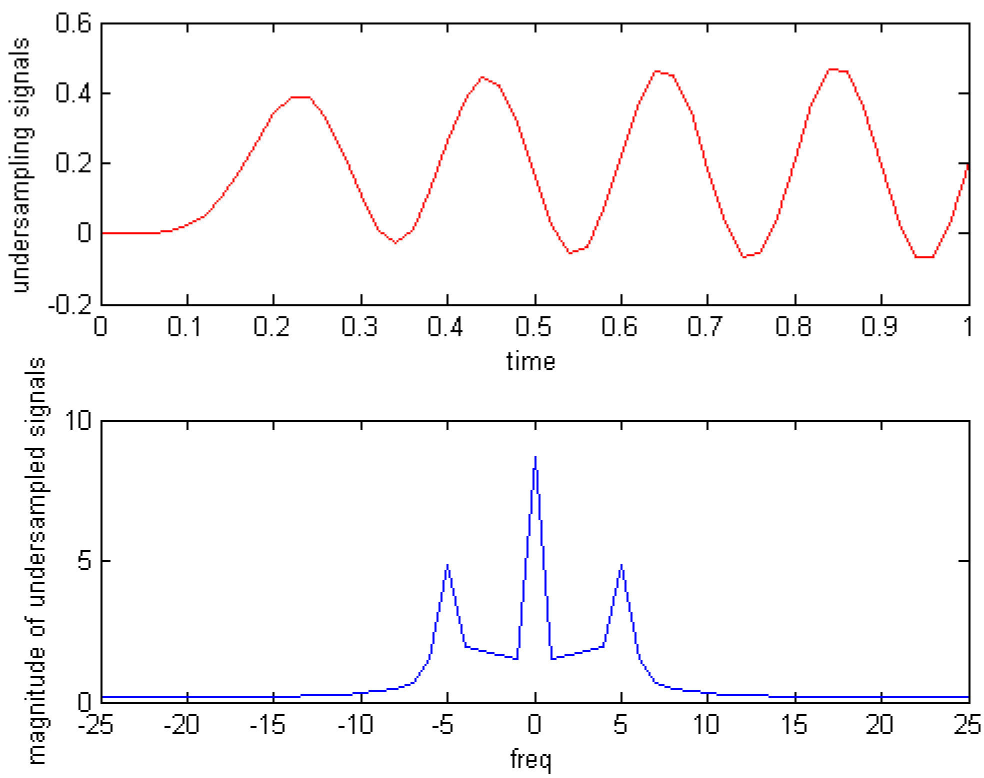
```
plot(freq,abs(s))

xlabel('freq')

ylabel('magnitude of over sampled signals')

% construction from minimum sampling

t=0:1/10:1; % fs = 2fm =10 (critical sampling )

y=2*cos(2*pi*5*t);

[B,A] = butter(10,0.1,'low' );

zero_added_signal=zeros(1,length(y)*10);

for i=1:length(y)

zero_added_signal(i*10)=y(i);

end

zero_added_signal(1:9)=[];


t=linspace(0,1,length(zero_added_signal));

filtered_signal = filter(B,A,zero_added_signal);

figure(2)

subplot(2,1,1)

plot(t,filtered_signal,'r' )

xlabel('time')

ylabel('minimum sampled signals')

s=fft(filtered_signal);

s=fftshift(s);

fs=100; % why 100?? By adding zeros between each sample of the original signal,

          % the minimum sampled signal has a higher sampling rate than

          % the original signal which is equal 10*10

freq=linspace(-fs/2,fs/2,length(s));

subplot(2,1,2)
```

```
plot(freq,abs(s))

xlabel('freq')

ylabel('magnitude of minimum sampled signals')

% construction from undersampling sampling

t=0:0.2:1; %fs = 5 less than nyquest rate (fN=10);

y=2*cos(2*pi*5*t);

[B,A] = butter(10,0.2,'low' );

% complete this part as shown in the construction from minimum sampling

%and do the necessary changes , you have to do low pass filtering and

% displays the spectrum

zero_added_signal=zeros(1,length(y)*10);

for i=1:length(y)

zero_added_signal(i*10)=y(i);

end

zero_added_signal(1:9)=[];


t=linspace(0,1,length(zero_added_signal));

filtered_signal = filter(B,A,zero_added_signal);

figure(3)

subplot(2,1,1)


plot(t,filtered_signal,'r' )

xlabel('time')

ylabel('undersampling signals')

s=fft(filtered_signal);

s=fftshift(s);

fs=50;
```

```
freq=linspace(-fs/2,fs/2,length(s));

subplot(2,1,2)

plot(freq,abs(s))

xlabel('freq')

ylabel('magnitude of undersampled signals')

% Figure 6: This shows the spectrum of the undersampled signal.

% It suffers from aliasing, as seen by the presence of additional frequency

% components that were not present in the original signal.
```




minimum sampled signals
0.4
0.2
0
-0.2
-0.4
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
time
magnitude of minimum sampled signals
10
8
6
4
2
0
-50 -40 -30 -20 -10 0 10 20 30 40 50
freq


undersampling signals
0.6
0.4
0.2
0
-0.2
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
time
magnitude of undersampled signals
10
5
0
-25 -20 -15 -10 -5 0 5 10 15 20 25
freq