

CHITTAGONG UNIVERSITY OF ENGINEERING & TECHNOLOGY



DEPARTMENT OF ELECTRICAL & ELECTRONIC ENGINEERING

Name of the Student : Mohammad Iftekher Ebne Jalal

Student ID : 1802119

Section : B

Course Code : EEE-366

Course Title : Digital Signal Processing Sessional

Experiment No. : 01

Experiment Name : Study of Sampling, Quantization and Encoding.

Date of Performance : 09.11.2022

Date of Submission : 16.11.2022

Remarks :

Table of Contents

1. Objectives	3
2. Part A : Sampling of an Analog Signal and Reconstruction of the Sample Signal	3
Step 01 : Generate the signal : $x(t) = \sin(2\pi ft + 0^\circ)$	3
Step 02 : Taking samples at 20 Hz sampling frequency	4
Step 03 : Reconstruct Analog signal by interpolation	5
Step 04 : Repeating step 2 and 3 at the sampling rate of 50Hz, 100Hz and 10 Hz.....	7
Step 05 : Generate another Signal	9
Step 06 : determining the Sampling frequency for the signal in Step 05	10
Comment on the obtained Results from Step 4 and Step 6 :	12
3. Part B : Study of Quantization and Quantization Error	15
Step 01 : Source code on Matlab to realize a Uniform Quantizer.....	15
Step 02 : Sample the signal in Step 05 of Part A in 200 Hz.....	16
Step 03 : Quantize the Signal by 3-bit Uniform Quantizer	18
Step 05 : Obtaining SQNR from 2 Equations	21
Step 06 : Reconstruct the Signal by Interpolation	21
Step 07 : Repeat Step3 to 6 for 4 and 6 bit Uniform Quantizer	23
Comment on the result obtained in Step 06 :	27
4. Task C : Encoding the Quantized Sequence	27
Step 01 : Obtain sampled Quantized Data.....	27
Step 02 : Assigning Binary Value	28

1. Objectives

- (A) Sampling of an analogue signal and reconstruction of the sample signal
- (B) Study of quantization and quantization error
- (C) Encoding of the quantized sequence

2. Part A : Sampling of an Analog Signal and Reconstruction of the Sample Signal

Step 01 : Generate the signal : $x(t) = \sin(2\pi ft + 0^\circ)$

Code :

```
% Analog Signal Generation  
  
t=0:0.001:0.2;  
f=10;  
A=1;  
x=A*sin(2*pi*f*t);  
figure(1)  
plot(t,x,'r','Linewidth',2);  
title('Analog signal (10Hz)')  
grid on
```

Output :

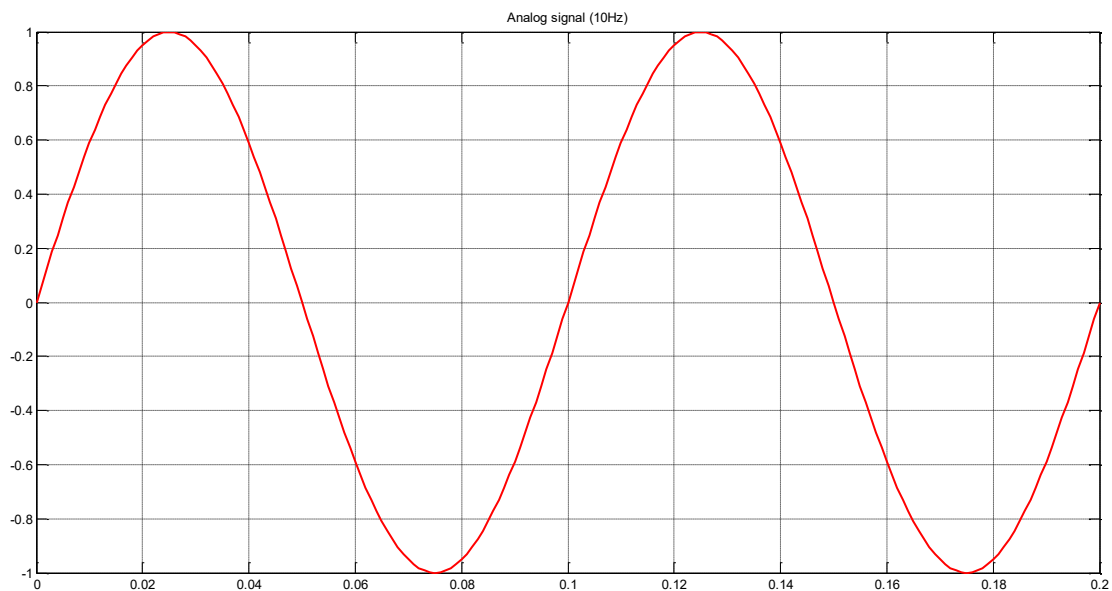


Figure: Analog signal (10Hz)

Step 02 : Taking samples at 20 Hz sampling frequency

(Adding to the previous code)

Code:

```
%% Sampling Signal with 20Hz Sampling frequency
```

```
fs20=2*f;  
ts20=0:1/fs20:0.2;  
xs20=A*sin(2*pi*f*ts20);
```

```
figure(2)  
subplot(221);  
stem(ts20,xs20,'Linewidth',2);  
title('Sampling signal (20Hz)')  
grid on
```

Output :

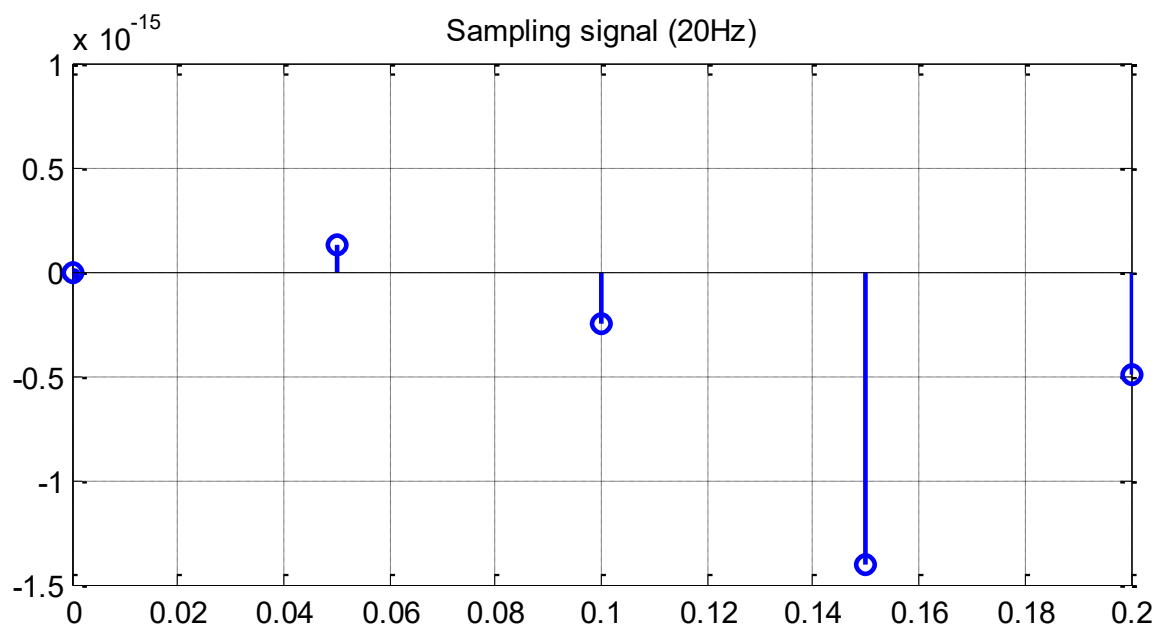


Figure: Sampled Signal at 20Hz Frequency

Step 03 : Reconstruct Analog signal by interpolation

Interpolate function syntax appears as *interp1(x,v,xq)* where, x=sample points, v=corresponding values, xq= specific query points.

In this work, we will be considering specific query points to be from 0 to 0.2 with the interval of (1/200) or **0.005 second**. The method is chosen to be *spline*.

Code :

```
% Interpolation
figure (3)
ti20=0:1/200:0.2; % Interpolate values for every 1/200 second
xi20=interp1(ts20, xs20, ti20, 'spline')
stem(ts20, xs20);
hold on
plot(ti20, xi20);
title('Sampling signal (20Hz)')
```

Output :

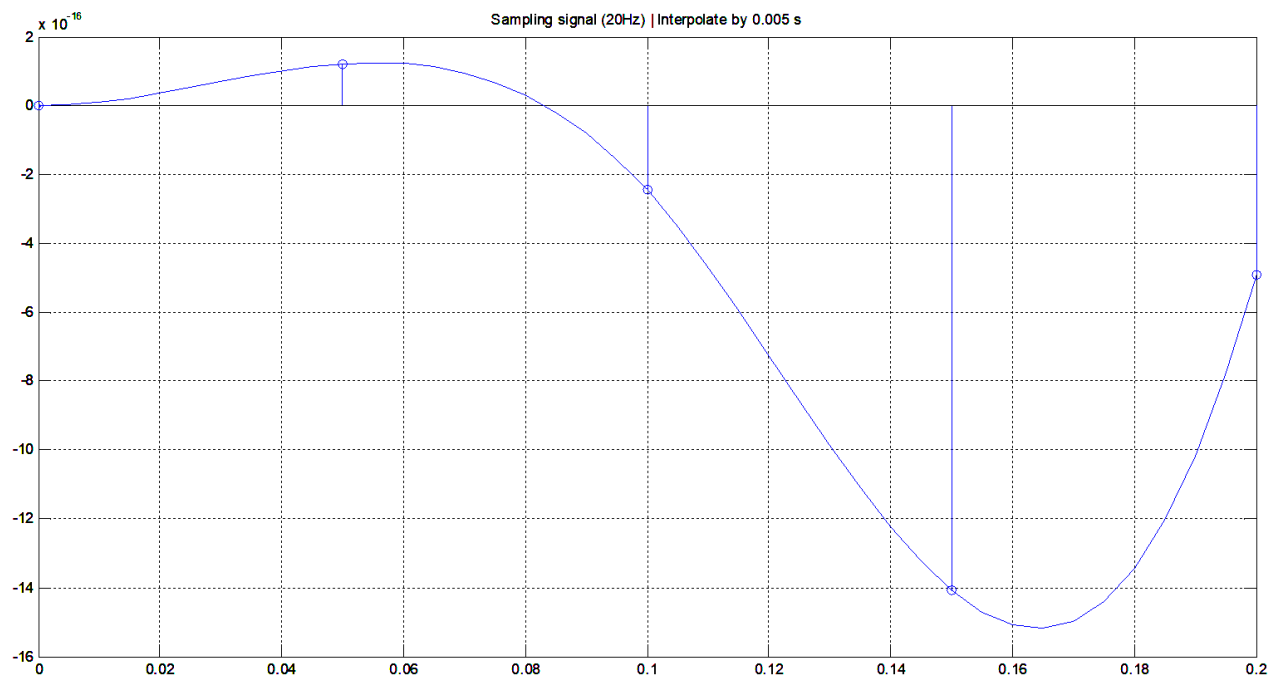


Figure: Interpolated Signal (0.005s Interval)

We can reduce the interval and interpolate only the specific sampling points by taking the interval similar to the sampling signal interval (i.e. specific query points). Only 1 line needs change here which is :

```
ti20=0:1/fs:0.2;
```

If the **interval is reduced to 1/20 second which is equal to the sampling rate** , we will get

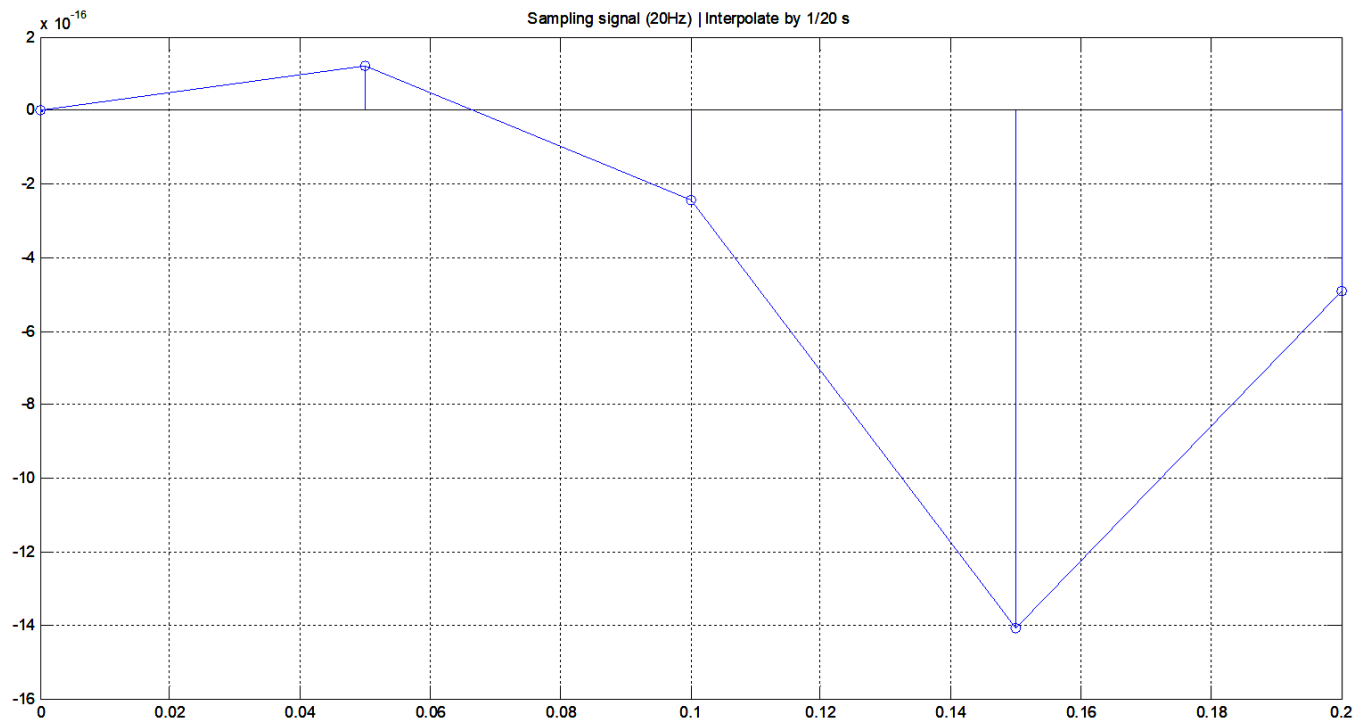


Figure: Interpolated Signal (1/20 s Interval)

Step 04 : Repeating step 2 and 3 at the sampling rate of 50Hz, 100Hz and 10 Hz

Code (50 Hz) :

```
% Sampling and Interpolating at 50Hz Sampling frequency

fs50=5*f;      % Sampling frequency fs=50Hz as f=10 in step 1
ts50=0:1/fs50:0.2;
xs50=A*sin(2*pi*f*ts50);
ti50=0:1/fs50:0.2; % Interpolate at the sampling frequency
xi50=interp1(ts50,xs50,ti50,'spline')
figure(2)
subplot(222);
stem(ts50,xs50);
hold on
plot(ti50,xi50);
title('Sampling signal (50Hz)')
grid on
```

Output (50 Hz):

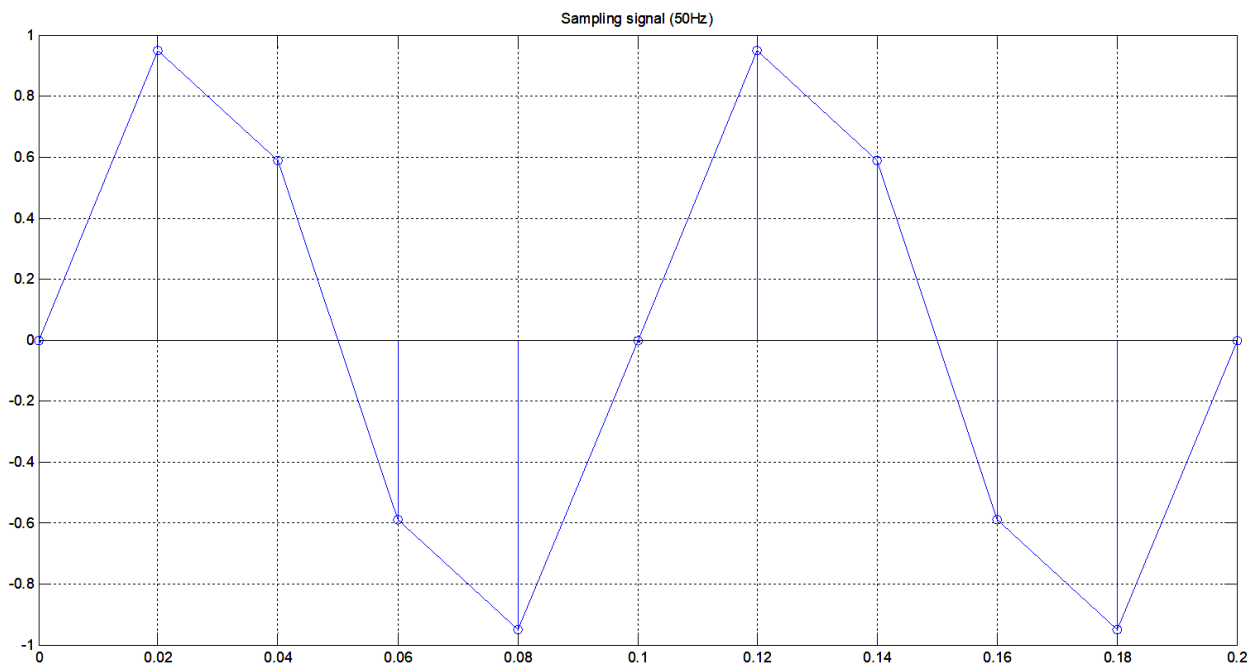


Figure: Sampled and Interpolated Signal (50Hz)

No by changing the sampling frequency f_s , we can derive the output for both 100Hz and 10Hz which are as follows

Output (100 Hz):

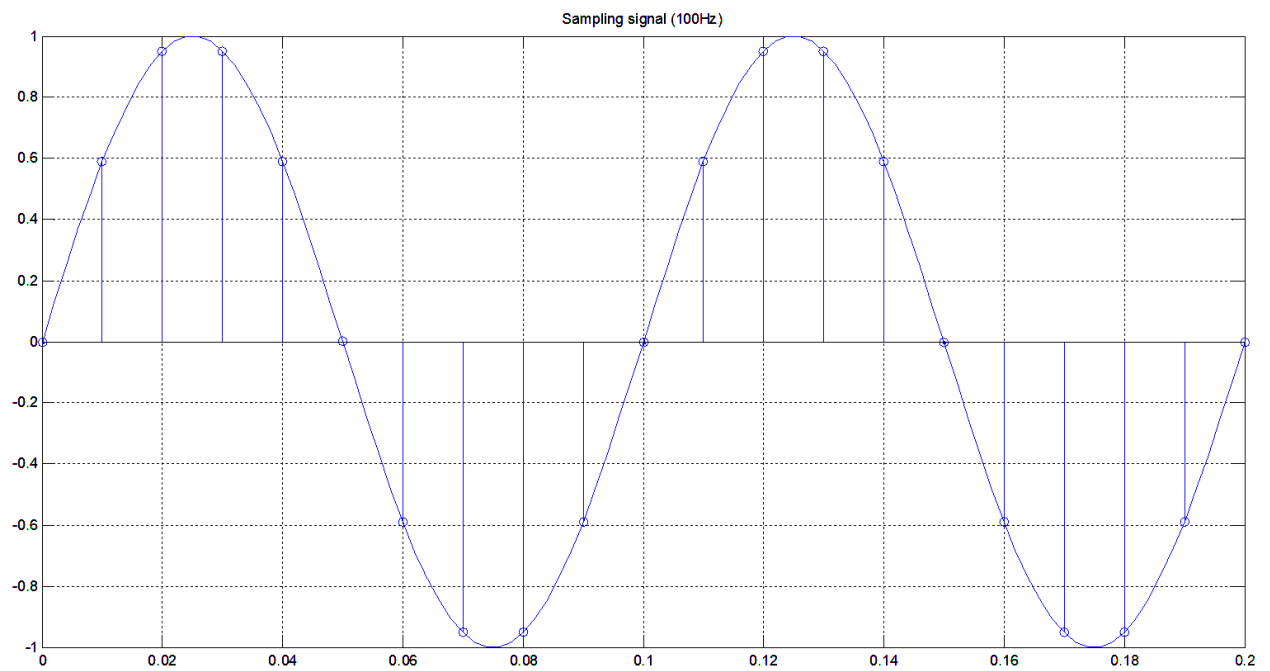


Figure: Sampled and Interpolated Signal (100Hz)

Output (10Hz) :

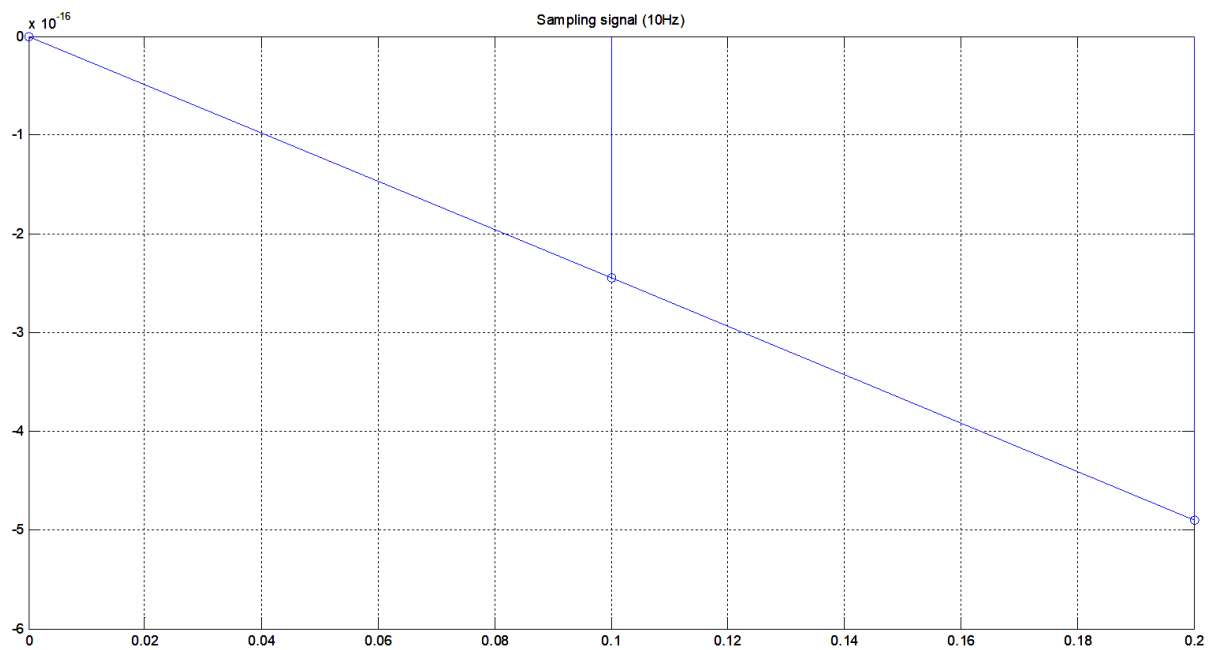


Figure: Sampled and Interpolated Signal (10Hz)

Step 05 : Generate another Signal

Given signal : $y = \sin(2\pi 10t) + \sin(2\pi 50t) + \sin(2\pi 100t)$

Code:

```
f1=10;  
f2=50;  
f3=100;  
t=0:0.001:0.2  
y = sin(2*pi*f1*t) + sin(2*pi*f2*t) + sin(2*pi*f3*t);  
figure(4)  
subplot(311)  
plot(t,y)  
grid on
```

Output:

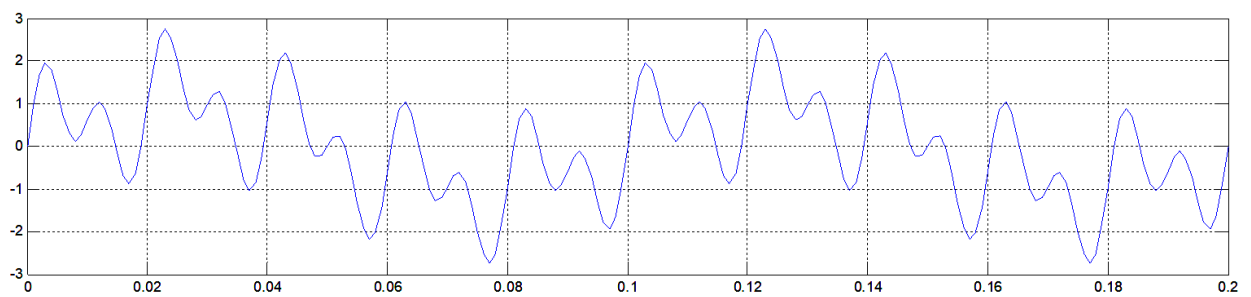


Figure: Analog Signal

Step 06 : determining the Sampling frequency for the signal in Step 05

According to the Nyquist principle, the sampling frequency must be at least 2times or greater than the highest frequency present in the message signal. So, the sampling frequency should be equal to or greater than $2 \times 100 = 200$ Hz

Code (Sampling frequency 200Hz):

```
fsD=2*f3;  
tsD=0:1/fsD:0.2;  
ysD = sin(2*pi*f1*tsD) + sin(2*pi*f2*tsD) + sin(2*pi*f3*tsD);  
tiD=0:1/fsD:0.2;  
yiD=interp1(tsD,ysD,tiD,'spline')  
figure(4)  
subplot(312);  
stem(tsD,ysD);  
title('Sampling signal (200Hz)')  
grid on  
subplot(313)  
stem(tsD,ysD);  
hold on  
plot(tiD,yiD);  
title('Interpolated signal (200Hz)')  
grid on  
ylim([-3 3])
```

Output (Sampling frequency 200Hz):

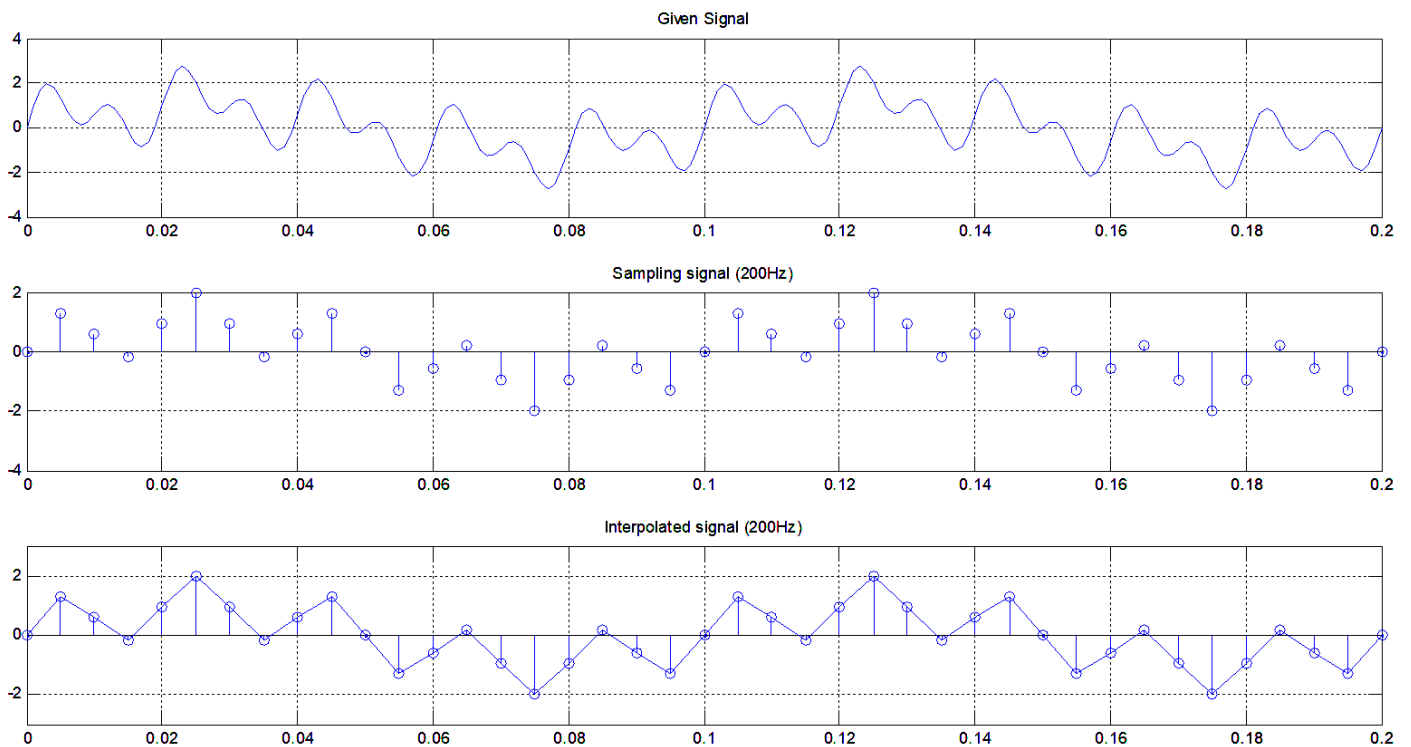


Figure: Given, Sampled and Interpolated Signal (200Hz Sampling)

If we sample below the Nyquist rate i.e. sampling at 50 Hz we would get

Code (Sampling at 50 Hz):

```
t=0:0.001:0.2
y = sin(2*pi*10*t) + sin(2*pi*50*t) + sin(2*pi*100*t);
figure(4)
subplot(311)
plot(t,y)
title('Given Signal')
grid on
fsD=50; % Sampling frequency=50Hz
tsD=0:1/fsD:0.2;
ysD = sin(2*pi*10*tsD) + sin(2*pi*50*tsD) + sin(2*pi*100*tsD);
tiD=0:1/fsD:0.2;
yiD=interp1(tsD,ysD,tiD,'spline')
figure(4)
subplot(312);
stem(tsD,ysD);
title('Sampling signal (50Hz)')
grid on
subplot(313)
stem(tsD,ysD);
hold on
plot(tiD,yiD);
title('Interpolated signal (50Hz)')
grid on
xlim([0 0.1])
```

Output (Sampling at 50 Hz):

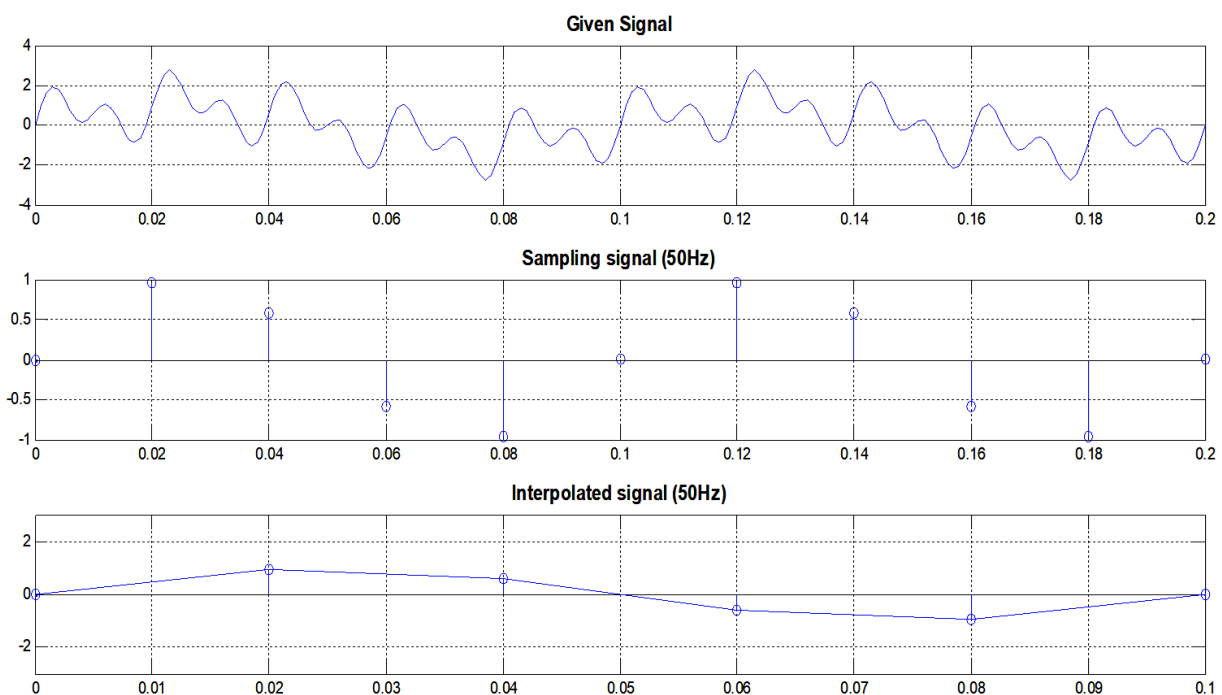


Figure: Given, Sampled and Interpolated Signal (50Hz Sampling)

If we go above Nyquist rate i.e. 1KHz we will get

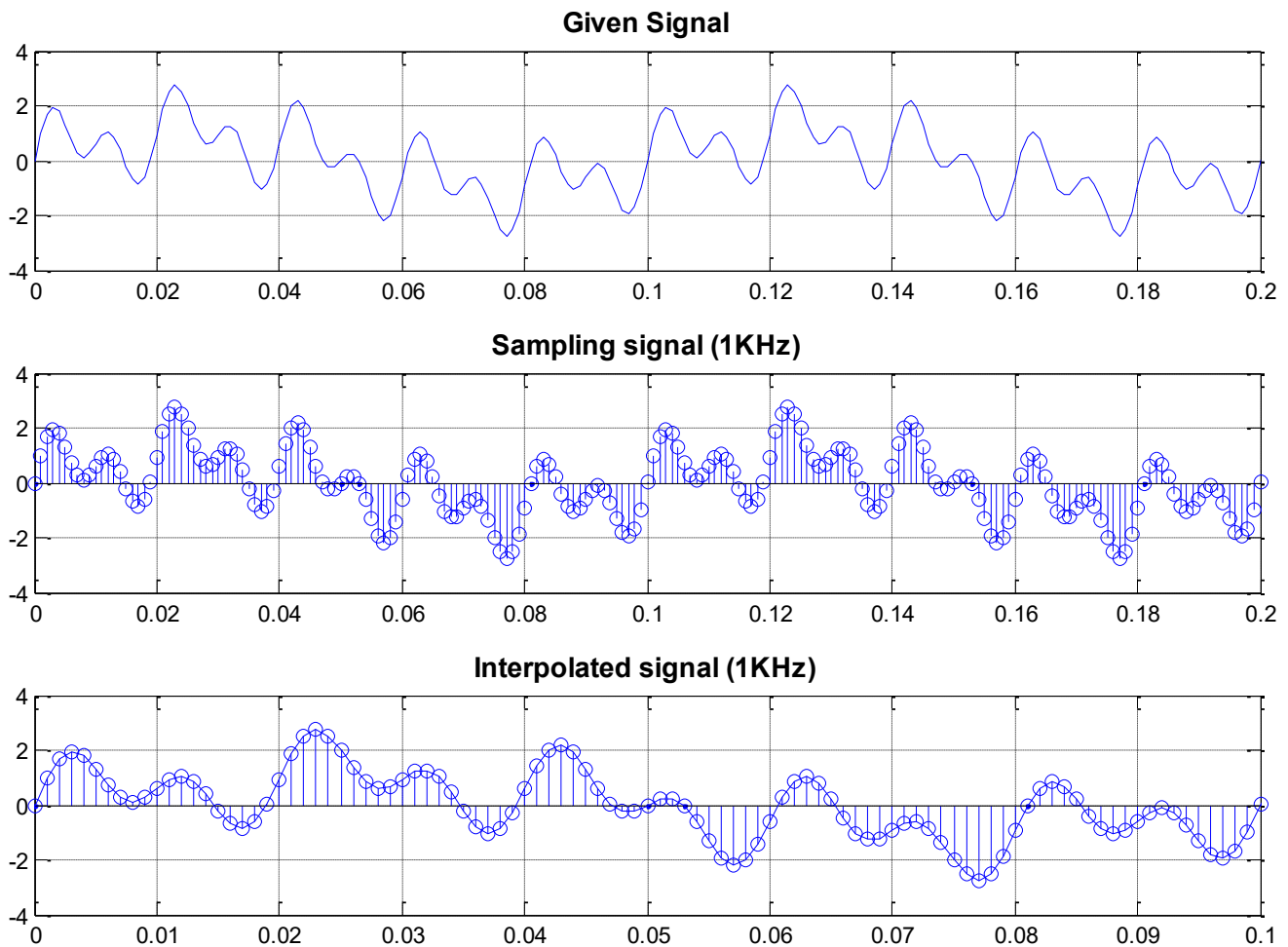


Figure: Given, Sampled and Interpolated Signal (1 KHz Sampling)

Comment on the obtained Results from Step 4 and Step 6 :

Comment on Step 04 :

Combined output from step 4 to show the actual signal, 20Hz, 50Hz, 100Hz and 10Hz version of the signal, we can look at them. The number of samples increased as we increased the sampling frequency. This played a massive role when we tried to reconstruct the signals later. From initial look, we could see more data points with high sampling frequency.

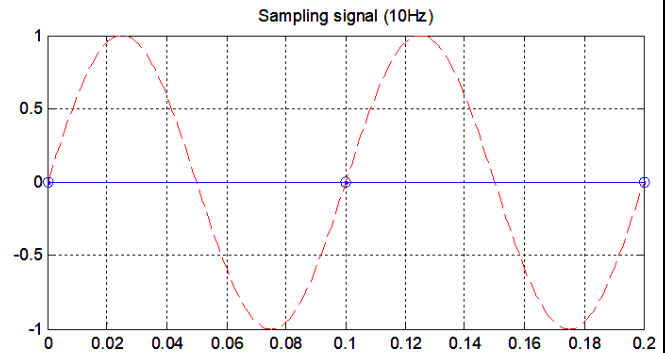
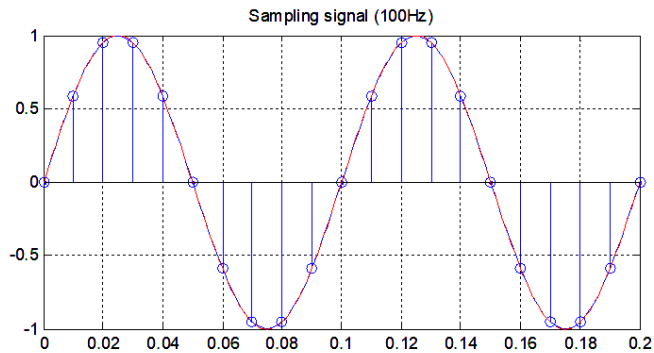
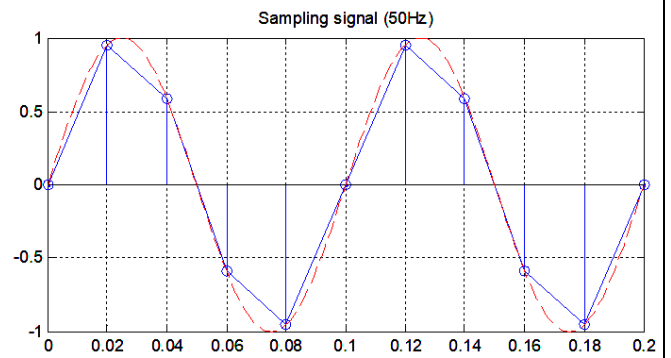
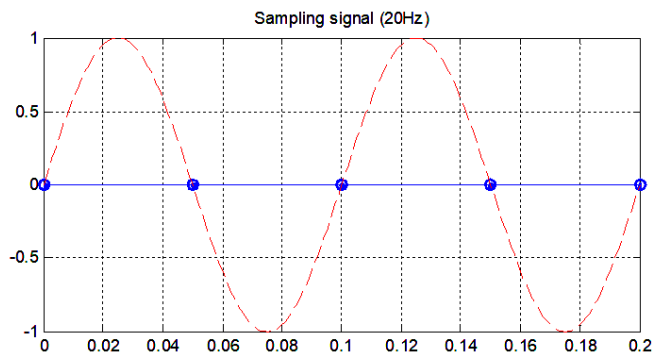


Figure : The graphs indicate sampling and reconstruction of the given signal at sampling frequencies of 20Hz,50Hz,100Hz and 10Hz respectively. (Here, **the red dashed lines represent the given signal**)

Frequency of the given signal is 10 Hz
 So Nyquist frequency = $2 \times 10\text{Hz} = 20\text{Hz}$
 i.e. the sampling frequency must be equal to or greater than 20Hz.

If we zoom into the 20Hz sampled frequency graph, we will see

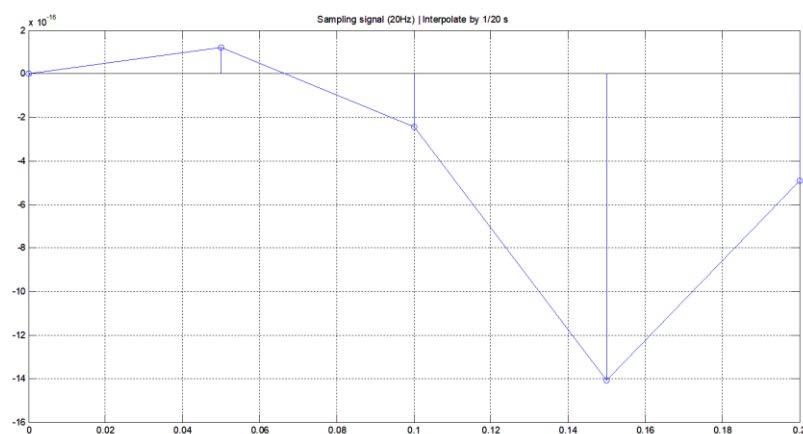


Figure: 20Hz Sampling

It demonstrates the partial or very close reconstruction of the original signal as it is equal to the Nyquist frequency.

Comment on Step 06 :

If we re-analyze the results obtained in the step 06 we will be seeing that

Maximum frequency of the given signal is 100 Hz

So Nyquist frequency = $2 \times 100\text{Hz} = 200\text{Hz}$

i.e. the sampling frequency must be equal to or greater than 200Hz.

The samples at 200Hz can reconstruct the signal as we can see in the figure added in step 06.

Then we tried 2 frequencies :

(i) 50 Hz < Nyquist Frequency

(ii) 1000 Hz > Nyquist Frequency

We can see that the 50Hz version cannot recreate the original signal resulting in a huge error. On the other hand, the 1KHz version can reconstruct the signal very accurately. This demonstrate the 2 facts:

1. Higher frequencies above the Nyquist Frequency can reconstruct more accurately
2. Exactly the Nyquist Frequency is the minimum sampling frequency that can recreate the signal.
3. Sampling below Nyquist Frequency cannot give accurate results.

3. Part B : Study of Quantization and Quantization Error

Step 01 : Source code on Matlab to realize a Uniform Quantizer

To develop the source code we need few parameter knowledge of the system which are :

- **No. of bits** : The no. of bits of the ADC
- **Level** : The no. of levels this dataset will be divided is equal to 2^{bits}
- **Step Size** : Step size is the width of each level of the signal. It can be calculated as
$$\text{Step Size} = \frac{V_{max} - V_{min}}{\text{Level} - 1}$$

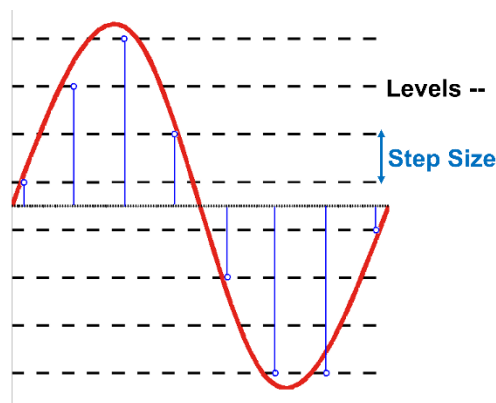


Figure: Step size and Level in Signal Processing

Level Assignment: Comparing each sample values with the terminal level values and assigning it to a certain level to quantize the data.

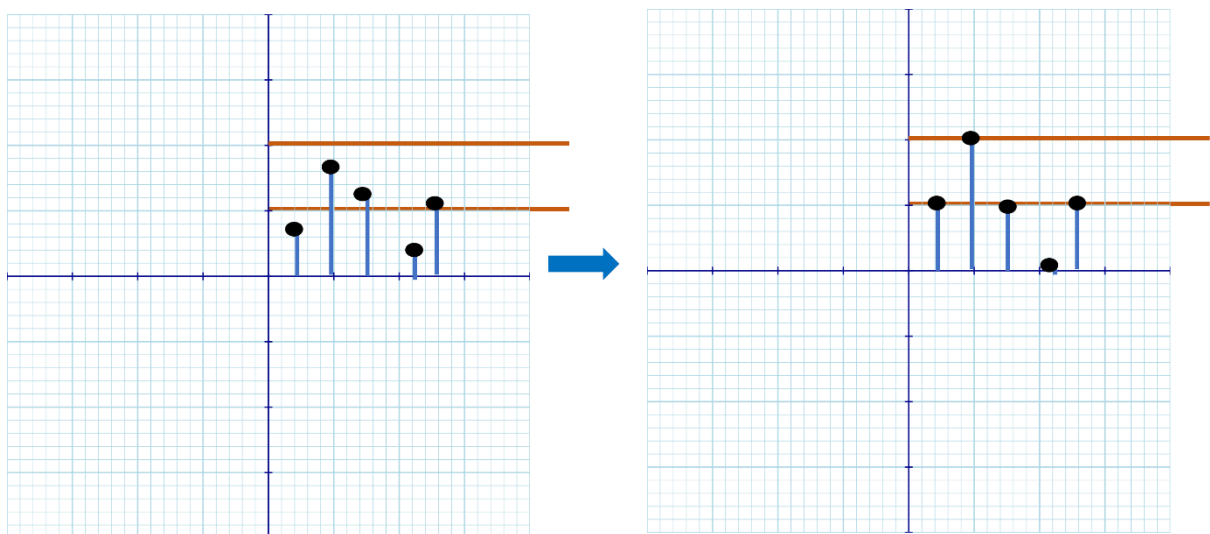


Figure: Quantization

So a sample code can be :

```
nBit=3;
Level=2^nBit
StepSize=(max(Signal)-min(Signal))/(Level-1)

for i=1:Level
    L(i)=min(Signal)+ step*(i-1);
    %First L1, then L1+step , next L1+step*2, then L1+step*3
end
```

Step 02 : Sample the signal in Step 05 of Part A in 200 Hz

Code:

```
%% Analog Signal

f1=10;
f2=50;
f3=100;
nB=3; %No. of bits
Level=2^nB;

N=2;
n1=0:0.0001:N-1;
y = sin(2*pi*f1*n1/N) + sin(2*pi*f2*n1/N) + sin(2*pi*f3*n1/N);
figure(1)
plot(n1,y);
hold on
title('Analog Signal')
hold off

%% Sampling

fsD=2*f3; % Sampling frequency fs=200Hz
n2=0:1/fsD:N-1;
ys = sin(2*pi*f1*n2) + sin(2*pi*f2*n2) + sin(2*pi*f3*n2)
figure(2)
stem(n2,ys);
title('Sampling Signal')
hold off
```


Output:

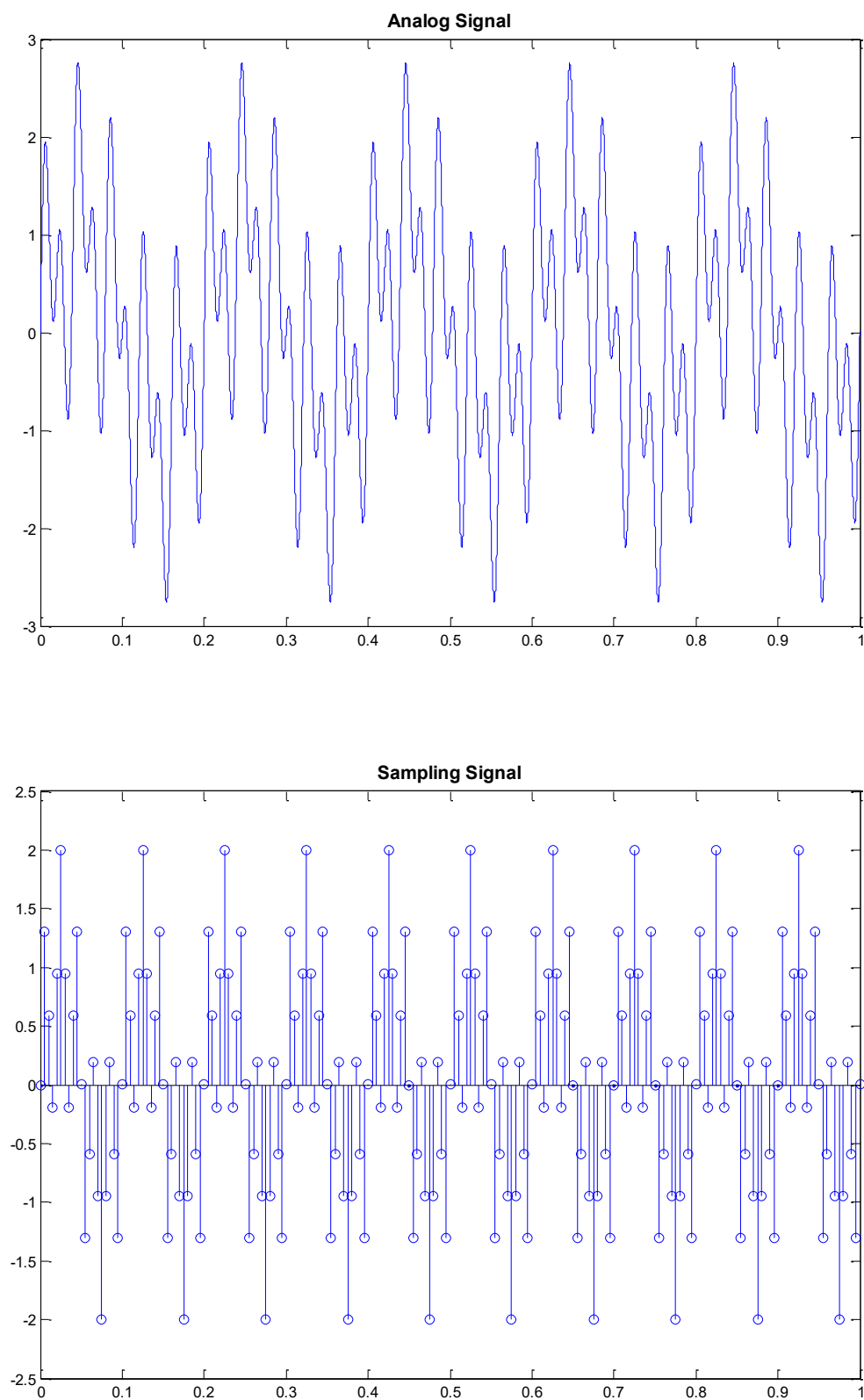


Figure: Analog and Sampling Signal

Step 03 : Quantize the Signal by 3-bit Uniform Quantizer

(Adding to the codes written in step 02)

Code :

```
%% Quantization

step=(max(ys)-min(ys))/(Level-1) %We get 1 slot less than the
level

for i=1:Level
    L(i)=min(ys)+ step*(i-1) %First L1, then L1+step , next
L1+step*2, then L1+step*3
end

yq=ys;

%Level Assignment
for m=1:length(L)-1;
    p= yq>L(m) & yq<L(m+1) & abs(yq-L(m)) < abs (yq-L(m+1));
    yq(p)= L(m);

    p= yq>L(m) & yq<L(m+1) & abs(yq-L(m)) > abs (yq-L(m+1));
    yq(p)=L(m+1);
end

figure(3)
stem(n2,yq)
hold off

figure(5)
plot(n1,y,'r');
hold on
stem(n2,ys,'m');
hold on
stairs(n2,yq)
xlim([0 0.2])
legend ('Analog Signal','Sampling Signal', 'Quantized
Signal');
title('Quantization of Analog Signal | 200Hz Sampling
Frequency')
```

Output :

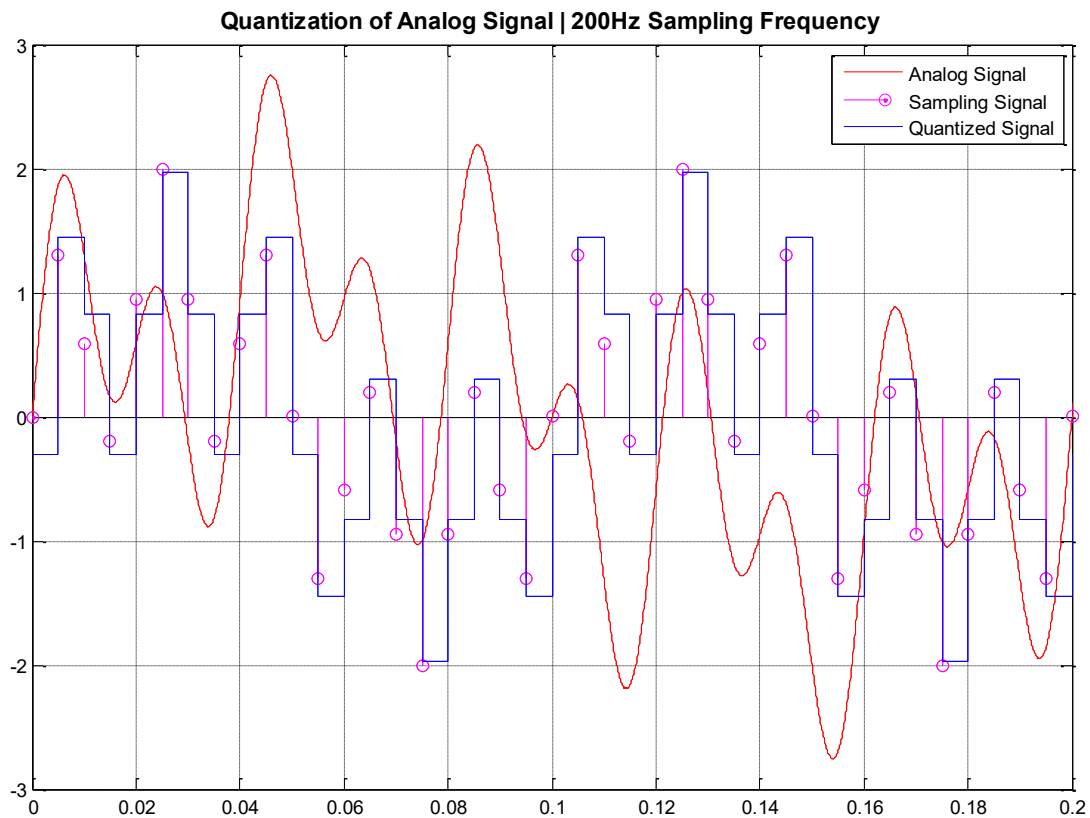


Figure: Quantization of Analog Signal (200Hz Sampling)

Step 04 : Obtaining Quantization Noise Power

Noise power at decibel scale is determined with $20 \cdot \log(\text{StepSize}^2/12)$

(Adding to the codes written in step 02 and 03)

Code:

```
%% Determination of Power
P= 20*log(step^2/12)
```

Output:

```
P =

-72.0828

fx >>
<
```

If we want to visualize the error caused on the graph, we can also do so.

Code:

(Separately coded after properly taking the given message signal)

```
yq=floor((ys+1)*2^(nB-1));  
yq=yq/(2^(nB-1)); % Quantize  
yq=yq-(2^(nB)-1)/2^(nB) % Round  
ye=ys-yq; % Quantization error  
  
figure(4)  
stem(ys);  
hold on;  
stairs(yq,'c');  
hold on;  
plot(ye,'r','Linewidth',2);  
xlim([0 200]);  
  
legend('Sampled','Quantized','Error')  
title(sprintf('Sampled, Quantized and Error for %g bits, %g  
quantization levels ',nB,2^nB));  
hold off
```

Output:

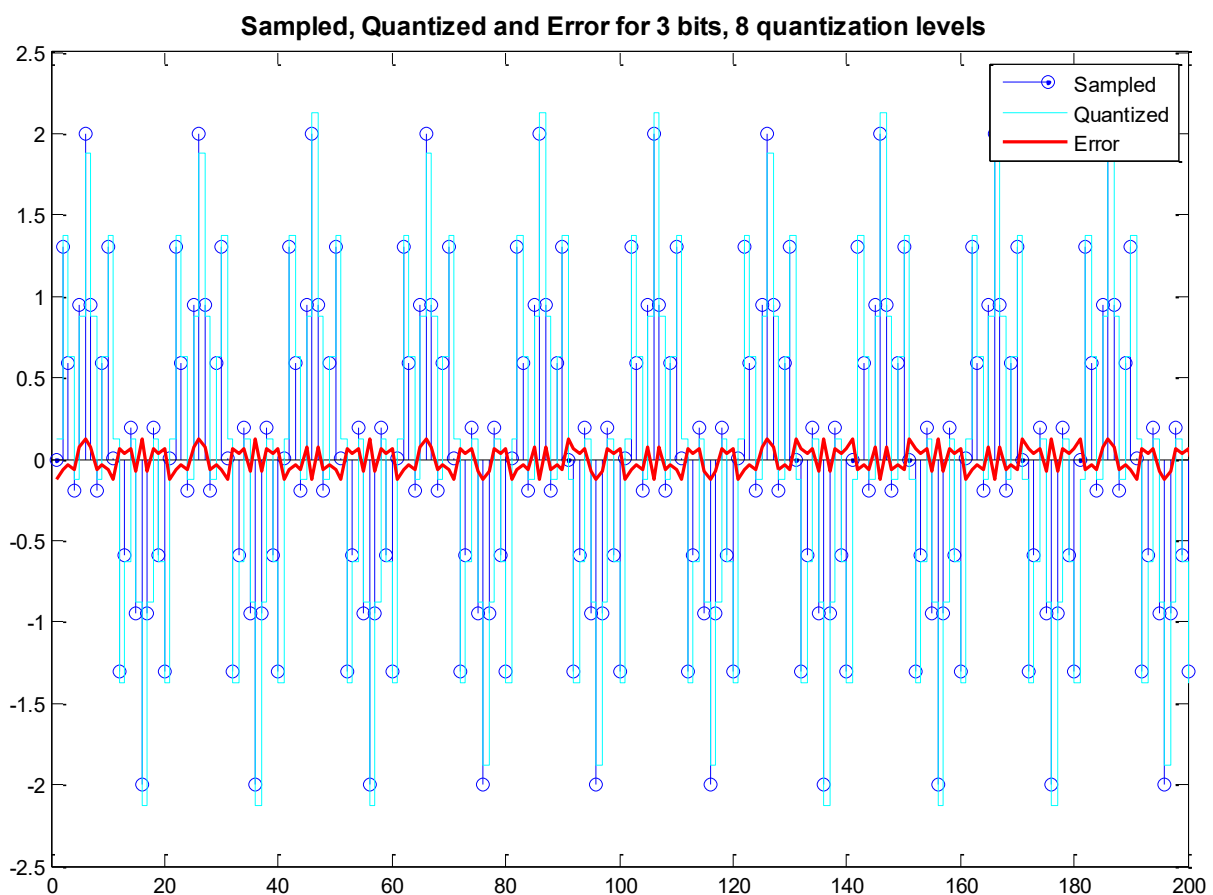


Figure: Sampled, Quantized and Error

Step 05 : Obtaining SQNR from 2 Equations

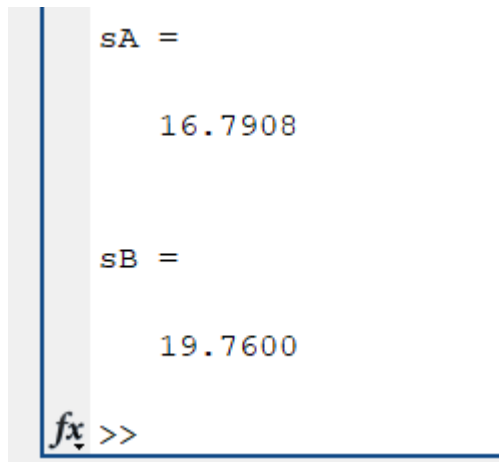
(Adding the previous codes from Step 02 and Step 03)

Code:

```
%Equation-A
sA= meansqr(ys)/((0.5^2)/12); %NOT dB value
sA=10*log10(sA)%dB Value

%Equation-B
sB=1.76+(6*nB) %dB Value
```

Output:



```
sA =
    16.7908

sB =
    19.7600

fx >>
```

Step 06 : Reconstruct the Signal by Interpolation

(Adding the previous codes from Step 02 and Step 03)

Code:

```
% Reconstructing the Signal

figure (3)
subplot(221);
plot(n1,y,'r','Linewidth',2);
title ('Original Signal')
xlim([0 0.3])
hold on

subplot(222);
stem (n2,ys,'Linewidth',2)
title ('Sampled Signal')
xlim([0 0.3])
hold on
```

```

subplot(223);
stem (n2,yq,'Linewidth',2)
title ('Quantized Signal')
xlim([0 0.3])
hold on

yr=interp1(n2,yq,n2)
subplot(224);
plot(n2,yr,'r','Linewidth',2);
title ('Reconstructed Signal')
xlim([0 0.3])
hold off

```

Output :

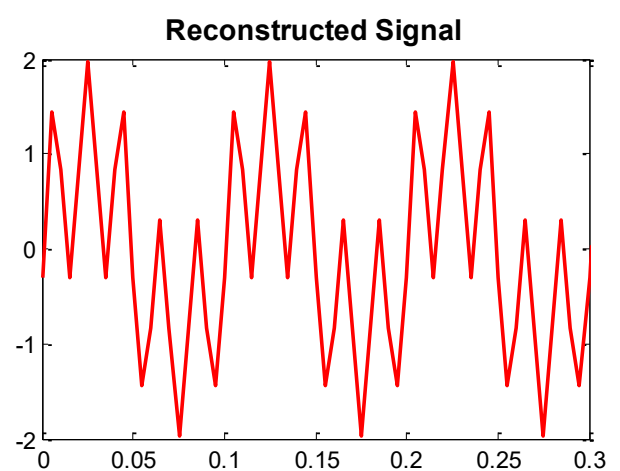
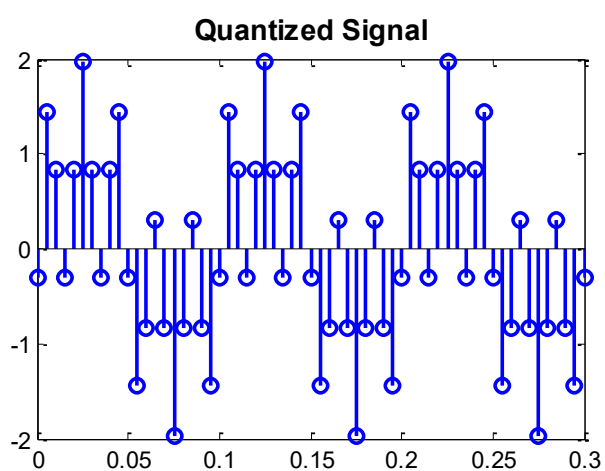
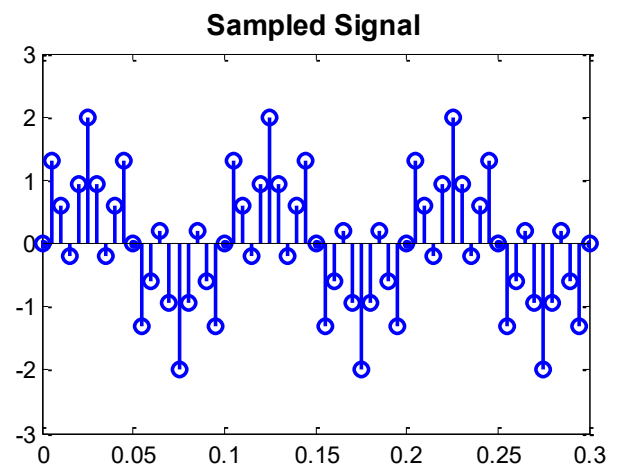
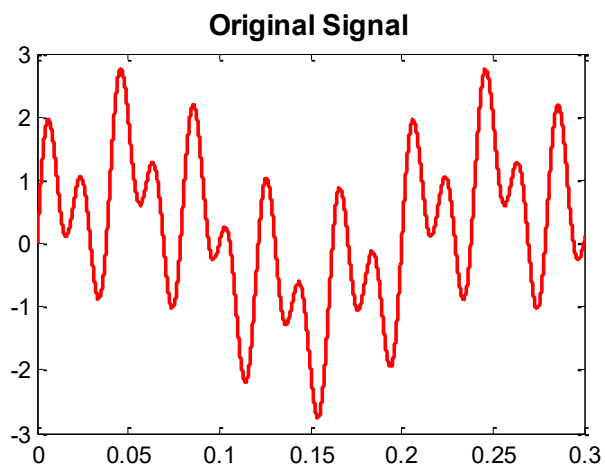


Figure: Given, Sampled, Quantized and Interpolated Signal

Step 07 : Repeat Step3 to 6 for 4 and 6 bit Uniform Quantizer

Code (for 4 bit) :

```
%% Analog Signal

f1=10;
f2=50;
f3=100;
nB=4; %No. of bits %CHANGED FROM PREVIOUS CODES
Level=2^nB;
N=2;
n1=0:0.0001:N-1;
y = sin(2*pi*f1*n1/N) + sin(2*pi*f2*n1/N) + sin(2*pi*f3*n1/N);
figure(1)
plot(n1,y);
hold on
title('Analog Signal')
hold off

%% Sampling
fsD=2*f3; % Sampling frequency fs=200Hz
n2=0:1/fsD:N-1;
ys = sin(2*pi*f1*n2) + sin(2*pi*f2*n2) + sin(2*pi*f3*n2);
figure(2)
stem(n2,ys);
title('Sampling Signal')
hold off

%% Quantization
step=(max(ys)-min(ys))/(Level-1);
for i=1:Level
    L(i)=min(ys)+ step*(i-1); %First L1, then L1+step & Cont.
end

yq=ys;

%Level Assignment
for m=1:length(L)-1;
    p= yq>L(m) & yq<L(m+1) & abs(yq-L(m)) < abs (yq-L(m+1));
    yq(p)= L(m);

    p= yq>L(m) & yq<L(m+1) & abs(yq-L(m)) > abs (yq-L(m+1));
    yq(p)=L(m+1);
end

figure(3)
stem(n2,yq)
hold off

figure(3)
```

```

plot(n1,y,'r');
hold on
stem(n2,ys,'m');
hold on
stairs(n2,yq);
xlim([0 0.2])
legend ('Analog Signal','Sampling Signal', 'Quantized
Signal');
title('Quantization of Analog Signal | 200Hz Sampling
Frequency')
grid on

%% Determination of Power
Noise_Power = 20*log(step^2/12)

%% Determination of SQNR
sA= meansqr(ys)/((step^2)/12); %NOT dB value
sqnrA=10*log10(sA) %dB Value
sqnrB=1.76+(6*nB) %dB Value

%% Reconstruct
figure (4)
subplot(221);
plot(n1,y,'r','Linewidth',2);
title ('Original Signal')
xlim([0 0.3])
hold on
subplot(222);
stem (n2,ys,'Linewidth',2)
title ('Sampled Signal')
xlim([0 0.3])
hold on
subplot(223);
stem (n2,yq,'Linewidth',2)
title ('Quantized Signal')
xlim([0 0.3])
hold on
yr=interp1(n2,yq,n2);
subplot(224);
plot(n2,yr,'r','Linewidth',2);
title ('Reconstructed Signal')
xlim([0 0.3])

```

Output :


```

Command Window
-0.1311

>> Report_TaskB_V3_4bit

step =

    0.2667

Noise_Power =

   -102.5684

sqnrA =

    22.2508

sqnrB =

    25.7600

fx >>

```

Figure: Step Size, Noise Power, SQNR for 4bit Quantizer

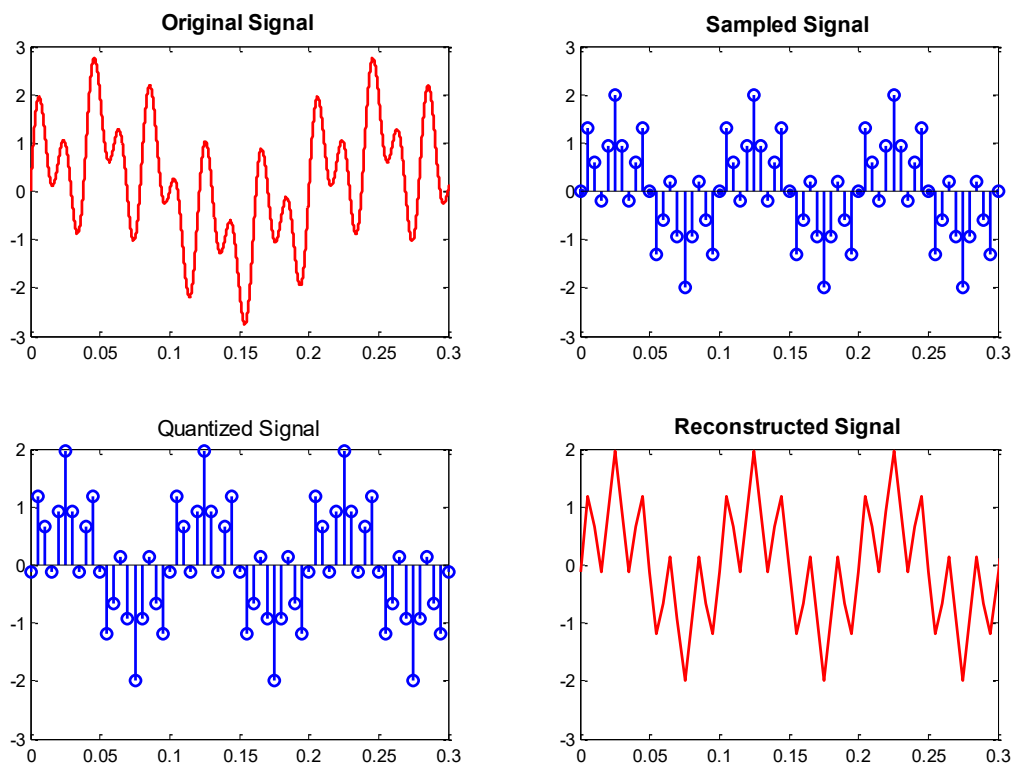


Figure: Given, Sampled, Quantized and Interpolated Signal for 4bit Quantizer

Similarly for 6 bit :

```

C:\Users\User\Desktop\CUET 3-2\EEE-366\Lab-01\Classwork
Command Window
25.7600

>> Report_TaskB_V3_6bit

step =

    0.0635

Noise_Power =

   -159.9717

sqnrA =

    34.7158

sqnrB =

    37.7600

fx >>

```

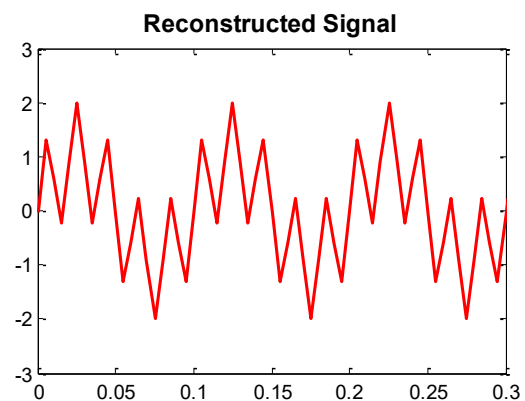
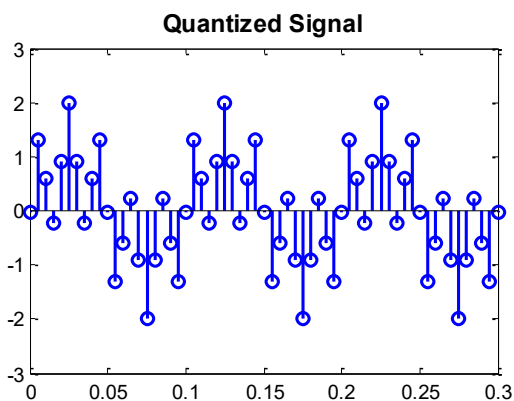
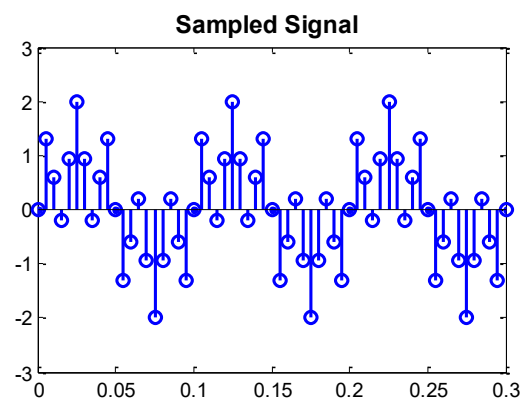
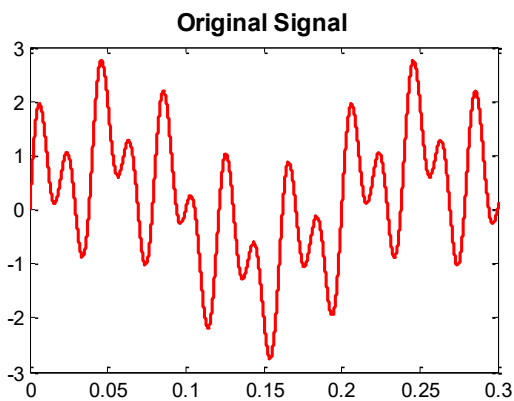


Figure: (i) Step Size, Noise Power, SQNR for 6 bit Quantizer (ii) Given, Sampled, Quantized and Interpolated Signal for 6 bit Quantizer

Comment on the result obtained in Step 06 :

In the step 06 we have basically reconstructed the original signal by interpolation where we had used 3-bit uniform quantizer. The similar thing has been replicated with 4-bit and 6-bit quantizer in action. The changes in the bit size have brought few observations :

- a. Increased bit size increases levels.
- b. The step size reduces by increasing bit.
- c. The precision of quantization improves with high bit size.
- d. The reconstruction signal improves by increased bit size.

4. Task C : Encoding the Quantized Sequence

Step 01 : Obtain sampled Quantized Data

Code:

```
f1=10;
f2=50;
f3=100;
nB=4; %No. of bits
Level=2^nB;
N=2;
n1=0:0.0001:N-1;
y = sin(2*pi*f1*n1/N) + sin(2*pi*f2*n1/N) + sin(2*pi*f3*n1/N);
figure(1)
subplot(311)
plot(n1,y);
hold on
title('Analog Signal')
hold off

%% Sampling
fsD=2*f3; % Sampling frequency fs=200Hz
n2=0:1/fsD:N-1;
ys = sin(2*pi*f1*n2) + sin(2*pi*f2*n2) + sin(2*pi*f3*n2)
figure(1)
subplot(312)
stem(n2,ys);
title('Sampling Signal')
hold off

%% Quantization
step=(max(ys)-min(ys))/(Level-1)
for i=1:Level
    L(i)=min(ys)+ step*(i-1);
end

yq=ys;
```

```

%% Level Assignment
for m=1:length(L)-1;
    p= yq>L(m) & yq<L(m+1) & abs(yq-L(m)) < abs (yq-L(m+1));
    yq(p)= L(m);

    p= yq>L(m) & yq<L(m+1) & abs(yq-L(m)) > abs (yq-L(m+1));
    yq(p)=L(m+1);
end

subplot(313)
stairs(n2,yq);
title('Quantized Signal')
hold off

```

Output :

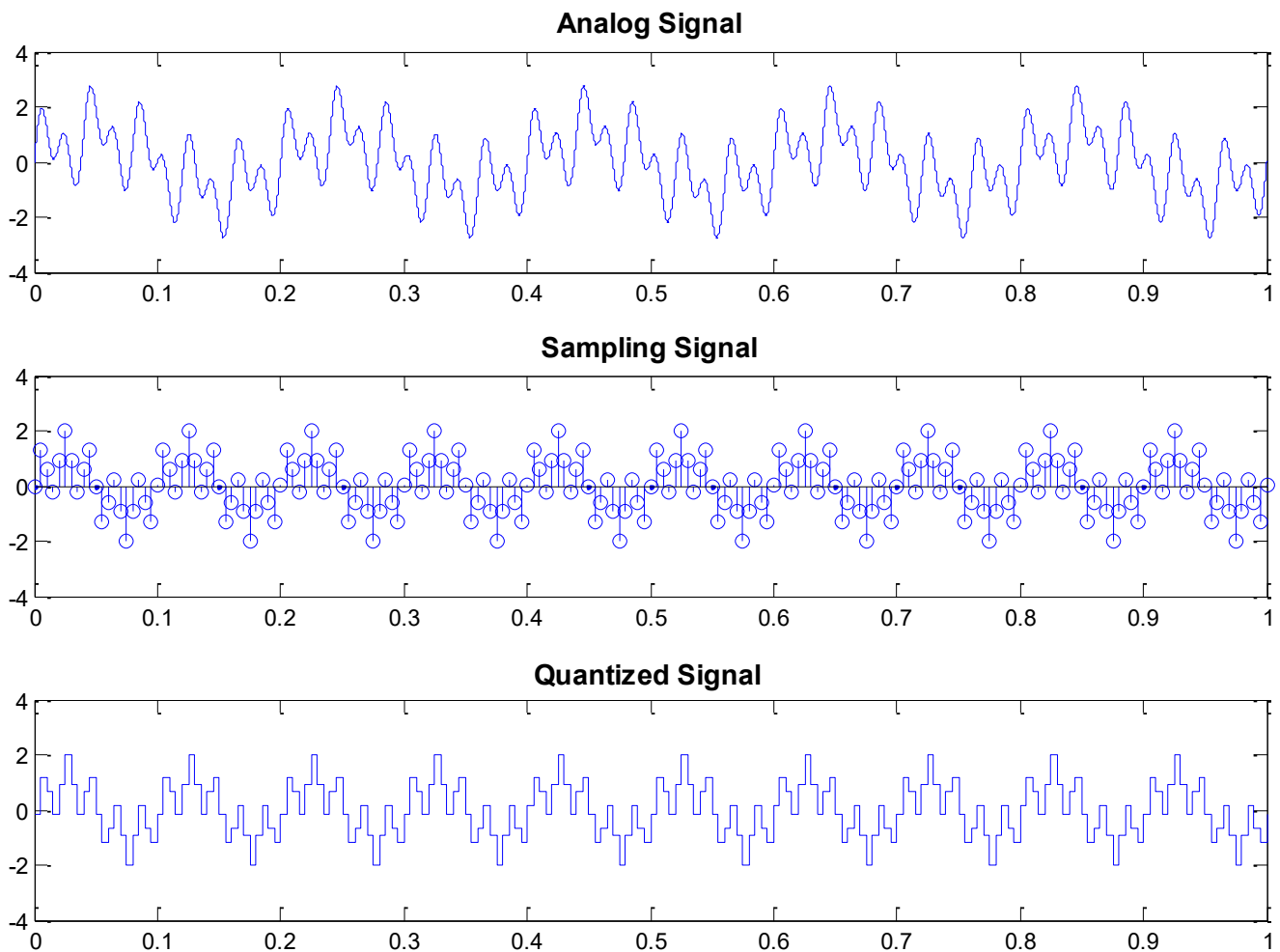


Figure: Obtained Quantized Signal

Step 02 : Assigning Binary Value

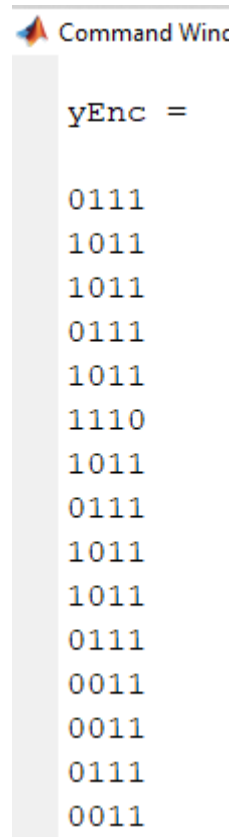
(Adding the previous step code)

Code:

```
%% Encoding
for c=1:length(yq)
    yc(c)=round(yq(c)-min(yq))/step;
end

figure(2)
Z=unique(yc);
display(Z)
yEnc=dec2bin(yc)
```

Output:



The screenshot shows the MATLAB Command Window with the following output:

```
yEnc =

0111
1011
1011
0111
1011
1110
1011
0111
1011
1011
0111
0011
0011
0111
0011
```

There are other values as well which are 4bit numbers assigned on the specific level.

Discussion:

In this work we have successfully generated, sampled, quantized, encoded the analog signals. We have also reconstructed the analog signal from its digital footprint by using the Matlab. The application of Nyquist rate have been thoroughly understood with different samples. The role of the number of bits in ADC have also been understood to a great extent.